RESEARCH

Sebastian Lang

Methoden des bestärkenden Lernens für die Produktionsablaufplanung

OPEN ACCESS

Springer Vieweg

Methoden des bestärkenden Lernens für die Produktionsablaufplanung

Sebastian Lang

Methoden des bestärkenden Lernens für die Produktionsablaufplanung

Springer Vieweg

Sebastian Lang
Magdeburg, Deutschland

Diese Publikation wurde unterstützt durch den Open-Access-Publikationsfonds der Otto-von-Guericke-Universität Magdeburg.

ISBN 978-3-658-41750-5 ISBN 978-3-658-41751-2 (eBook)
https://doi.org/10.1007/978-3-658-41751-2

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Planung/Lektorat: Stefanie Probst
Springer Vieweg ist ein Imprint der eingetragenen Gesellschaft Springer Fachmedien Wiesbaden GmbH und ist ein Teil von Springer Nature.
Die Anschrift der Gesellschaft ist: Abraham-Lincoln-Str. 46, 65189 Wiesbaden, Germany

Danksagung

Die vorliegende Dissertation entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Institut für Logistik und Materialflusstechnik (ILM) an der Otto-von-Guericke-Universität (OVGU) Magdeburg sowie am Fraunhofer-Institut für Fabrikbetrieb und -automatisierung IFF Magdeburg.

Mein Dank gilt insbesondere meinem Doktorvater Herrn Prof. Dr.-Ing. habil. Prof. E.h. Dr. h.c. mult. Michael Schenk für die Betreuung meiner Arbeit. Als Leiter der oben genannten Institute hat er meinen wissenschaftlichen Werdegang von Anfang an begleitet und mir stets den Freiraum gegeben, eigenen Forschungsinteressen nachzugehen. Durch wertvolle Diskussionen und konstruktive Hinweise hat er mich bei der Entwicklung des Dissertationsthemas maßgeblich unterstützt. Des Weiteren gilt mein großer Dank Frau Prof. Dr.-Ing. Sanja Lazarova-Molnar, Professorin und Leiterin der Forschungsgruppe Systems, Data, Simulation & Energy am Karlsruher Institut für Technologie (KIT) und Professorin an der University of Southern Denmark (SDU), für die Übernahme des Zweitgutachtens und die konstruktiven Hinweise, welche der Qualität der Arbeit und insbesondere der Stichhaltigkeit der Ergebnisdarstellung zu Gute kamen.

Ein besonderer Dank geht an Herrn Dr.-Ing. Tobias Reggelin, dem Leiter der Arbeitsgruppe Simulation am ILM, der mich seit meiner Bachelorarbeit gefördert hat. Durch ihn wurde mein Interesse in der ereignisdiskreten Modellierung und Simulation geweckt, welche das Fundament für diese Dissertation bildet.

Herzlich bedanken möchte ich mich bei meinen Kolleginnen und Kollegen am Fraunhofer IFF, die mir insbesondere in der letzten Phase meiner Dissertation den Rücken freigehalten haben. Stellvertretend und im Besonderen sind an dieser Stelle meine Vorgesetzten Herr Prof. Dr. techn. Norbert Elkmann, Leiter des Geschäftsbereichs für Robotersysteme, sowie Herr Dr.-Ing. Roland Behrens, Leiter der Arbeitsgruppe für Modellbasierte MRK-Integration und Sicherheit, zu

nennen. Ohne ihre Unterstützung hätte die Vollendung dieser Arbeit wesentlich mehr Zeit beansprucht.

Danken möchte ich meinen lieben Kolleginnen und Kollegen am ILM für das stets angenehme Arbeitsklima, spannende Diskussionen, hilfreiche Ratschläge und auch für die vielen privaten Aktivitäten außerhalb der Arbeitszeit, welche den steinigen Weg der Dissertation ein wenig erleichtert haben. Aus diesem Kreis möchte ich ganz besonders Herrn Viktor Artiushenko danken, der mich als studentische Hilfskraft in meinem Dissertationsvorhaben bei der algorithmischen Implementierung und der Durchführung von Experimenten unterstützt hat. Gleichermaßen möchte ich mich bei Herrn Falk Gerpott, Herrn Maximilian Kuetgens, Herrn Nico Lanzerath, Herrn Funing Li und Herrn Falk Niemann bedanken, die mich im Rahmen ihrer Masterarbeiten dabei unterstützten, verschiedene Bestandteile der in dieser Dissertation entwickelten Methode zu evaluieren.

Zu guter Letzt möchte ich mich vor allem bei meiner Familie, insbesondere bei meinen Eltern, sowie bei meinen Freundinnen und Freunden bedanken, die stets an mich geglaubt, mich moralisch unterstützt und bis hierhin begleitet haben.

Magdeburg Sebastian Lang
Februar 2023

Kurzfassung

Die Produktion und Logistik umfasst langfristig-strategische, mittelfristig-taktische und kurzfristig-operative Planungs- und Steuerungsaufgaben. Viele kurzfristig-operative Aufgaben lassen sich als mathematische Optimierungsprobleme formulieren, so auch die Produktionsablaufplanung, die im Fokus dieser Arbeit steht. Bei der Produktionsablaufplanung müssen Freigabetermine für Aufträge so geplant und Aufträge auf Ressourcen so verteilt werden, dass keine Kundentermine verletzt, Ressourcen bestmöglich ausgelastet und Bestände minimiert werden. Die resultierenden Optimierungsprobleme sind oftmals NP-schwer, d. h., dass exakte Berechnungsverfahren gewöhnlich nicht in annehmbarer Zeit und unter wirtschaftlichen Bedingungen anforderungsgerechte Lösungen ermitteln. In der Industrie werden stattdessen heuristische Planungsregeln, problemspezifische Heuristiken oder problemunspezifische Metaheuristiken verwendet. Heuristische Planungsregeln überzeugen durch eine einfache Anwendbarkeit und hohe Recheneffizienz, bieten jedoch oftmals eine unzureichende Lösungsqualität. Problemspezifische Heuristiken können gute Ergebnisse bei einer gleichzeitig hohen Recheneffizienz erzielen, sind jedoch in der Entwicklung und Pflege kostenintensiv und benötigen hierfür hochqualifizierte Fachkräfte mit Kompetenzen in den Bereichen Operations Research und Informatik. Metaheuristiken sind vergleichsweise einfach zu implementieren und berechnen gute Lösungen für eine Vielzahl von Optimierungsproblemen. Aufgrund ihrer iterativen stochastischen Suchstrategie benötigen Metaheuristiken u. U. eine hohe Rechenzeit, bevor eine anforderungsgerechte Lösung gefunden wird. Des Weiteren kann für eine Metaheuristik keine verlässliche Aussage getroffen werden, wie viel Zeit sie benötigt, um eine bestimmte Lösungsgüte zu garantieren.

Das bestärkende Lernen repräsentiert eine weitere Klasse potenzieller Lösungsstrategien, welche womöglich in der Lage sind, die oben geschilderten

Probleme der etablierten Lösungsverfahren zu adressieren. Die Methoden des bestärkenden Lernens eint insbesondere ein Vorteil gegenüber dem konventionellen (überwachten) maschinellen Lernen: Sie benötigen für eine Modelleingabe keine bekannte Ausgabe, um die Lösung eines Problems zu erlernen. Stattdessen werden Modellausgaben durch eine Belohnungsfunktion bewertet, sodass mittels Versuch und Irrtum Lernsignale generiert werden. Bestärkende Lernverfahren haben in jüngster Vergangenheit bei Problemen Erfolge erzielt, die Sequenzen von Entscheidungen in nichtdeterministischen Umgebungen erfordern. Hierzu zählen insbesondere klassische Brettspiele sowie zwei- und dreidimensionalen Computerspiele. Verschiedene Planungs- und Steuerungsprobleme in der Produktion und Logistik sind mit diesen Spielen insoweit vergleichbar, als dass sie sich ebenfalls als stufenartige Entscheidungsketten modellieren lassen.

Trotz der diskutierten Vorteile ist der Einsatz von bestärkenden Lernverfahren für die Produktionsablaufplanung noch kaum beforscht. Obgleich sich bereits einige Forschungsarbeiten mit Methoden des bestärkenden Lernens für bestimmte Probleme der Produktionsablaufplanung beschäftigen, existiert bisher kaum verallgemeinertes und formalisiertes Wissen hinsichtlich der Anwendung des bestärkenden Lernens für die Produktionsablaufplanung. Vor diesem Hintergrund präsentiert die vorliegende Arbeit eine Methode, welche die Adaption, Integration und den Einsatz von bestärkenden Lernverfahren für die Produktionsablaufplanung formalisiert. Um ihre Praxistauglichkeit zu gewährleisten, wird die Methode anhand von typischen Problemstellungen der Produktionsablaufplanung entwickelt und evaluiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

A	Anforderung
A2C	Advantage Actor-Critic (bestärkendes Lernverfahren)
A3C	Asynchronous Advantage Actor-Critic (bestärkendes Lernverfahren)
AC	Actor-Critic (Klasse von bestärkenden Lernverfahren)
Adam	Adaptive Moment Estimation (stochastisches Gradientenabstiegsverfahren)
ANB	Aktionsnutzen-bewertend (Klasse von bestärkenden Lernverfahren)
AOI	Automatische Optische Inspektion
B&B	Branch And Bound (lineares Optimierungsverfahren)
B&C	Branch And Cut (lineares Optimierungsverfahren)
BO	Bayes'sche Optimierung (bestärkendes Lernverfahren)
BOA	Bayes'scher Optimierungsalgorithmus (Metaheuristik)
DDPG	Deep Deterministic Policy Gradient (bestärkendes Lernverfahren)
DES	Discrete Event Simulation (ereignisdiskrete Simulation)
DP	Dynamische Programmierung (Stufenoptimierungsverfahren)
DQN	Deep Q-Networks (bestärkendes Lernverfahren)
DRL	Deep Reinforcement Learning (tiefes bestärkendes Lernen)
EDD	Earliest Due Date (Prioritätsregel für die Reihenfolgebildung)
ELU	Exponential Linear Unit (neuronale Aktivierungsfunktion)
EPA	Entscheidungspolitik-approximierend (Klasse von bestärkenden Lernverfahren)
EXP	Experiment

F-ALLOK-KNN	Künstliches Neuronales Netz zur Allokation von Auftragsfamilien (Families) zu Ressourcen
FAM_EDD	Family Earliest Due Date (Prioritätsregel für die Reihenfolgebildung)
FF	Forschungsfragen
FIFO	First In First Out (Prioritätsregel für die Reihenfolgebildung)
FJS	Flexibler Job-Shop (Ablaufplanungsproblem)
FP	Fertigungsplanungsstrategie des Unternehmens
GA	Genetischer Algorithmus (Metaheuristik)
GRASP	Greedy Randomized Adaptive Search Procedure (Metaheuristik)
GRU	Gated Recurrent Unit (rekurrentes Neuronenmodell)
HFS	Hybrider Flow-Shop (Ablaufplanungsproblem)
ILP	Integer Linear Program (ganzzahliges lineares Programm)
ISBO	Integrated Simulation-Based Optimization (problemspezifische Heuristik)
J-ALLOK-KNN	Künstliches Neuronales Netz zur Allokation von Aufträgen (Jobs) zu Ressourcen
JIT	Just-in-Time (Logistikkonzept)
KI	Künstliche Intelligenz
KNN	Künstliches Neuronales Netz
LP	Lineares Programm
LPT	Longest Processing Time (Prioritätsregel für die Reihenfolgebildung)
LST	Least Slack Time (Prioritätsregel für die Reihenfolgebildung)
LSTM	Long Short-Term Memory (rekurrentes Neuronenmodell)
MCTS	Monte Carlo Tree Search
MEP	Markov-Entscheidungsproblem
MILP	Mixed-Integer Linear Program (gemischt-ganzzahliges lineares Programm)
MIN-WL	Minimum Workload (Allokationsregel für die Zuweisung von Aufträgen zu Ressourcen)
ML	Maschinelles Lernen
MLP	Multiple Layer Perceptron (mehrlagiges Perzeptron)
NEAT	NeuroEvolution of Augmenting Topologies (bestärkendes Lernverfahren)
PMP	Parallel-Maschinen-Problem (Ablaufplanungsproblem)
POST-SEQ-KNN	Künstliches Neuronales Netz Sequenzierung von Aufträgen vor der Ressourcenallokation

PPO	Proximal Policy Optimization (bestärkendes Lernverfahren)
PPS	Produktionsplanung und -steuerung
PRE-SEQ-KNN	Künstliches Neuronales Netz zur Sequenzierung von Aufträgen nach der Ressourcenallokation
RACOS	Randomized Coordinate Shrinking (bestärkendes Lernverfahren)
ReLU	Rectified Linear Unit (neuronale Aktivierungsfunktion)
RL	Reinforcement Learning (bestärkendes Lernen)
RMSProp	Root Mean Square Propagation (stochastisches Gradientenabstiegsverfahren)
RNN	Rekurrentes Neuronales Netz
SA	Simulated Annealing
SAC	Stochastic Actor-Critic (bestärkendes Lernverfahren)
SARSA	State-Action-Reward-State-Algorithm (bestärkendes Lernverfahren)
SCR	Smallest Critical Ration (Prioritätsregel für die Reihenfolgebildung)
SMD	Surface Mount Device (Leiterplattenbestückungsmaschine)
SPT	Shortest Processing Time (Prioritätsregel für die Reihenfolgebildung)
SRACOS	Sequential Randomized Coordinate Shrinking (bestärkendes Lernverfahren)
Tanh	Tangens-Hyperbolicus (neuronale Aktivierungsfunktion)
TD3	Twin Delayed Deep Deterministic Policy Gradient (bestärkendes Lernverfahren)
TRPO	Trust Region Policy Optimization (bestärkendes Lernverfahren)
TS	Tabu Search (Metaheuristik)
TWEANN	Topology and Weight Evolving Artificial Neural Network (Klasse von bestärkenden Lernverfahren)
UML	Unified Modeling Language (vereinheitlichte Modellierungssprache)

Mathematische Notation

Für Probleme der Produktionsablaufplanung

Für die Beschreibung von Problemcharakteristiken

$\alpha \mid \beta \mid \gamma$	Kurzbeschreibung eines Produktionsablaufplanungsproblems gemäß der Notation von Graham et al. (1979)
α	Problemart (siehe Abschnitt 2.3.1.1, S. 14 f.)
β	(Liste von) Nebenbedingung(en) (siehe Abschnitt 2.3.1.2, S. 15 f.)
γ	(Liste von) Zielfunktion(en) (siehe Abschnitt 2.3.1.3, S. 16 f.)

Indizes

i, h	Indexvariablen für Ressourcen
j, k, q	Indexvariablen für Aufträge
b	Indexvariable für Auftragslose
f	Indexvariable für Auftragsfamilien
o, u	Indexvariablen für Operationen
v	Indexvariable für Operationspläne
l	Indexvariable für Produktionsstufen (oder auch Produktionsbereiche)

Auftragsbezogene Größen

J	Menge von Aufträgen (Jobs)
$\lvert J \rvert$ bzw. n	Anzahl von Aufträgen
B	Menge von Auftragslosen

$\|B\|$	Anzahl von Auftragslosen
$\|b\|$ bzw. n_b	Anzahl von Aufträgen in Auftragslos b
Fam	Menge von Auftragsfamilien
$\|Fam\|$	Anzahl von Auftragsfamilien
f_j	Familie von Auftrag j
f_{obs}	Observierte (durch den Agenten zu allozierende) Auftragsfamilie

Ressourcenbezogene Größen

L	Menge von Produktionsstufen (Levels) (oder auch Produktionsbereichen)
$\|L\|$	Anzahl von Produktionsstufen (oder auch Produktionsbereichen)
$\|M\|$ bzw. m	Anzahl von Ressourcen
$\left\|M^{(l)}\right\|$ bzw. $m^{(l)}$	Anzahl von Ressourcen in Produktionsstufe l
$i^{(l)}$	Ressource i in Produktionsstufe l
f_i bzw. $f_i^{(l)}$	Familienrüstung von Ressource i bzw. Familienrüstung von Ressource i in Produktionsstufe l
f_{SMD}	Familienrüstung der betrachtete SMD-Ressource
v_i bzw. $v_i^{(l)}$	Bearbeitungsgeschwindigkeitsfaktor von Ressource i bzw. von Ressource i in Produktionsstufe l
$W_i^{(l)}$	Arbeitslast (Workload) von Produktionsressource i in Produktionsstufe l
$W_\sigma^{(l)}$	Standardabweichung σ der Arbeitslast der Ressourcen in Produktionsstufe l
$\hat{W}_i$	Geschätzte Arbeitslast von Produktionsressource i
W_0 bzw. $W_0^{(l)}$	Nichtallozierte Arbeitslast bzw. nichtallozierte Arbeitslast in Produktionsstufe l

Zeitbezogene Größen

OP_j	Menge von Operationsplänen für Auftrag j
$\|OP_j\|$	Anzahl von Operationsplänen, die Auftrag j alternativ durchlaufen kann
$OP_{j,v}$	Operationsplan v von Auftrag j
O_j	Menge von Operationszeiten für Auftrag j
$\|O_j\|$	Anzahl von Operationen, die an Auftrag j durchgeführt werden können

o_j bzw. $o_j^{(l)}$	Bearbeitungszeit von Operation o von Auftrag j bzw. von Operation o von Auftrag j in Produktionsstufe l
$o_{i,j}$ bzw. $o_{i,j}^{(l)}$	Bearbeitungszeit von Operation o von Auftrag j auf Ressource i bzw. von Operation o von Auftrag j auf Ressource i in Produktionsstufe l
$o_{j,v}$ bzw. $o_{j,v}^{(l)}$	Bearbeitungszeit von Operation o aus Operationsplan v von Auftrag j bzw. von Operation o aus Operationsplan v von Auftrag j in Produktionsstufe l
$o_{i,j,v}$ bzw. $o_{i,j,v}^{(l)}$	Bearbeitungszeit von Operation o aus Operationsplan v von Auftrag j auf Ressource i bzw. von Operation o aus Operationsplan v von Auftrag j auf Ressource i in Produktionsstufe l
$\bar{o}^{(l)}$	Durchschnittliche Operationszeit über alle Aufträge in Produktionsstufe l
$o_{last}^{(l)}$	Durchschnittliche Operationszeit über die letzten fünf bearbeiteten Aufträge in Produktionsstufe l
$o_{max}^{(l)}$	Maximale Operationszeit über alle Aufträge in Produktionsstufe l
s_j bzw. $s_{j,k}$	Rüstzeit (Setup) von Auftrag j bzw. von Auftrag j in Abhängigkeit vom vorausgegangenen Auftrag k
s bzw. $s^{(l)}$	Konstante Rüstzeit bzw. konstante Rüstzeit in Produktionsstufe l
$s_{minor}^{(l)}$ bzw. $s_{major}^{(l)}$	Kleine bzw. große Rüstzeit in Produktionsstufe l
$p_{j,o}$ bzw. $p_{j,o}^{(l)}$	Gesamte Prozesszeit von Operation o von Auftrag j bzw. von Operation o von Auftrag j in Produktionsstufe $l\rightarrow$ Subsumiert neben der Operationszeit alle planbaren Zeitanteile (z. B. Rüstzeit $s_{j,k}$) des Auftrags auf der Produktionsressource
$p_{i,j,o}$ bzw. $p_{i,j,o}^{(l)}$	Gesamte Prozesszeit von Operation o von Auftrag j auf Ressource i bzw. von Operation o von Auftrag j auf Ressource i der Produktionsstufe l
$\tilde{p}_i$ bzw. $\tilde{p}_i^{(l)}$	Verbleibende Prozesszeit der aktuell in Bearbeitung befindlichen Operation auf Ressource i bzw. auf Ressource i in Produktionsstufe l
r_j bzw. $r_{j,o}$ bzw. r_b	Freigabezeit (Release) von Auftrag j bzw. von Operation o von Auftrag j bzw. von Auftragslos b
d_j bzw. $d_{j,o}$ bzw. $d_{j,v,o}$	Fertigstellungsfrist (Due Date) von Auftrag j bzw. von Operation o von Auftrag j bzw. von Operation o aus Operationsplan v von Auftrag j

D	Relativer Wertebereich der Fertigstellungsfristen (zur Bestimmung von zufälligen Fertigstellungsfristen)
w_d	Durchschnittlicher Verspätungsfaktor (zur Bestimmung von zufälligen Fertigstellungsfristen)
t_{sim}	Verstrichene Simulationszeit

Optimierungskriterien und Zielfunktionen

C_j bzw. $C_{j,o}$	Fertigstellungszeitpunkt (Completion) von Auftrag j bzw. von Operation o von Auftrag j
C_{max}	Gesamtdauer des Ablaufplans (Makespan)
F_j	Durchlaufzeit (Flowtime) eines Auftrags
F	Kumulierte Durchlaufzeit über alle Aufträge
L_j bzw. $L_{j,o}$ bzw. $L_{j,v,o}$	Unpünktlichkeit (Lateness) von Auftrag j bzw. von Operation o von Auftrag j bzw. von Operation o aus Operationsplan v von Auftrag
$L_{k,o}(i)$	Funktion, welche die erwartete Unpünktlichkeit von Operation o des k-ten Auftrags in der Warteschlange von Ressource i berechnet
L_{max}	Maximale Verspätung über alle Aufträge (Maximum Lateness)
E_j	Terminunterschreitung (Earliness) von Auftrag j
$\hat{E}_{min}$	Geschätzte untere Schranke für Verfrühung eines Auftrags
$\hat{E}_{max}$	Geschätzte obere Schranke für Verfrühung eines Auftrags
T_j	Terminüberschreitung (Tardiness) von Auftrag j
$\hat{T}_{min}$	Geschätzte untere Schranke für die Verspätung eines Auftrags
$\hat{T}_{max}$	Geschätzte obere Schranke für die Verspätung eines Auftrags
T	Gesamtverspätung über alle Aufträge (Total Tardiness)
U	Anzahl verspäteter Aufträge (Total Unit Penalty)

Für gradientenabhängiges bestärkendes Lernen

Zeitliche Größen

t, k	Zeitschritte
n	Benutzerdefinierbare Anzahl von Zeitschritten, nach welcher die Ziel-Modelle des bestärkenden Lernverfahrens aktualisiert werden (für Verfahren, die mittels Temporal-Difference Learning über einen Zeitschritt lernen) bzw. über welche eine Trajektorie zur Berechnung von Lernsignalen gebildet wird (für Verfahren, die mittels N-Step-Bootstrapping lernen)
T	Finaler Zeitschritt
$ENDE_{t+1}$	$= 1$, wenn die Umgebung in $t + 1$ terminiert, sonst $= 0$

Zustandsbezogene Größen

$\mathcal{S}$	Menge von Zuständen (States) einer Umgebung
s	Ein beliebiger Zustand (Variable)
S_t	Beobachteter Zustand in t
S_T	Finaler Zustand, in welchem die Umgebung terminiert
$\eta(s)$	Anzahl Beobachtungen von s
$\mu(s)$	Eintrittswahrscheinlichkeit von s

Aktionsbezogene Größen

$\mathcal{A}$	Menge von Aktionen, zwischen denen ein Agent auswählen kann
$\mathcal{A}(S_t)$	Menge verfügbarer Aktionen in S_t, zwischen denen der Agent wählen kann
a	Eine beliebige Aktion (Variable)
a_t bzw. a_k	Eine beliebige Aktion, die in t bzw. k auswählbar ist (Variable)
A_t	Gewählte Aktion in t

Belohnungsbezogene Größen

$\mathcal{R}(s, a)$	Belohnungsfunktion, welche die Auswahl von a in s bewertet
$\mathcal{R}_C(j)$	Belohnungsfunktion in Abhängigkeit von der Fertigstellungszeit der aktuellen Operation des selektierten Auftrags j

$\mathcal{R}_F(j)$	Belohnungsfunktion in Abhängigkeit von der Durchlaufzeit oder der Wartezeit der aktuellen Operation des selektierten Auftrags j
$\mathcal{R}_{\mathbb{E}[C]}(i,k)$	Belohnungsfunktion in Abhängigkeit von der geschätzten Fertigstellungszeit deraktuellen Operation des k-ten Auftrags in der Warteschlange von Ressource i
$\mathcal{R}_L(j)$ bzw. $\mathcal{R}_{\mathbb{E}[L]}(j)$	Belohnungsfunktion in Abhängigkeit von der Unpünktlichkeit bzw. geschätzten Unpünktlichkeit der aktuellen Operation von Auftrag j
$\mathcal{R}_{Ls}(j)$ bzw. $\mathcal{R}_{\mathbb{E}[Ls]}(j)$	Belohnungsfunktion in Abhängigkeit von der skalierten Unpünktlichkeit bzw. geschätzten skalierten Unpünktlichkeit der aktuellen Operation von Auftrag j
$\mathcal{R}_T$	Belohnungsfunktion zur Minimierung der Gesamtverspätung über alle Aufträge
$\mathcal{R}_T(j)$ bzw. $\mathcal{R}_{\mathbb{E}[T]}(j)$	Belohnungsfunktion in Abhängigkeit von der Verspätung bzw. geschätzten Verspätung der aktuellen Operation von Auftrag j
$\mathcal{R}_W(i,l)$	Belohnungsfunktion in Abhängigkeit von der Arbeitslast auf der allozierten Ressource i in Produktionsbereich l
$\mathcal{R}_{W\sigma}(l)$	Belohnungsfunktion in Abhängigkeit von der Standardabweichung der Arbeitslast über alle Ressourcen von Produktionsbereich l
R_E	Konstante Belohnung für verfrüht fertiggestellte (Operationen von) Aufträge(n)
w_T	Gewichtungsfaktor für Bestrafungen von verspäteten (Operationen von) Aufträgen
$\mathcal{R}_{\text{Total}}$	Multikriterielle Belohnungsfunktion
$\mathcal{R}_C$	Belohnungsfunktion zur Minimierung der Gesamtdauer des Ablaufplans C_{max}
$\mathcal{R}_T$	Belohnungsfunktion zur Minimierung der Gesamtverspätung über alle Aufträge T
ω_C und ω_T	Gewichtungsfaktoren der Belohnungsfunktionen $\mathcal{R}_{\text{C}}$ und $\mathcal{R}_{\text{T}}$ innerhalb der multikriteriellen Belohnungsfunktion $\mathcal{R}_{\text{Total}}$

$Scale_C$ bzw. $Scale_T$	Skalierungsfaktor für die Belohnungsfunktion $\mathcal{R}_C$ bzw. $\mathcal{R}_T$
R_t	Erhaltene Belohnung in t
G_t	Gesamtbelohnung in der Zukunft ab t unter Einhaltung der aktuellen Entscheidungspolitik
G_t^*	Maximale Gesamtbelohnung in der Zukunft ab t unter Einhaltung der optimalen Entscheidungspolitik
γ	Diskontierungsrate für Belohnungen

Agentenbezogene Größen

θ	Agent bzw. Menge der trainierbaren Parameter eines Agenten
θ' bzw. θ^-	Agent mit aktualisierten bzw. vergangenen Parametern
θ_A	Actor-Modell bzw. Menge der trainierbaren Parameter eines Actor-Modells
θ_C	Critic-Modell bzw. Menge der trainierbaren Parameter eines Critic-Modells
θ_{Allok}	Allokationsagent bzw. Menge der trainierbaren Parameter eines Allokationsagenten
θ_{Seq}	Sequenzierungsagent bzw. Menge der trainierbaren Parameter eines Sequenzierungsagenten
δ_A bzw. δ_C	Lernsignal des Actor- bzw. Critic-Modells

Entscheidungspolitikbezogene Größen

π	Entscheidungspolitik
$\pi(s)$	Gewählte Aktion in s unter deterministischer Entscheidungspolitik
$\pi(s;\theta)$	In Abhängigkeit von den Agentenparametern θ approximierte deterministische Entscheidungspolitik
$\pi(a\|s)$	Wahrscheinlichkeit der Auswahl von a in s unter stochastischer Entscheidungspolitik
$\pi(a\|s;\theta)$	In Abhängigkeit von den Agentenparametern θ approximierte stochastische Entscheidungspolitik
P	Durch stochastische Entscheidungspolitik parametrisierteWahrscheinlichkeitsverteilung
$J(\theta)$	Agentenparameter θ abhängige Performanzfunktion in entscheidungspolitik-approximierenden Verfahren

Adv_t	Advantage in Zeitschritt t (Vorteilhaftigkeit einer Aktion A_t ermittelt aus der Differenz zwischen dem Aktionsnutzen und Zustandsnutzen)
$SurAdv_t$	Surrogate-Advantage in Zeitschritt t (Advantage zusätzlich gewichtet mit dem Verhältnis der Auswahlwahrscheinlichkeiten einer A_t unter der aktuellen und vergangenen Entscheidungspolitik)
$\mathcal{N}(0, \sigma_{\text{rauschen}})$	Um den Wert null zentrierte Normalverteilung mit benutzerdefinierter Standardabweichung, um kontinuierliche Aktionen einer deterministischen Entscheidungspolitik mit zufälligen Werten zu verrauschen
$H(\pi(S_t; \theta))$	Entropie der in Abhängigkeit von den Agentenparametern θ approximierten stochastischen Entscheidungspolitik unter Zustand S_t
c_H	Entropie-Koeffizient

Zustandsbewertungsbezogene Größen

$v_*(s)$	Zustandsnutzenfunktion unter optimaler Entscheidungspolitik (kumulierte Belohnung in der Zukunft ab s)
$v_\pi(s)$	Zustandsnutzenfunktion unter π (erwartete kumulierte Belohnung in der Zukunft ab s)
$v_\pi(s; \theta)$	In Abhängigkeit von den Agentenparametern θ approximierte Zustandsnutzenfunktion unter π
V_t	Erwartete Gesamtbelohnung in der Zukunft für beobachteten Zustand in t (Ausgabewert von $v_*(s)$, $v_\pi(s)$ bzw. $v_\pi(s; \theta)$)
c_V	Zustandsnutzenfunktion-Koeffizient

Aktionsbewertungsbezogene Größen

$q_\pi(s, a)$	Aktionsnutzenfunktion unter π (erwartete kumulierte Belohnung bei Auswahl von a in s und nachfolgender Anwendung von π)
$q_\pi(s, a; \theta)$	In Abhängigkeit von den Agentenparametern θ approximierte Aktionsnutzenfunktion unter π
Q_t	Vektor der erwartete Aktionsnutzen für $\mathcal{A}(S_t)$ (Ausgabewert von $q_\pi(s, a)$ und $q_\pi(s, a; \theta)$)
Q_{t,a_t}	Erwarteter Aktionsnutzen für a_t (Element mit Index a_t von Q_t)
U_t bzw. U_k	Zielgröße für Aktionsnutzenwerte in t bzw. k (ermittelt durch Rückwärtsrechnung auf Basis von Q_t bzw. Q_k)

Weitere Größen

α	Lernrate für Gradientenabstiegs- und -aufstiegsverfahren
ε	Wahrscheinlichkeit für die Auswahl einer zufälligen Aktion
ϵ	Clipping-Term (nur für PPO-Algorithmus)
τ	Polyak-Faktor für graduelle Aktualisierung des Ziel-Agentenmodells

Für gradientenfreies bestärkendes Lernen

NeuroEvolution of Augmenting Topologies

D	Gesamte genetische Distanz
d_n bzw. d_s	Neuronale bzw. synaptische genetische Distanz
N_1 bzw. N_2	Anzahl neuronaler Gene des ersten bzw. des zweiten verglichenen Genoms
S_1 bzw. S_2	Anzahl synaptischer Gene des ersten bzw. des zweiten verglichenen Genoms
i, j	Indexvariablen für Gene
n_i bzw. s_i	Neuronales bzw. synaptisches Gen i
c_{sim}	Kompatibilitätskoeffizient von identischen (*»similar«*) Genen
$Dist_n(\cdot, \cdot)$	Neuronale Distanzfunktion für identische Gene
$Dist_s(\cdot, \cdot)$	Synaptische Distanzfunktion für identische Gene
c_{diff}	Kompatibilitätskoeffizient von verschiedenen (*»different«*) Genen
$Diff(\cdot, \cdot)$	Funktion, welche den Wert eins zurückgibt, wenn zwei Gene unterschiedlich sind, ansonsten null

Statistische Kenngrößen

#	Anzahl
μ	Mittelwert
σ	Standardabweichung
min	Minimum
max	Maximum
K	Kurtosis
$\bar{K}$	Mittlere normalisierte Kurtosis

Einleitung 1

In den vergangenen Jahren hat die Forschung einige bedeutende Durchbrüche im Gebiet der künstlichen Intelligenz erzielt, allen voran in den Bereichen des Bildverstehens (z. B. Krizhevsky et al. 2012) und der Sprachverarbeitung (z. B. Bahdanau et al. 2015). Die beiden angeführten Beispiele haben gemeinsam, dass sie Modelle und Algorithmen des maschinellen Lernens verwenden, um aus großen Datenmengen Wissen zu formalisieren. In der Tat lassen sich die meisten Fortschritte der letzten Jahre auf dem Gebiet der künstlichen Intelligenz auf Errungenschaften in der Beforschung von Methoden des maschinellen Lernens zurückführen (Kerting und Tresp 2019). Als Meilensteine gelten insbesondere die Methoden, welche das Anlernen vielschichtiger künstlicher neuronaler Netze erlauben und unter dem Begriff »Deep Learning« zusammengefasst werden (LeCun et al. 2015). Insbesondere in der Bild- und Spracherkennung wird als Lernstrategie das sogenannte überwachte Lernen verfolgt. Hierbei wird auf Basis einer bestimmten Anzahl von Eingaben X und einer zuordenbaren Anzahl von bekannten Ausgaben Y eine Übertragungsfunktion von X nach Y approximiert.

Die Renaissance künstlicher neuronaler Netze, deren Ursprünge bereits auf McCulloch und Pitts (1943) zurückgehen, war ab der zweiten Hälfte des letzten Jahrzehnts ebenfalls Fortschrittstreiber für ein weiteres Teilgebiet des maschinellen Lernens – das bestärkende Lernen. Die Methoden des bestärkenden Lernens eint insbesondere ein Vorteil gegenüber dem überwachten Lernen: Sie benötigen für eine Modelleingabe keine bekannte Ausgabe, um die Lösung eines Problems zu erlernen. Stattdessen werden Modellausgaben durch eine Belohnungsfunktion bewertet, sodass mittels Versuch und Irrtum Lernsignale generiert werden. Bestärkende Lernalgorithmen haben in der jüngsten Vergangenheit bei Problemen Erfolge erzielt, welche Sequenzen von Entscheidungen in nichtdeterministischen Umgebungen erfordern, bspw. in klassischen Brettspielen (Silver et al. 2017),

S. Lang, *Methoden des bestärkenden Lernens für die Produktionsablaufplanung*,
https://doi.org/10.1007/978-3-658-41751-2_1

in Atari-Videospielen (Mnih et al. 2015) oder in komplexen dreidimensionalen Echtzeitstrategie-Computerspielen (DeepMind 2019).

Verschiedene Planungs- und Steuerungsprobleme in der Produktion und Logistik sind mit den angeführten Beispielen insoweit vergleichbar, als dass sie sich ebenfalls mit Hilfe von aufeinander aufbauenden Entscheidungsketten lösen lassen. Der folgende Abschnitt diskutiert, warum Methoden des bestärkenden Lernens das Potenzial besitzen, einigen der aktuellen Entwicklungen und Herausforderungen im Bereich der Produktion und Logistik zu begegnen.

1.1 Motivation und Problemstellung

Die Produktion und Logistik umfasst langfristige strategische sowie mittelfristige und kurzfristige operative Planungs- und Steuerungsaufgaben, welche sich hinsichtlich ihrer Planungszeiträume unterscheiden. Abbildung 1.1 zeigt die Planungsmatrix nach Fleischmann (2008, S. 9), die typische Aufgaben der Produktionslogistik hinsichtlich ihres zeitlichen Planungshorizontes kategorisiert. Planungsaufgaben werden vornehmlich der langfristig strategischen und der mittelfristig operativen Ebene zugeordnet. Produktionslogistische Steuerungsaufgaben werden ausschließlich mit der kurzfristigen operativen Ebene assoziiert. Der Übergang zwischen Planungs- und Steuerungsaufgaben ist oftmals fließend.

Viele kurzfristige operative Aufgaben lassen sich als mathematische Optimierungsprobleme formulieren, wie z. B. die Produktionsablaufplanung, die im Fokus dieser Arbeit steht. Bei der Produktionsablaufplanung müssen Freigabetermine für Aufträge so geplant und Aufträge auf Ressourcen so verteilt werden, dass keine Kundentermine verletzt, Ressourcen bestmöglich ausgelastet und Bestände minimiert werden. Abhängig vom vorliegenden Produktionssystem entspricht die Produktionsablaufplanung häufig einem Flow-Shop-Scheduling-Problem (Linienfertigung) oder einem Job-Shop-Scheduling-Problem (Werkstattfertigung). In beiden Fällen handelt es sich i. d. R. um NP-schwere kombinatorische Optimierungsprobleme(Hoogeveen et al. 1992; Potts et al. 1995; Zhang et al. 2019). Derzeit wird davon ausgegangen, dass kein deterministisch arbeitender Algorithmus existiert, welcher ein Optimierungsproblem der Komplexität »NP« in Polynomialzeit lösen kann (Zimand 2004, S. 52; Hromkovič 2014, S. 190). Die Komplexitätstheorie spricht von Polynomialzeit, wenn die Berechnungszeit eines Algorithmus bei zunehmender Problemgröße nicht stärker als eine Polynomfunktion mit einem konstanten natürlichen Exponenten wächst (Sipser 2006, S. 257).

In der Praxis können lediglich Probleme von sehr geringer Größe durch Methoden der ganzzahligen linearen Optimierung exakt gelöst werden. Für größere Problemstellungen ist eine exakte Optimierung gewöhnlich nicht durchführbar.

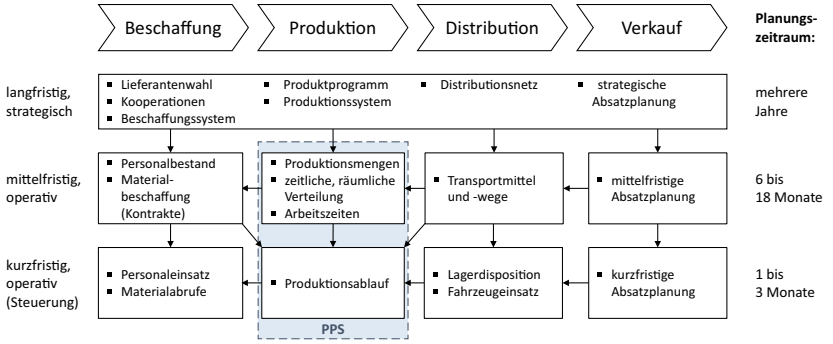


Abbildung 1.1 Planungsaufgaben in der Produktion und Logistik (Fleischmann 2008, S. 9)

In der Industrie werden stattdessen häufig einfache Prioritätsregeln für die Sequenzierung von Aufträgen eingesetzt (Schuh et al. 2012a, S. 53), welche durch eine einfache Anwendbarkeit überzeugen. Die resultierenden Auftragsreihenfolgen sind jedoch gemeinhin weit entfernt von einem optimalen Produktionsplan. Alternative Lösungsansätze bieten problemspezifische Heuristiken oder Metaheuristiken. Problemspezifische Heuristiken können in vorhersagbarer Zeit qualitativ hochwertige Lösungen konstruieren. Ihr Hauptnachteil ist, dass sie stets auf ein bestimmtes Optimierungsproblem zugeschnitten sind. Entsprechend der Diversität industrieller Produktionssysteme existiert eine Vielzahl von Heuristiken für die Planung von Auftragsreihenfolgen und Ressourcenbelegungen. So präsentieren bspw. T'kindt und Billaut (2006) über 60 verschiedene Heuristiken zur Auftragsreihenfolge- und Ressourcenbelegungsplanung für verschiedene Produktionstypen. Insbesondere für klein- und mittelständische Unternehmen sind problemspezifische Heuristiken nur schwer zugänglich, da ein umfangreiches Domänenwissen in mathematischer Modellierung sowie in der Entwicklung und Implementierung von Algorithmen erforderlich ist. Aus diesem Grund ist zunehmend der Einsatz von Metaheuristiken, wie z. B. evolutionäre Algorithmen, in der Industrie zu beobachten (vgl. Klug 2017, S. 146). Metaheuristiken zeichnen sich durch ihre generischen zufallsbasierten Suchstrategien aus, die gewöhnlich von physikalischen oder biologischen Prozessen inspiriert sind (Blum und Roli

2003). Somit können Metaheuristiken für eine Vielzahl von Optimierungsproblemen bei vergleichsweise geringem Anpassungsaufwand eingesetzt werden. Ihre stochastische Lösungssuche birgt jedoch auch Nachteile. Zum einen können Metaheuristiken eine hohe Rechenzeit beanspruchen, bevor sie zu einer optimalen Lösung konvergieren, zum anderen ist es nicht möglich vorherzusagen, wann eine Metaheuristik eine Lösung findet, die den Qualitätsanforderungen entspricht. Aus diesem Grund eignen sie sich nur bedingt zur Planungsunterstützung in hochdynamischen und stochastischen Produktionsumgebungen, bei welchen die Eingangsdaten der Optimierung kurzfristigen Änderungen unterliegen und nur schwer vorhersagbar sind. Zusammenfassend zeigt Abbildung 1.2, dass die diskutierten Planungsmethoden eine Forschungslücke hinsichtlich algorithmischer Laufzeit, Entwicklungsaufwand und Lösungsgüte aufweisen.

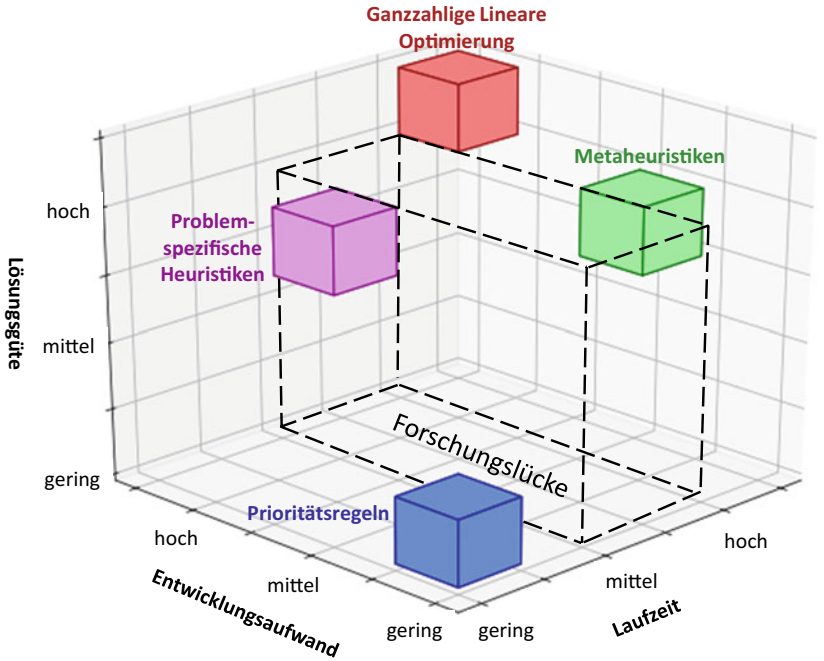


Abbildung 1.2 Klassifizierung von quantitativen Methoden der Produktionsplanung hinsichtlich Laufzeit, Entwicklungsaufwand und Lösungsgüte

Wie in Abbildung 1.1 zu erkennen war, berücksichtigt Fleischmann einen Zeitraum von einem bis drei Monaten für Planungsaufgaben der kurzfristigen operativen Ebene. Hierbei gilt es zu betonen, dass in der Wissenschaft und Praxis keine einvernehmliche Definition für die durchschnittliche Dauer von kurzfristigen operativen Planungsaktivitäten existiert. Zum Beispiel veranschlagen Schenk et al. (2014, S. 295) lediglich eine Zeitspanne von einigen Tagen bis Wochen für die Abwicklung von kurzfristigen operativen Planungs- und Steuerungsaufgaben. Die folgenden Entwicklungen und Herausforderungen weisen darauf hin, dass sich der zeitliche Horizont für die kurzfristige operative Produktionsablaufplanung weiter verringern könnte:

- **Steigende Kundenanforderungen:** Aufgrund der zunehmenden Globalisierung verschärft sich die Wettbewerbssituation zwischen Unternehmen, was sich u. a. durch einen steigenden Kostendruck und durch kürzere Innovations-, Design- und Produktlebenszyklen äußert (Koch 2017, S. 26). Insbesondere in Ländern mit hohen Personalkosten versuchen produzierende Unternehmen Kunden mit einem höheren Servicelevel an sich zu binden, um konkurrenzfähig zu bleiben. Dies äußert sich u. a. in kürzeren Lieferzeiten (Jodlbauer 2016, S. 316) oder höheren Individualisierungsmöglichkeiten für Produkte (Gräßler 2004; Lindemann et al. 2006).
- **Steigende Variantenvielfalt von Produkten:** Die steigende Anzahl zu berücksichtigender Varianten in der Produktion kann als direkte Folge der zunehmenden Individualisierungsmöglichkeiten von Produkten betrachtet werden. Beispielsweise müssen in der automobilen Endmontage bereits heutzutage 15.000 bis 20.000 verschiedene Teilevarianten durch die Produktionsplanung und -steuerung verwaltet werden (Klug 2018, S. 45).
- **Steigende Komplexität von Produkten:** Die steigende Komplexität von Produkten resultiert zum einen aus deren zunehmenden Individualisierungsmöglichkeiten. Zum anderen sind die zunehmende Integration von mechanischen, elektronischen und softwaretechnischen Produktkomponenten, für die unterschiedliche Entwicklungsanforderungen und Innovationszyklen charakteristisch sind, weitere Komplexitätstreiber (Lindemann et al. 2006, S. 1 ff.). Eine höhere Produktkomplexität kann mit einem Anstieg der Komplexität des Produktionsprozesses einhergehen, bspw. durch Zunahme der notwendigen Operationen zur Herstellung eines Produktes.
- **Just-in-Time (JIT):** Das JIT-Konzept hat zum Ziel, das richtige Material in der richtigen Menge und Qualität zur richtigen Zeit an die Produktionslinie zu liefern (Rüttimann und Stöckli 2016). Die Intention hierbei ist die Minimierung von Lagerbeständen in der Produktion (Ohno et al. 2013, S. 35). Eine

Materialanlieferung gemäß des JIT-Konzepts kann für die Produktionsplanung eine große Herausforderung darstellen, da diese bereits einige Tage vor Produktionsbeginn eine anforderungsgerechte Auftragsreihenfolge festlegen muss, um Zulieferern ein ausreichend großes Zeitfenster für die Planung und Durchführung von Materialtransporten zu gewähren (Klug 2018, S. 417 f.). Hieraus folgt, dass das JIT-Konzept sehr anfällig gegenüber Störungen ist. Konkret beeinträchtigen bspw. kurzfristige Abweichungen des Materialverbrauchs oder der avisierten Lieferzeiten die Einhaltung des JIT-Konzepts. Im ungünstigsten Fall kann der vorbestimmte Produktionsplan nicht umgesetzt werden und die verbleibende Zeit bis Produktionsstart reicht nicht aus, um einen neuen optimalen Produktionsplan zu berechnen.

- **Zunehmende Dezentralisierung der Produktion und Logistik:** Vor dem Hintergrund der ansteigenden Nachfrage nach Individualisierungsmöglichkeiten von Produkten strebt die Mehrheit der deutschen Unternehmen (vgl. Staufen AG 2019, S. 35) »Losgröße 1« an: Die vollständig kundenindividuelle Produktion zu Kosten einer Serienfertigung (Spath 2013, S. 117). Konzepte wie die Matrix-Produktion versuchen durch eine organisatorische und räumliche Dezentralisierung von Produktionsprozessen dem Trend zur Losgröße 1 entgegenzukommen. Anstelle von festverketteten Fördersystemen erfolgt der Transport von Produkten und Material durch fahrerlose Transportsysteme, wobei für jedes Produkt und nach jedem Arbeitsgang entschieden wird, welcher Prozessschritt als nächstes ausgeführt wird (Bauernhansl et al. 2014, S. 116 ff.). Einerseits resultiert hieraus eine höhere Flexibilität für die Auftragsreihenfolge- und Ressourcenbelegungsplanung, andererseits erschweren die nicht getakteten, dezentralen und oftmals unkoordinierten Steuerungsentscheidungen eine ganzheitliche Optimierung der Produktionsabläufe.

Zusammengefasst können kürzere Lieferzeiten sowie eine zunehmende Variantenvielfalt und Komplexität von Produkten zusätzliche und schwieriger zu berücksichtigende Restriktionen für die Produktionsablaufplanung verursachen. Darüber hinaus können aus Störungen in der JIT-Materialversorgung sowie aus einer zunehmenden Dezentralisierung und Enttaktung von produktionslogistischen Steuerungsentscheidungen verringerte Planungshorizonte und verringerte zeitliche Kapazitäten für die Produktionsablaufplanung resultieren. Um diesen zukünftigen Herausforderungen zu begegnen, benötigen Produktionsplaner*innen entscheidungsunterstützende Systeme, welche sensibel auf Zustandsänderungen

im Produktions- und Logistiksystem reagieren, sowie Planungs- und Steuerungsentscheidungen in ausreichend geringer Zeit zur Verfügung stellen. Aufgrund ihrer spezifischen Nachteile ist es fraglich, ob ganzzahlige lineare Optimierungsmethoden, Metaheuristiken, problemspezifische Heuristiken und Prioritätsregeln diesen Anforderungen gerecht werden.

Die dieser Forschungsarbeit zugrundeliegende Hypothese lautet, dass Methoden des bestärkenden Lernens vorteilhafte Lösungsstrategien für die Produktionsablaufplanung darstellen und sich hinsichtlich Lösungsgüte, Entwicklungsaufwand und Laufzeit zwischen den in Abbildung 1.2 dargestellten konventionellen Methoden einordnen, respektive dass

1. ein durch bestärkendes Lernen trainiertes Modell in kürzerer Zeit als Metaheuristiken,
2. bei zeitgleich geringerem Entwicklungsaufwand als für problemspezifische Heuristiken,
3. bessere Auftragsreihenfolge- und Ressourcenbelegungsentscheidungen ermittelt als Prioritätsregeln.

Eine geringere Rechenzeit im Vergleich zu Metaheuristiken wird vermutet, da beim maschinellen Lernen nur das Training einen erheblichen Zeitaufwand darstellt, während der Einsatz eines angelernten Modells lediglich eine insignifikante Laufzeit beansprucht. Der Entwicklungsaufwand gegenüber problemspezifischen Heuristiken ist mutmaßlich geringer, da einige frei verfügbare Software-Bibliotheken für das maschinelle Lernen im Allgemeinen (z. B. Abadi et al. 2016; Paszke et al. 2019) und für das bestärkende Lernen im Speziellen (z. B. Dhariwal et al. 2017; Liang et al. 2017; Hill et al. 2018) existieren, welche lediglich für Problemstellungen der Produktionsablaufplanung adaptiert werden müssen. Die Hypothese, dass durch bestärkendes Lernen trainierte Modelle bessere Auftragsreihenfolge- und Ressourcenbelegungsentscheidungen ermittelt als Prioritätsregeln, basiert darauf, dass angelernte Modelle Auftragscharakteristiken und Systemzustandsvariablen der Produktion für Steuerungsentscheidungen berücksichtigen, wohingegen Prioritätsregeln keine Informationen des Produktionssystems miteinbeziehen.

1.2 Zielstellung der Arbeit und Forschungsfragen

Trotz der diskutierten Vorteile ist der Einsatz von bestärkenden Lernstrategien für die Produktionsablaufplanung noch vergleichsweise wenig beforscht. Obgleich sich bereits einige Forschungsarbeiten mit Methoden des bestärkenden Lernens für bestimmte Probleme der Produktionsablaufplanung beschäftigen, existiert bisher kaum verallgemeinertes und formalisiertes Wissen hinsichtlich der Anwendung von bestärkenden Lernen für die Produktionsablaufplanung. Vor diesem Hintergrund soll im Rahmen dieser Arbeit eine Methode entwickelt werden, welche zur Adaption, Integration und Anwendung von bestärkenden Lernverfahren für die Produktionsablaufplanung anleitet. Hierbei soll die zu entwickelnde Methode die folgenden Forschungsfragen (FF) adressieren:

FF1: Auf welche Art und Weise können Methoden des bestärkenden Lernens für die Produktionsablaufplanung angewandt werden?
FF2: Wie können Entscheidungsprobleme der Produktionsablaufplanung als maschinelle Lernaufgabe formuliert werden, sodass eine Problemlösung mithilfe bestärkender Lernverfahren realisierbar ist?
FF3: Welche produktionslogistischen Daten sind geeignet, um Entscheidungsmodelle für die Produktionsablaufplanung mithilfe von bestärkenden Lernen zu trainieren?
FF4: Wie müssen Belohnungsfunktionen gestaltet sein, um das bestärkende Anlernen von Entscheidungsmodellen für die Produktionsablaufplanung zu steuern?
FF5: Wie müssen Methoden des bestärkenden Lernens sowie durch bestärkendes Lernen trainierte Entscheidungsmodelle mit dem Prozess der Produktionsablaufplanung sowie mit den Prozessen des zugrundeliegenden produktionslogistischen Systems integriert werden?

Um ihre Praxistauglichkeit zu gewährleisten, soll die Methode anhand von typischen Problemstellungen der Produktionsablaufplanung entwickelt werden. Die Methode soll zwei Zielgruppen adressieren: Zum einen soll sie Produktions- und Logistikplaner*innen bei der Konzeption einer auf bestärkendem Lernen basierenden Produktionsplanung und -steuerung unterstützen. Zum anderen soll sie für Wirtschafsinformatiker*innen einen Leitfaden zur Konzeption von auf bestärkenden Lernen basierenden Informationssystemen für die Produktionsablaufplanung darstellen.

1.3 Forschungsmethodik und Aufbau der Arbeit

Die Auswahl einer geeigneten Forschungsmethodik ist notwendig, um den Prozess der Forschung und Methodenentwicklung zu systematisieren und ergebnisorientiert zu gestalten. Da sich die vorliegende Arbeit im Umfeld des maschinellen Lernens und der computergestützten Produktionsablaufplanung ansiedelt, erfolgt die Auswahl einer Forschungsmethodik aus dem Bereich der Wirtschaftsinformatik. Die Wirtschaftsinformatik versteht sich als *„[…] interdisziplinäre Wissenschaft zwischen den Wirtschaftswissenschaften, insbesondere der Betriebswirtschaftslehre, und der Informatik“* (Kurbel und Strunz 1990, S. 3). Die Planung und Steuerung von Produktions- und Logistikprozessen weist in der Wirtschaftsinformatik eine lange wissenschaftliche Tradition auf (vgl. Heinrich und Ardelt 2012, S. 107 f.).

Die Forschung im Bereich der Wirtschaftsinformatik ist insbesondere durch zwei erkenntnistheoretische Paradigmen charakterisiert (Becker und Pfeiffer 2006, S. 12; Wilde und Hess 2007, S. 281; Becker et al. 2009, S. 20 f.): Die quantitativ, empirische Wirtschaftsinformatik, welche durch Beobachtungen und formalisierte Modelle informationsverarbeitende Prozesse zu erklären versucht (Holl 1999, S. 171), ordnet sich dem verhaltenswissenschaftlichen Paradigma (Behavioral Science) zu. Demgegenüber steht das gestaltungsorientierte Paradigma (Design Science), das eine wert- und nutzenorientierte Schaffung von neuen Methoden, Produkten und Technologien verfolgt (March und Smith 1995).

Da in dieser Arbeit die Entwicklung einer neuen Methode im Vordergrund steht, wird im Folgenden mit dem »Information Systems Research Framework« von Hevner et al. (2004) eine gestaltungsorientierte Forschungsmethodik adaptiert. Wie Abbildung 1.3 zeigt, basiert der von Hevner et al. (2004) adaptierte Ansatz im Wesentlichen auf drei Säulen.

Die Umwelt als linke Säule motiviert die Zielstellung der Forschung. Sie ist einerseits durch Menschen und Organisationskonzepte in Unternehmen definiert, aus deren Bedürfnissen und Anforderungen ein Forschungsbedarf resultiert, andererseits durch Technologien, welche Treiber für die Entwicklung neuartiger Informationssysteme sind. Die Motivation der Forschung wurde bereits in Abschnitt 1.1 aus menschlicher und organisatorischer Perspektive dargelegt. Im Ausblick dieser Arbeit (Abschnitt 7.2) werden zudem aktuelle technologische Entwicklungen diskutiert, welche den Einsatz von bestärkenden Lernverfahren für die Produktionsablaufplanung in Zukunft begünstigen werden.

Die Forschung als zentrale Säule beschreibt die Zielstellung der Arbeit, respektive das zu entwickelnde Artefakt sowie dessen Evaluation hinsichtlich der Bedürfnisse und Anforderungen der definierten Umwelt. Die Zielstellung der

Arbeit – die Entwicklung einer Methode zum Einsatz von bestärkenden Lernverfahren für die Produktionsablaufplanung – wurde detailliert in Abschnitt 1.2 ausgearbeitet. Kapitel 5 behandelt die Entwicklung und Erklärung der Methode. Die prototypische Evaluation der Methode wird in Kapitel 6 thematisiert.

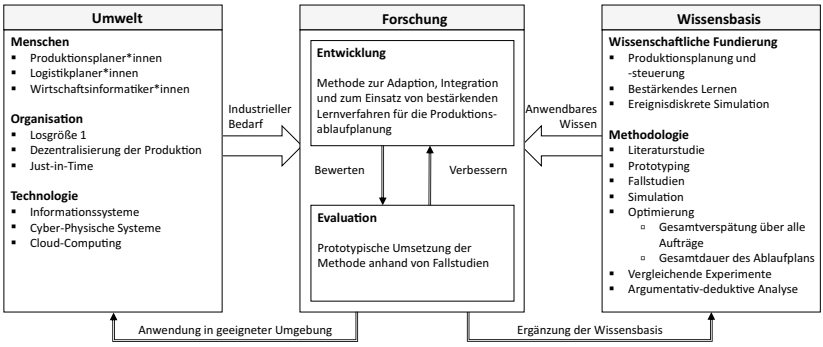


Abbildung 1.3 Adaptierte Forschungsmethodik in Anlehnung an Hevner et al. (2004)

Die Wissensbasis als rechte Säule umfasst zum einen relevante Theorien und Methoden, die den Forschungsrahmen bilden und als Grundlage der Methodenentwicklung dienen. Vor diesem Hintergrund werden in Kapitel 2 die Grundlagen der Produktionsplanung und -steuerung behandelt, mit der Zielstellung, den Anwendungsbereich von bestärkenden Lernverfahren für die Produktionsablaufplanung zu präzisieren. Die Grundlagen des maschinellen Lernens, welche für die Entwicklung und Evaluation der Methode notwendig sind, werden in Kapitel 3 dargelegt. Zum anderen beinhaltet die Wissensbasis diverse Methodologien, welche für die Evaluation der entwickelten Methode eingesetzt werden. In Kapitel 4 wird mithilfe einer Literaturstudie der aktuelle Stand der Wissenschaft und Technik hinsichtlich des Einsatzes von bestärkenden Lernverfahren für die Produktionsablaufplanung dargelegt. Ziel der Literaturstudie ist, die in Abschnitt 1.1 skizzierte Forschungslücke zu belegen und hierauf aufbauend die Anforderungen an die zu entwickelnde Methode in Kapitel 5 zu konkretisieren. Ebenfalls trägt die Literaturstudie zur Beantwortung der ersten Forschungsfrage (FF1) aus Abschnitt 1.2 bei. Die Evaluation der entwickelten Methode erfolgt unter Anwendung von traditionellen Forschungsmethoden der Wirtschaftsinformatik (Wilde und Hess 2007). Anhand von simulierten Fallstudien sollen Prototypen von Entscheidungsunterstützungssystemen, die auf bestärkenden Lernverfahren

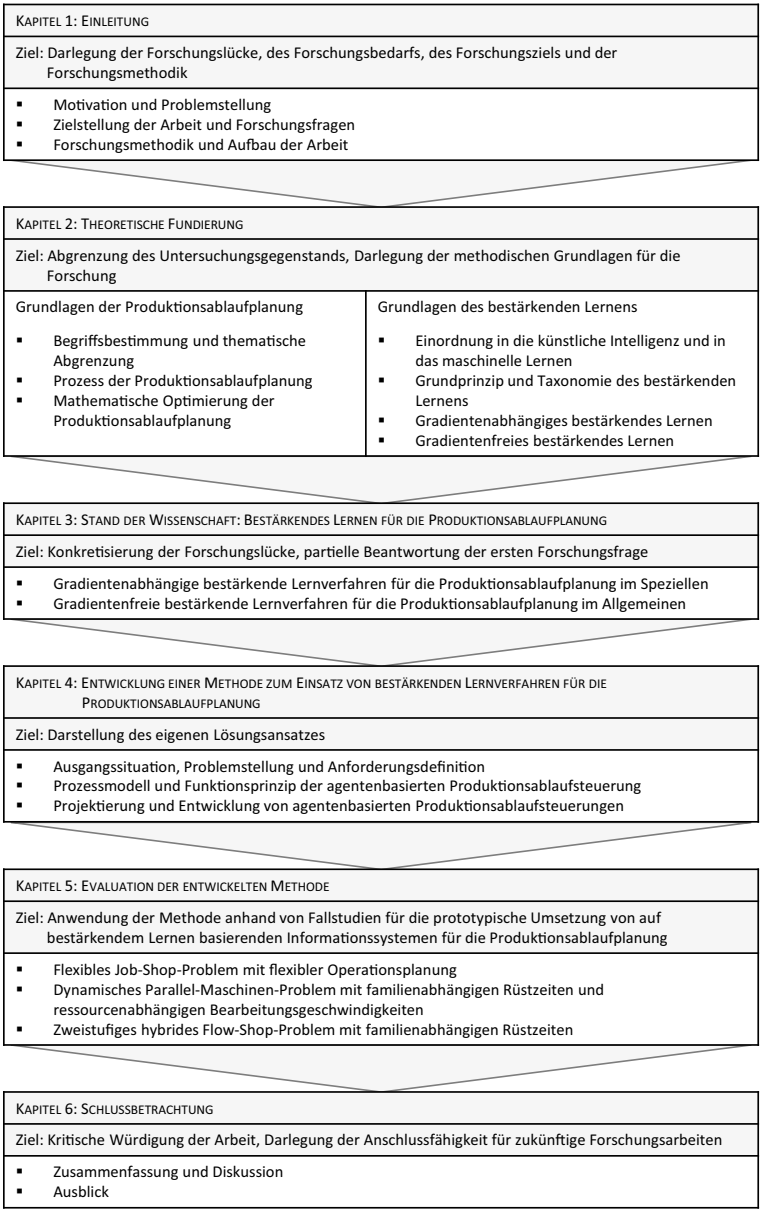


Abbildung 1.4 Vorgehen und Aufbau der Arbeit

basieren, für die Produktionsablaufplanung implementiert und mit konventionellen Lösungsmethoden verglichen werden. Konkret handelt es sich bei den simulierten Fallstudien um Optimierungsprobleme der Produktionsablaufplanung. Die Lösungsgüte der Ablaufpläne wird vornehmlich anhand der Gesamtverspätung über alle Aufträge und anhand der Gesamtdauer des resultierenden Ablaufplans bewertet. Basierend auf den Experimentergebnissen sollen mithilfe von argumentativ-deduktiven Analysen Schlussfolgerungen getroffen werden, ob die entwickelte Methode die Bedürfnisse und Anforderungen der definierten Umwelt reflektiert. Abbildung 1.4 fasst das Vorgehen und den Aufbau der Arbeit zusammen.

2 Grundlagen der Produktionsablaufplanung

Zielstellung des folgenden Kapitels ist die theoretische Fundierung der Produktionsablaufplanung. Zunächst erfolgt in Abschnitt 2.1 eine Begriffsdefinition sowie eine thematische Einordnung in die Produktionsplanung und -steuerung (PPS). Anschließend wird in Abschnitt 2.2 der Prozess der Produktionsablaufplanung im Kontext des etablierten Aachener PPS-Modells skizziert. Abschnitt 2.3 widmet sich der Produktionsablaufplanung aus Sicht der mathematischen Optimierung.

2.1 Begriffsbestimmung und thematische Abgrenzung

Die Produktionsablaufplanung ist eine Unteraufgabe der Produktionsplanung und -steuerung. Die Produktionsplanung und -steuerung (PPS) beschreibt *„[…] die räumliche, zeitliche und mengenmäßige Planung, Steuerung und Kontrolle des gesamten Geschehens im Produktionsbereich“* (Drexl et al. 1994, S. 1022). Hierbei sieht sich die Produktionsplanung in der Rolle der Ausgestaltung aller Einzelprozesse in der Fertigung und Montage, während die Produktionssteuerung den Fertigungs- und Montageablauf im Zuge der Auftragsabwicklung regelt (Schuh et al. 2012a, S. 29). Wichtige Zielstellungen der PPS sind u. a. eine hohe Termin- und Mengentreue, eine optimale Kapazitätsauslastung, kurze Durchlaufzeiten, eine minimale Kapitalbindung durch niedrige Lager- und Umlaufbestände, eine hohe Flexibilität und die Minimierung von Kosten (Schenk et al. 2014, S. 391).

Die wissenschaftliche Literatur bietet eine Vielzahl von Modellen, welche die PPS theoretisch fundieren. Eine Literaturrecherche und Übersicht zu PPS-Modellen im deutschsprachigen Raum bieten Meudt et al. (2017). Obgleich in den verschiedenen Modellen die Aufgaben der PPS zeitlich und organisatorisch ähnlich strukturiert sind, existiert kein modellübergreifender Standard hinsichtlich der Verwendung von Begriffen. Vor diesem Hintergrund wird der

S. Lang, *Methoden des bestärkenden Lernens für die Produktionsablaufplanung*,
https://doi.org/10.1007/978-3-658-41751-2_2

Begriff der Produktionsablaufplanung nicht in allen PPS-Modellen verwendet. In der Literatur dienen u. a. die Begriffe »Auftragssteuerung« (Zäpfel 1982), »Feinsteuerung« (Scheer 1995), »Fertigungssteuerung« (Lödding 2016), »Eigenfertigungssteuerung und -planung« (Schotten 1998; Luczak und Eversheim 1999; Schuh und Stich 2012a) oder »Ablaufplanung« (Kistner und Steven 2001; Siepermann 2013) als Synonyme, wobei ebenfalls keine vollständige Kongruenz zwischen den Definitionen der verschiedenen Begriffe besteht.

Alle genannten Begriffe haben jedoch gemeinsam, dass sie einen Teilprozess der PPS beschreiben, der auf der kurzfristigen operativen Entscheidungsebene an der Schnittstelle zwischen der Produktionsplanung und der Produktionssteuerung stattfindet (vgl. Meudt et al. 2017). Im Rahmen dieser Arbeit wird der Begriff »Produktionsablaufplanung« (Dyckhoff und Spengler, S. 241) verwendet, um diesen Prozess zu benennen. Diese Entscheidung fußt zum einen darauf, dass der Begriff »Ablaufplanung« als Übersetzung des englischen und international etablierten Begriffs »Scheduling« verwendet wird (vgl. Nebl 1997, 341 f.; Adam 2001, 535 f.), zum anderen darauf, dass der Begriff »Produktion« den betriebswirtschaftlichen Charakter der Ablaufplanung besser widerspiegelt als bspw. der technisch orientierte Begriff der Fertigung. Im Rahmen dieser Arbeit wird die Produktionsablaufplanung wie folgt definiert.

Arbeitsdefinition: Produktionsablaufplanung
In Orientierung an Dyckhoff und Spengler (2007, S. 241), Schuh et al. (2012b, S. 188 ff.), Meudt et al. (2017) sowie Wiendahl und Wiendahl (2019, S. 286 f.)
Die Produktionsablaufplanung im engeren Sinne umfasst die Zuordnung von Produktionsaufträgen oder Produktionslosen zu verfügbaren Produktionsressourcen, im Folgenden »Ressourcenbelegungsplanung« genannt, sowie die Reihenfolge bildung der durchzuführenden Arbeitsumfänge an Produktionsaufträgen oder Produktionslosen auf der zugeordneten Produktionsressource, im Folgenden »Auftragsreihenfolgeplanung« genannt. Im weiteren Sinne umfasst die Produktionsablaufplanung ebenfalls die Konsolidierung von Produktionsaufträgen zu Produktionslosen, im Folgenden »Losgrößenplanung« genannt, welche vor der Ressourcenbelegungs- und Reihenfolgeplanung stattfindet, jedoch nicht für alle Auftragsabwicklungstypen erforderlich ist. Das Ergebnis der Produktionsablaufplanung ist der Produktionsablaufplan, der definiert, zu welchem Zeitpunkt welcher Auftrag bzw. welches Produktionslos auf welcher Ressource bearbeitet wird.

Diese Definition ist den Zwecken und dem Forschungsgegenstand dieser Arbeit dienlich, vernachlässigt jedoch einige Aktivitäten, die manche PPS-Modelle der kurzfristigen operativen Entscheidungsebene zuordnen. Hierunter fällt insbesondere der Prozess der Auftragsfreigabe, welcher u. a. von Corsten und

Gössinger (2016) sowie Wiendahl und Wiendahl (2019) der Produktionssteuerung zugerechnet wird. Im Rahmen der Auftragsfreigabe werden die Reihenfolge und die Zeitpunkte zur Einlastung von Aufträgen in das Produktionssystem festgelegt (Lödding 2016, S. 8). Die Vernachlässigung von Auftragsfreigabeprozessen ist im Rahmen dieser Arbeit zulässig, da durch Beispiele der Auftragsreihenfolgeplanung, der Einsatz von bestärkenden Lernverfahren an sehr ähnlich gearteten Problemen (Sequenzierungsproblem) demonstriert wird. Gewonnene Erkenntnisse hinsichtlich der Modellierung, des Einsatzes und der Lösungsqualität von bestärkenden Lernverfahren werden vor diesem Hintergrund für andere Sequenzierungsprobleme der PPS weitgehend übertragbar sein. Abbildung 2.1 zeigt ein Gantt-Diagramm eines beispielhaften Produktionsablaufplans, welcher alle Charakteristiken der Arbeitsdefinition widerspiegelt.

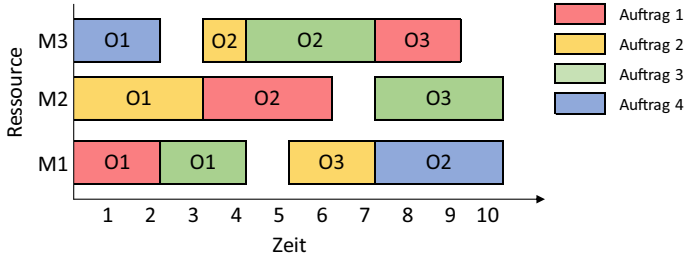


Abbildung 2.1 Beispielhafter Produktionsablaufplan für vier Aufträge mit jeweils zwei oder drei auszuführenden Operationen verteilt auf drei Produktionsressourcen

In diesem Beispiel werden vier Aufträge auf drei Produktionsressourcen M1, …, 3 gefertigt. Die Produktionsressourcen sind der Ordinate des Diagramms zugeordnet, während auf der Abszisse die Zeit abgetragen ist. Die Fertigung der Aufträge 1 bis 3 erfordert die Durchführung von jeweils drei Operationen O1, …, 3. Die Fertigung von Auftrag 4 erfordert lediglich die Operationen O1 und O2. Die Balkensegmente geben Aufschluss darüber, zu welchem Zeitpunkt auf welcher Ressource welche Operation von welchem Auftrag gestartet wird, andauert und endet. Dem Beispiel liegt die Annahme zugrunde, dass alle Operationen von allen Aufträgen auf allen Ressourcen durchgeführt werden können. Sofern vorher eine Losgrößenplanung stattfindet, würden in Abbildung 2.1 die Operationen von Produktionslosen anstatt von einzelnen Aufträgen abgetragen werden.

2.2 Prozess der Produktionsablaufplanung

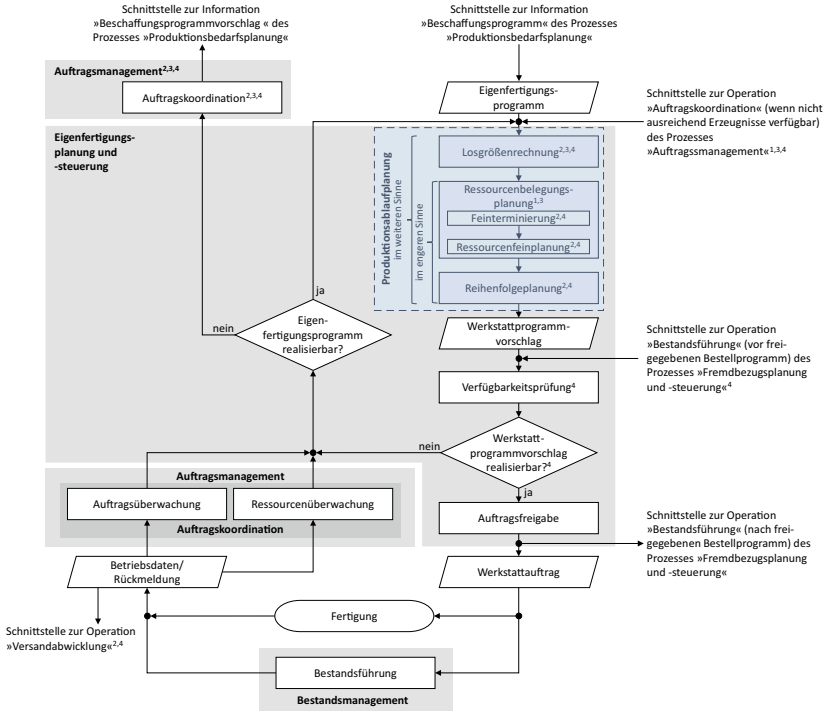
Für ein besseres Verständnis und zur Präzisierung des Forschungsgegenstands soll im Folgenden das Vorgehen der Produktionsablaufplanung im Rahmen eines etablierten PPS-Modells erläutert werden. Zu diesem Zweck bezieht sich diese Arbeit auf das Aachener PPS-Modell, welches ursprünglich von Schotten (1998) sowie Luczak und Eversheim (1999) erdacht und von Schuh und Stich (2012a, 2012b) weiterentwickelt wurde. Eine Orientierung am Aachener PPS-Modell ist naheliegend, da es insbesondere im deutschsprachigen Raum ein etabliertes Modell zur Gestaltung von PPS-Systemen darstellt (Lödding 2016, S. 6; Meudt et al. 2017). Im Folgenden wird das Prozessmodell der Eigenfertigungsplanung und -steuerung aus dem Aachener PPS-Modell herangezogen, um das Vorgehen der Produktionsablaufplanung zu erklären. Gemäß Meudt et al. (2017) siedelt sich die Eigenfertigungsplanung und -steuerung im Aachener PPS-Modell an der Schnittstelle zwischen der Produktionsplanung und der Produktionssteuerung an, was den Betrachtungsbereich dieser Arbeit am besten widerspiegelt. Das Aachener PPS-Modell definiert hierbei für verschiedene Auftragsabwicklungstypen unterschiedliche Prozessmodelle, im Einzelnen für den Auftragsfertiger (Schuh et al. 2012b, S. 150 ff.), für den Rahmenauftragsfertiger (Schuh et al. 2012b, S. 164 f.), für den Variantenfertiger (Schuh et al. 2012b, S. 178 f.) sowie für den Lagerfertiger (Schuh et al. 2012b, S. 188 ff.)

Eine Erläuterung aller Prozessmodelle würde über den Rahmen dieser Arbeit hinausgehen. Aus diesem Grund präsentiert Abbildung 2.2 ein aggregiertes Prozessmodell der Eigenfertigungsplanung und -steuerung, welches die Charakteristiken aller Auftragsabwicklungstypen vereint. Das aggregierte Prozessmodell basiert im Wesentlichen auf dem Prozessmodell der Eigenfertigungsplanung und -steuerung des Auftragsfertigers, da dieses die größte Komplexität aufweist und die Prozesse der anderen Auftragsabwicklungstypen unter diesem subsumiert werden können. Die wesentlichen Aufgaben der Eigenfertigungsplanung und -steuerung sind die Losgrößenrechnung, die Feinterminierung, die Ressourcenfeinplanung, die Reihenfolgebildung, die Verfügbarkeitsprüfung und die Auftragsfreigabe (Schuh et al. 2012a, S. 51)

Im Rahmen der Losgrößenrechnung, die gemäß der in Abschnitt 2.1 präsentierten Arbeitsdefinition zur Produktionsablaufplanung im weiteren Sinne gehört, werden die in jedem Produktionsbereich auszuführenden Operationen in Lose unterteilt. Eine effiziente Losbildung ist insbesondere dann relevant, wenn Rüstzeiten und -kosten zwischen der Durchführung unterschiedlicher Produktionsoperationen bestehen. In diesem Fall besteht die Herausforderung darin, einen Kompromiss zwischen hohen Umlaufbeständen und einer geringen Anzahl

von Rüstvorgängen bzw. zwischen niedrigen Umlaufbeständen und einer hohen Anzahl von Rüstvorgängen auszumachen (Schuh et al. 2012a, S. 53; Glaser et al. 1992; Kurbel 2003). Obgleich die Losgrößenrechnung ebenfalls als mathematisches Optimierungsproblem modelliert werden kann, wird sie in dieser Arbeit nicht gesondert betrachtet. In Kapitel 4 werden jedoch einige Arbeiten besprochen, die eine implizite Losgrößenoptimierung über das Entscheidungsproblem erlauben, ob eine Ressource auf einen neuen Produkttypen umgerüstet oder ob die aktuelle Rüstung beibehalten werden soll. Zudem ist die in Kapitel 5 vorgestellte Methode zum Einsatz von bestärkenden Lernverfahren für die Produktionsablaufplanung imstande, die Minimierung von Rüstvorgängen als Zielkriterium zu berücksichtigen.

Die daran anschließenden Aufgaben »Feinterminierung«, »Ressourcenfeinplanung« und »Reihenfolgeplanung« entsprechen der in Abschnitt 2.1 formulierten Definition der Produktionsablaufplanung im engeren Sinne. Die Feinterminierung ermittelt für alle Lose bzw. für die zu produzierenden Einzelaufträge die Start- und Endtermine, berücksichtigt hierbei jedoch noch nicht das verfügbare Kapazitätsangebot der Produktionsressourcen. Erst während der Ressourcenfeinplanung erfolgt eine Gegenüberstellung des aus der Feinterminierung resultierenden Kapazitätsbedarfs mit den durch die Produktionsressourcen bereitgestellten Kapazitäten. Im Fall einer Kapazitätsüberlastung können das Kapazitätsangebot und die Kapazitätsnachfrage bspw. durch zusätzliche Schichten oder durch Neuterminierung von Arbeitsgängen miteinander abgestimmt werden. Anstelle der aufeinanderfolgenden Feinterminierung und Ressourcenfeinplanung kann ebenfalls eine kombinierte Ressourcenbelegungsplanung durchgeführt werden. Hierbei werden für die durchzuführenden Produktionsoperationen die Start- und Endtermine sowie die Kapazitätszuordnung simultan geplant (Schuh et al. 2012a, S. 54 f.). Je nach Auftragsabwicklungstyp empfiehlt das Aachener PPS-Modell eine Feinterminierung mit anschließender Ressourcenfeinplanung oder eine kombinierte Ressourcenbelegungsplanung (siehe Abbildung 2.2). Entsprechend der in Abschnitt 2.1 formulierten Definition für die Produktionsablaufplanung bezieht sich diese Arbeit stets auf die kombinierte Ressourcenbelegungsplanung. Die in Abschnitt 2.3 vorgestellten Optimierungsmodelle für die Produktionsablaufplanung erlauben eine simultane Betrachtung von Feinterminierung und Ressourcenfeinplanung, sodass eine gesonderte Berücksichtigung hinfällig wird. Das Ergebnis der Ressourcenbelegungsplanung ist die Verteilung von an Produktionsaufträgen auszuführenden Operationen auf die verfügbaren Produktionskapazitäten. Im Rahmen der Reihenfolgeplanung wird schließlich festgelegt, in welcher Reihenfolge die einer Produktionskapazität zugeordneten Operationen abgearbeitet werden (Schuh et al. 2012a, S. 55 f.). Die Reihenfolgebildung kann hierbei



[1] nicht enthalten im Prozessmodell der Eigenfertigungsplanung und -steuerung des Auftragsfertigers
[2] nicht enthalten im Prozessmodell der Eigenfertigungsplanung und -steuerung des Rahmenauftragsfertigers
[3] nicht enthalten im Prozessmodell der Eigenfertigungsplanung und -steuerung des Variantenfertigers
[4] nicht enthalten im Prozessmodell der Eigenfertigungsplanung und -steuerung des Lagerfertigers

Abbildung 2.2 Aggregiertes Prozessmodell der Eigenfertigungsplanung und -steuerung des Aachener PPS-Modells (in Anlehnung an Schuh et al. 2012b, S. 151)

verschiedenen Zielkriterien unterliegen, z. B. der Minimierung von Zykluszeiten, Verspätungen oder Rüstvorgängen.

Der resultierende Produktionsablaufplan (im Aachener PPS-Modell und in Abbildung 2.2 als »Werkstattprogrammvorschlag« bezeichnet) wird durch Prüfung der Verfügbarkeit der notwendigen Ressourcen, Materialien und Kapazitäten plausibilisiert. Sofern der Plan nicht realisierbar ist, muss das Eigenfertigungsprogramm (Produktionsprogramm) einer kritischen Prüfung unterzogen und ggf.

die Produktionsbedarfsplanung oder der Produktionsablaufplan angepasst werden. Andernfalls werden die Aufträge gemäß der im Produktionsablaufplan festgelegten Starttermine freigegeben (Schuh et al. 2012a, S. 57).

2.3 Mathematische Optimierung der Produktionsablaufplanung

Die Produktionsablaufplanung kann zu großen Teilen über mathematische Modelle formalisiert und optimiert werden. Die in Kapitel 5 vorgestellte Methode zum Einsatz von bestärkenden Lernverfahren für die Produktionsablaufplanung erfordern ebenfalls eine modellhafte Abbildung der zu optimierenden Planungsprobleme. Vor diesem Hintergrund wird nachstehend ein Formalismus zur Beschreibung von Problemen der Produktionsablaufplanung dargelegt, welcher in den folgenden Kapiteln Anwendung findet. Im Anschluss daran werden Methoden aufgezeigt, mit welchen Probleme der Produktionsablaufplanung als mathematische Optimierungsmodelle implementiert werden können. Zu guter Letzt werden die gängigen Lösungsverfahren zur Berechnung und Optimierung von Produktionsablaufplänen sowie deren Vor- und Nachteile beschrieben.

2.3.1 Mathematische Formalisierung

Aus mathematischer Sicht liegt jedem Ablaufplanungsproblem im Allgemeinen und jedem Problem der Produktionsablaufplanung im Speziellen die Überlegung zugrunde, dass eine endliche Menge von Aufträgen J (jobs) durch eine endliche Menge von Ressourcen M (machines) bearbeitet werden muss (Brucker 2007, S. 1). Hierbei entsprechen die Kardinalitäten von J und M, also $|J|$ und $|M|$, der Anzahl einzuplanender Aufträge bzw. der Anzahl vorhandener Ressourcen. Anstelle der Kardinalitäten werden in der Scheduling-Literatur oftmals der Parameter m zur Angabe der Ressourcenanzahl und der Parameter n zur Angabe der Auftragsanzahl verwendet. Im Folgenden werden einzelne Aufträge primär über die Laufvariable j und einzelne Ressourcen primär über die Laufvariable i indiziert, sodass gilt $j = (1, \ldots, |J|)$ bzw. $i = (1, \ldots, |M|)$. Die Fertigstellung eines jeden Auftrags j erfordert die Durchführung einer bestimmten Anzahl von Operationen aus einer endlichen Menge von Operationen O_j. Die Kardinalität $\left|O_j\right|$ definiert die Anzahl der Operationen, die an einem Auftrag durchgeführt werden können. Für die Indizierung einzelner Operationen wird im Folgenden primär die Laufvariable o verwendet, sodass gilt $o_j \forall o = \left(1, \ldots, \left|O_j\right|\right)$. Jede Operation

o_j ist durch eine Bearbeitungszeit definiert. Sofern unterschiedliche Ressourcen die gleichen Operationen durchführen können, kann die Bearbeitungszeit für eine Operation auf unterschiedlichen Ressourcen variieren. In diesem Fall definiert $o_{i,j}$ die Bearbeitungszeit für Operation o des Auftrags j auf Ressource i. Des Weiteren ist für einige Probleme eine endliche Menge von Produktionsstufen (bei Fließfertigungen) bzw. Produktionsbereichen (bei Werkstatt- oder Inselfertigungen) L charakteristisch, wobei jeder Produktionsstufe bzw. jedem Produktionsbereich l eine bestimmte Menge von Ressourcen $M^{(l)} \subseteq M$ zugeordnet ist. Ressource i in Stufe bzw. Bereich l wird dann als $i^{(l)}$ deklariert. Diese einführenden Definitionen sind zunächst ausreichend, um die mathematische Formalisierung der Ablaufplanung im Folgenden weiterauszubauen. In späteren Abschnitten werden die bisherigen Definitionen je nach Bedarf und Notwendigkeit erweitert.

Gewöhnlich werden Probleme der Ablaufplanung gemäß der Notation von Graham et al. (1979) als Tripel $\alpha|\beta|\gamma$ charakterisiert (vgl. Jaehn und Pesch 2014, S. 12; Pinedo 2016, S. 14), dessen Felder im Folgenden erläutert werden.

2.3.1.1 Probleme

Der Parameter α definiert die Problemart und -eigenschaften. Im Folgenden werden die Grundprobleme der Ablaufplanung gelistet, ergänzt durch einige Modifikationen, die insbesondere in der Produktion und Logistik häufig auftreten. Die folgende α-Notation basiert im Wesentlichen auf Pinedo (2009, S. 22 ff.) sowie Jaehn und Pesch (2014, S. 12). Die α-Notation des Hybrid Flow-Shops und des Flexible Job-Shops sind eigens für diese Arbeit formuliert, da die Literatur für beide Probleme keine einheitliche Notation bietet (vgl. z. B. Pinedo 2016, S. 15; Ruiz und Vázquez-Rodríguez 2010, S. 2).

1	**Ein-Maschinen-Problem** – Es existiert nur eine Ressource ($\|M\| = m = 1$), welche alle Aufträge bearbeitet. Alle Aufträge müssen jeweils nur eine Operation durchlaufen ($\|O_j\| = 1 \forall j = (1, \ldots, \|J\|)$). Die Optimierung beschränkt sich somit auf die Lösung eines Sequenzierungsproblems (optimale Reihenfolge der auszuführenden Operationen)

$\boldsymbol{Pm}$	**Parallel-Maschinen-Problem** – Es existieren m identische Ressourcen, die parallel arbeiten. Alle Aufträge müssen jeweils nur eine Operation durchlaufen $\left(\left\|O_j\right\| = 1 \forall j = (1, \ldots, \|J\|)\right)$. Jeder Auftrag wird nur von einer Ressource bearbeitet. Die Optimierung umfasst die Lösung eines Allokationsproblems (optimale Zuweisung von Aufträge zu Ressourcen) sowie die Lösung von $\|M\|$ Sequenzierungsproblemen (optimale Reihenfolge der auszuführenden Operationen auf jeder Ressource)
$\boldsymbol{Fm}$	**Flow-Shop-Problem** – Es existieren m unterschiedliche Ressourcen. Jede Ressource ist genau einer Bearbeitungsstufe zugeordnet und umgekehrt ($m = \|L\|$). Jeder Auftrag muss von jeder Ressource bearbeitet werden. Es gilt $\left\|O_j\right\| = m \forall j = (1, \ldots, \|J\|)$, wobei jede Operation von genau einer Ressource durchgeführt wird. Die Bearbeitungsreihenfolge der Operationen ist für alle Aufträge gleich. Ein Flow-Shop ist somit die mathematische Formalisierung einer Fließfertigung mit jeweils einer Bearbeitungskapazität pro Produktionsstufe.
$\boldsymbol{HF}\left(\boldsymbol{m}^{(1)}, \ldots, \boldsymbol{m}^{(\|L\|)}\right)$	**Hybrid-Flow-Shop-Problem** – Es handelt sich um eine Verallgemeinerung des Flow-Shops mit dem Unterschied, dass jede Bearbeitungsstufe l eine spezifische Bearbeitungskapazität $m^{(l)}$ besitzt, d. h. entweder aus einer oder aus mehreren parallel arbeitenden Ressourcen besteht. Jeder Auftrag wird nur von einer Ressource pro Bearbeitungsstufe bearbeitet. In der Literatur wird dieses Problem häufig auch als Flexible-Flow-Shop bezeichnet (Gondek 2011).
$\boldsymbol{Jm}$	**Job-Shop-Problem** – Es handelt sich um eine Verallgemeinerung des Flow-Shops mit dem Unterschied, dass die Bearbeitungsreihenfolge der Operationen nicht für alle Aufträge gleich, jedoch weiterhin fest definiert ist. Hierbei muss nicht jeder Auftrag von jeder Ressource bearbeitet werden. Das Job-Shop-Problem ist somit die mathematische Formalisierung einer Werkstattfertigung mit jeweils einer Bearbeitungskapazität pro Produktionsschritt.

$FJ\left(m^{(1)}, \ldots, m^{(\lvert L \rvert)}\right)$*1 **bzw.** FJm*2	**Flexible-Job-Shop-Problem** – Es handelt sich um eine Verallgemeinerung des Job-Shops mit dem Unterschied, dass jeder Produktionsbereich l eine spezifische Bearbeitungskapazität $m^{(l)}$ besitzt, d. h. entweder aus einer oder aus mehreren parallel arbeitenden Ressourcen besteht. Jede Operation eines Auftrags wird nur von einer Ressource der entsprechenden Bearbeitungsstufe durchgeführt. Die Art und Anzahl der durchzuführenden Operationen sowie deren Bearbeitungsreihenfolge sind weiterhin auftragsspezifisch (Pinedo 2009, S. 23 f.). Gemäß dieser Erklärung*1 handelt es sich beim Flexible-Job-Shop um das Gegenstück des Hybrid-Flow-Shops. Alternativ bietet die wissenschaftliche Literatur, z. B. Xie et al. (2019), Zhang et al. (2019), eine weitere Definition*2, die gleichermaßen anerkannt ist. Nach dieser zeichnet sich der Flexible-Job-Shop dadurch aus, dass bestimmte oder alle Operationen auf allen Ressourcen, unabhängig vom Produktionsbereich, durchgeführt werden können. Je nachdem, ob alle oder nur bestimmte Operationen ressourcenflexibel bearbeitet werden können, handelt es sich um einen Total Flexible-Job-Shop oder um einen Partial Flexible-Job-Shop.
Om	**Open-Shop-Problem** – Es handelt sich um eine weitere Verallgemeinerung des Flow-Shops, mit dem Unterschied, dass keine Einschränkungen hinsichtlich der Bearbeitungsreihenfolge existieren. Jeder Auftrag muss weiterhin von jeder Ressource bearbeitet werden, kann diese aber in beliebiger Reihenfolge durchlaufen.

2.3.1.2 Nebenbedingungen

Der Parameter β definiert Nebenbedingungen, d. h. Vorgaben und Restriktionen, für die Erstellung eines zulässigen Ablaufplans. Hierbei ist β nicht auf einen Wert beschränkt, sondern kann ebenfalls eine Liste von Nebenbedingungen enthalten. Ursprünglich definierten Graham et al. (1979) sechs Arten von Nebenbedingungen $\beta_1, \ldots, \beta_6$, die bis heute um viele Weitere ergänzt wurden. Im Folgenden werden einige der wichtigsten und am weitest verbreiteten Nebenbedingungen für Ablaufplanungsprobleme in der Produktion und Logistik gelistet, die mitunter für die Anwendungsbeispiele in Kapitel 6 von Relevanz sind (Jaehn und Pesch 2014, S. 13 f.; Pinedo 2016, S. 15 ff.).

r_j	**Release dates – Früheste Freigabezeit** – Die Bearbeitung der ersten Operation eines jeden Auftrags j darf erst nach dessen jeweiligen Freigabezeit r_j starten. Ist r_j nicht als Nebenbedingung gelistet, stehen alle Aufträge zum Zeitpunkt $t = 0$ zur Verfügung.
$pmtn$	**Preemptions – Unterbrechungen** – Operationen von Aufträgen können beliebig unterbrochen und fortgeführt werden. Unterbrechungen haben keinen Einfluss auf die für eine Operation veranschlagte Bearbeitungszeit. Ist $pmtn$ nicht als Nebenbedingung gelistet, müssen Operationen von Aufträgen stets ohne Unterbrechung durchgeführt werden.
$prec$	**Precedence – Vorrangbeziehungen** – Es existieren eine oder mehrere Vorrangbeziehungen $j \rightarrow k$, d. h. Die Bearbeitung eines bestimmten Auftrags k darf erst dann starten, wenn die Bearbeitung eines bestimmten Auftrags j beendet ist. Ist $prec$ nicht als Nebenbedingung gelistet, können Aufträge in beliebiger Reihenfolge bearbeitet werden.
s_{jk}[*1] $fmls$[*2]	**Setup times – (Familien-)Reihenfolgeabhängige Rüstzeiten** – Nach der Bearbeitung eines bestimmten Auftrags j und vor der Bearbeitung des nachfolgenden Auftrags k muss eine zusätzliche Rüstzeit s_{jk} berücksichtigt werden, während welcher die Ressource keine Aufträge bearbeitet. Der Parameter s_{0k} repräsentiert die initiale Rüstzeit, sofern Auftrag k als erstes auf einer Ressource bearbeitet wird. Rüstzeiten können entweder auf Auftrags-[*1] oder Familienebene[*2] definiert sein. Im ersten Fall bestehen Rüstzeiten zwischen allen Aufträgen. Im zweiten Fall werden Aufträge sogenannten Familien zugeordnet. Es bestehen dann lediglich Rüstzeiten zwischen Aufträgen von unterschiedlichen Familien. Sofern reihenfolgeunabhängige Rüstzeiten vorliegen, müssen diese nicht explizit modelliert, können stattdessen in den Bearbeitungszeiten als Zuschlag berücksichtigt werden.
$brkdwn$	**Breakdowns – Ressourcenausfälle** – Ressourcen sind nicht zu jedem Zeitpunkt verfügbar. Der zeitliche Abstand zwischen zwei Ausfällen sowie die Ausfalldauer können entweder durch eine konstante Zeit (i. d. R. für geplante Ausfälle, z. B. Schichtwechsel oder Wartung) oder durch eine Zufallsverteilung (i. d. R. für ungeplante Ausfälle, z. B. Störungen) definiert sein. Ist $brkdwn$ nicht als Nebenbedingung gelistet, sind alle Ressourcen zu jedem Zeitpunkt verfügbar
$prmu$	**Permutation** – Eine Nebenbedingung, die ausschließlich in Flow-Shop-Problemen auftreten kann. Die Bearbeitungsreihenfolge von Aufträgen muss auf allen Ressourcen gleich sein. Ist $prmu$ nicht als Nebenbedingung gelistet, können Ressourcen wartende Aufträge in beliebiger Reihenfolge abarbeiten.

rcrc **Recirculation – Rezirkulation** – Eine Nebenbedingung, die ausschließlich in Job-Shop-Problemen auftreten kann. Aufträge besuchen eventuell eine Ressource bzw. einen Bearbeitungsbereich mehrere Male zur Durchführung unterschiedlicher Operationen. Ist $rcrc$ nicht als Nebenbedingung gelistet, wird jeder Auftrag von jeder Ressource höchstens einmal bearbeitet.

2.3.1.3 Zielfunktionen

Der Parameter γ definiert Zielfunktionen, nach welchen die Erstellung des Ablaufplans optimiert werden soll. Analog zu β ist γ ebenfalls nicht auf einen Wert beschränkt, sondern kann eine beliebige Anzahl von Zielfunktionen enthalten. Im Folgenden werden die geläufigsten Zielfunktionen zur Optimierung von Ablaufplänen gelistet (Baker und Trietsch 2009, S. 12 f.; Jaehn und Pesch 2014, S. 14 f.; Pinedo 2016, S. 18 f.). Hierbei können diejenigen Zielfunktionen, welche eine Summation über alle Aufträge durchführen, auf solche Weise verallgemeinert werden, dass einzelne Aufträge individuell gewichtet ($\omega_j \forall j \in \{1, \ldots, |J|\}$) werden können.

F **Total Flowtime – Kumulierte Durchlaufzeit** – Die Durchlaufzeit eines einzelnen Auftrags F_j berechnet sich aus der Differenz seiner Fertigstellungszeit C_j und dessen Freigabezeit r_j. Die kumulierte Durchlaufzeit F bildet sich über die Summe aller Auftragsdurchlaufzeiten:

$$F = \sum_{j}^{|J|} F_j = \sum_{j}^{|J|} \left(C_j - r_j\right) \tag{2.1}$$

C_{max} **Makespan – Gesamtdauer des Ablaufplans** – Die Gesamtdauer eines Ablaufplans ermittelt sich über den Fertigstellungszeitpunkt des letzten fertiggestellten Auftrags:

$$C_{max} = \max\left(C_j \forall j = (1, \ldots, |J|)\right) \tag{2.2}$$

T **Total Tardiness – Gesamtverspätung über alle Aufträge** – Die Terminüberschreitung eines einzelnen Auftrags T_j ist entweder gleich null, sofern der Auftrag vor seiner Fertigstellungsfrist d_j fertiggestellt wurde, oder gleich der Differenz aus Fertigstellungszeit C_j und Fertigstellungsfrist d_j. Die Gesamtverspätung über alle Aufträge bildet sich über die Summe aller Terminüberschreitungen:

$$T = \sum_{j}^{|J|} T_j = \sum_{j}^{|J|} \max\left(0, C_j - d_j\right) \tag{2.3}$$

U **Total Unit Penalty – Anzahl verspäteter Aufträge** – Ein Auftrag gilt als verspätet, falls seine Fertigstellungszeit größer als die Fertigstellungsfrist ist. In diesem Fall wird der Auftrag bestraft ($U_j = 1$). Andernfalls wird der Auftrag nicht bestraft ($U_j = 1$). Die Anzahl verspäteter Aufträge bildet sich über die Summe vergebener und nicht vergebener Strafen:

$$U = \sum_{j}^{|J|} \left(U_j = \begin{cases} 1, & wenn\ C_j > d_j \\ 0, & sonst \end{cases} \right) \tag{2.4}$$

L_{max} **Maximum Lateness – Maximale Verspätung** – Die Unpünktlichkeit eines einzelnen Auftrags L_j bildet sich aus der Differenz zwischen dessen Fertigstellungszeit C_j und dessen Fertigstellungsfrist d_j. Im Gegensatz zur Terminüberschreitung T_j kann die Unpünktlichkeit eines Auftrags sowohl positiv als auch negativ sein. Für die maximale Verspätung gilt dann:

$$L_{max} = \max\left(L_j \forall j = (1, \ldots, |J|)\right) = \max\left(C_j - d_j \forall j = (1, \ldots, |J|)\right) \tag{2.5}$$

2.3.2 Modellbildung

Für die mathematische Optimierung von Produktionsablaufplänen muss das zugrundeliegende Produktionssystem als experimentierbares Modell implementiert werden. Ein Modell ist eine *„vereinfachte Nachbildung eines […] Systems mit seinen Prozessen in einem anderen begrifflichen oder gegenständlichen System. […] Es unterscheidet sich hinsichtlich der untersuchungsrelevanten Eigenschaften nur innerhalb eines vom Untersuchungsziel abhängigen Toleranzrahmens vom Vorbild“* (VDI 3633 Blatt 1, S. 3). Es existiert eine Vielzahl von Modellierungsmethoden und Modellen. Abbildung 2.3 zeigt ein Klassifikationsschema, welches Modelle hinsichtlich ihrer Art und Weise der Nachbildung des Untersuchungsgegenstands unterteilt.

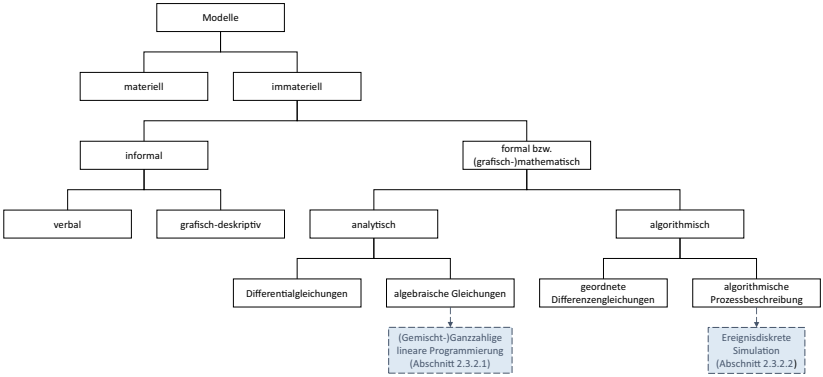


Abbildung 2.3 Klassifikation von Modellen und Modellierungsmethoden. (In Anlehnung an Page 1991, S. 5; Frank 1999, S. 51)

Die mathematische Optimierung von Produktionsablaufplänen erfordert mathematische Modelle der zugrundeliegenden Produktionssysteme. Diese können entweder analytischer oder algorithmische Natur sein. Differentialgleichungen eignen sich nur bedingt für die Modellbildung – viele Produktionsprozesse haben einen diskreten Charakter, weil die zu produzierenden Aufträge oftmals klar voneinander separierte Stückgüter sind. Hieraus resultieren für die Produktionsablaufplanung diskrete, kombinatorische Optimierungsprobleme, welche nicht

vollständig differenzierbar sind. Nichtsdestotrotz kann gewöhnlich jedes Produktionssystem durch algebraische Gleichungen modelliert werden. In der Literatur werden Probleme der Ablaufplanung häufig als (gemischt-)ganzzahlige Optimierungsprobleme modelliert. Abschnitt 2.3.2.1 widmet sich der ganzzahligen und gemischt-ganzzahligen linearen Programmierung als Modellierungsmethode für Probleme der Produktionsablaufplanung. Für die algorithmische Modellbildung von Produktionssystemen eignen sich algorithmische Prozessbeschreibungen. Aufgrund des bereits erwähnten diskreten Charakters vieler Produktionsprozesse und -systeme eignet sich insbesondere die ereignisdiskrete Simulation für die Modellbildung. Aus diesem Grund wird in Abschnitt 2.3.2.2 die ereignisdiskrete Simulation als Modellbildungsmethode für Probleme der Produktionsablaufplanung diskutiert.

2.3.2.1 Ganzzahlige und gemischt-ganzzahlige lineare Programmierung

Ein lineares Programm (LP) im Allgemeinen modelliert ein mathematisches Optimierungsproblem über eine lineare Zielfunktion, die entweder minimiert oder maximiert werden soll, sowie das zugrundeliegende System bzw. den zugrundeliegenden Prozess über eine Menge von linearen (Un-) Gleichungen. Diese werden als Nebenbedingungen bezeichnet und schränken den zulässigen Lösungsraum ein. Jede (Un-)Gleichung wird in Abhängigkeit von ganzzahligen und / oder kontinuierlichen Entscheidungsvariablen formuliert, welche gewöhnlich nichtnegative Werte annehmen müssen. Bei einem ganzzahligen linearen Programm (ILP – vom englischen Begriff »Integer Linear Program«) sind die (Un-)Gleichungen ausschließlich von ganzzahligen Entscheidungsvariablen abhängig. In einem gemischt-ganzzahligen linearen Programm (MILP – vom englischen Begriff »Mixed Integer Linear Program«) existiert mindestens eine (Un-)Gleichung, die abhängig von einer oder mehreren ganzzahligen Variablen ist, und mindestens eine (Un-)Gleichung, die abhängig von einer oder mehreren kontinuierlichen Entscheidungsvariablen ist (Zimmermann 2008, S. 111; Kallrath 2013, S. 14; Domschke et al. 2015, S. 17 f.). Bei Optimierungsproblemen der Ablaufplanung ist die Zielfunktion gewöhnlich eine zu minimierende Funktion der Form:

$$Min\ z = \sum_{g=1}^{|\gamma|} \omega_g \gamma_g \tag{2.6}$$

Formel (2.6) gilt sowohl für einfache Zielfunktionen mit lediglich einer zu optimierenden Zielstellung ($\omega_g = 1 \Leftrightarrow g = 1 \wedge |\gamma| = 1$) als auch für multikriterielle Zielfunktionen mit mehreren zu optimierenden Zielstellungen ($\omega_g \in \mathbb{R}^+ \forall g = (1, \ldots, |\gamma|) \Leftrightarrow |\gamma| > 1$) Der Parameter ω_g repräsentiert die Gewichtung der entsprechenden Zielfunktion γ_g Die Formulierung von Nebenbedingungen hängt maßgeblich von der Kodierung der Entscheidungsvariablen ab. Diesbezüglich sind für die Ablaufplanung insbesondere drei Ansätze weit verbreitet:

- Die rangbasierte Kodierung von Wagner (1959)

$$x_{ijk} = \begin{cases} 1, & wenn\ Auftrag\ i\ an\ Position\ j\ in\ der\ Auftragssequenz \\ & vor\ Ressource\ k\ platziert\ werden\ soll \\ 0, & sonst \end{cases} \tag{2.7}$$

- Die zeitbasierte Kodierung von Bowman (1959)

$$x_{ikt} = \begin{cases} 1, & wenn\ Auftrag\ i\ auf\ Ressource\ k\ in\ Periode\ t \\ & produziert\ werden\ soll \\ 0, & sonst \end{cases} \tag{2.8}$$

- Die disjunktiv-vorrangbasierte Kodierung von Manne (1960)

$$x_{ijk} = \begin{cases} 1, & wenn\ Auftrag\ i\ vor\ Auftrag\ j\ auf\ Ressource\ k \\ & produziert\ werden\ soll \\ 0, & sonst \end{cases} \tag{2.9}$$

Laut verschiedenen Studien (u. a. Pan 1997; Ku und Beck 2016) wird die disjunktiv-vorrangbasierte Kodierung als am effizientesten eingeschätzt, da sie für viele Ablaufplanungsprobleme die geringste Anzahl an Entscheidungsvariablen benötigt. Jeder der drei Kodierungsansätze erlaubt die Formulierung eines rein ganzzahligen Optimierungsproblems. Darüber hinaus finden sich in der Literatur mathematische Modelle, welche neben binären Entscheidungsvariablen kontinuierliche Hilfsvariablen verwenden, um bspw. bei einer rangbasierten oder

disjunktiv-vorrangbasierten Kodierung zusätzlich die Wartezeiten von Aufträgen oder die Stillstandzeiten zwischen aufeinanderfolgenden Aufträgen zu berechnen (vgl. Pinedo 2016, S. 157).

Der Vorteil eines (M)ILP liegt insbesondere in dessen Berechnungseffizienz, da das vollständige Produktionssystem über einfache lineare mathematische Gleichungen abgebildet wird. Darüber hinaus fördert eine mathematische Modellierung ein tiefgreifendes Verständnis hinsichtlich der abgebildeten Prozesse und deren Zusammenhänge (Kallrath 2013, S. 36).

Die Nachteile eines (M)ILP liegen zum einen in der komplizierten Modellierung von Nebenbedingungen. Bestimmte Eigenschaften eines Produktionssystems können häufig nicht als lineare Gleichungen und ausschließlich mithilfe von Entscheidungsvariablen modelliert werden. Oftmals bedarf es mehrerer Gleichungen und zusätzlicher Schlupfvariablen, um alle relevanten Eigenschaften eines Produktionssystems abzubilden. Des Weiteren ist ein (M)ILP ungeeignet, um dynamische Ablaufplanungsprobleme abzubilden, bei welchen sich die Eingangsdaten zur Laufzeit verändern, sowie ebenfalls nicht geeignet für einige Arten von stochastischen Ablaufplanungsproblemen, bei welchen z. B. zur Laufzeit zufällige Störungen auftreten. Der Grund hierfür ist, dass die Nebenbedingungen eines (M)ILP nicht den eigentlichen Prozess in seiner Dynamik und Ablaufreihenfolge abbilden. Aus diesem Grund können dynamische und datengetriebene Steuerungsansätze, z. B. maschinelles Lernen, nur unter einem erheblichen Modellierungsaufwand mit einem (M)ILP integriert werden.

Einen alternativen Modellierungsansatz, der die Nachteile von (M)ILP adressiert, bietet die ereignisdiskrete Simulation.

2.3.2.2 Ereignisdiskrete Simulation

Die ereignisdiskrete Simulation (DES – vom englischen Begriff »Discrete Event Simulation«) ist eine Methode zur Modellierung und Simulation von Warteschlangensystemen und -prozessen. Kerncharakteristik der DES ist, dass sich die Modellzustände nicht kontinuierlich, also in konstanten infinitesimalen Zeitschritten ändern, sondern sprunghaft zu diskreten und möglicherweise zufälligen Zeitpunkten (Law 2015, S. 6; Schriber et al. 2017, S. 736). Hierbei schreitet die Simulationszeit in Abhängigkeit von den abgearbeiteten Ereignissen voran (Law 2015, S. 7). Zustandsänderungen werden mit im Voraus geplanten Ereignissen verbunden, die in einer (Banks et al. 2010, S. 110) oder mehreren Listen (Schriber et al. 2017, S. 738) organisiert sind und von einem Ereignisverwalter ausgeführt werden. Das Auftreten von Ereignissen unterliegt dem durch Anwender*innen modellierten Prozess. DES wird häufig in Produktion und Logistik eingesetzt, da

die meisten Produktions- und Logistikprozesse einen diskreten Charakter haben (Scholz-Reiter et al. 2007, S. 126).

Ein DES-Modell besteht aus einer Menge sogenannter Entitäten. Entitäten sind Objekte, welche eine Menge von Attributen besitzen und ihren Zustand (Werte von Attributen) in Abhängigkeit von Ereignissen verändern. Zustandsänderungen von Entitäten können wiederum neue Ereignisse hervorrufen. Entitäten können bspw. Menschen, Produkte, Maschinen u. v. m. repräsentieren (Banks et al. 2010, S. 30; Law 2015, S. 3). Ferner können Entitäten in Flussobjekte und Ressourcen unterteilt werden (vgl. Schriber et al. 2017, S. 736). Flussobjekte werden während der Simulation erzeugt, durchlaufen das System und werden am Systemausgang vernichtet. Im Beispiel der Produktionsablaufplanung würden die zu allozierenden und zu sequenzierenden Aufträge durch Flussobjekte repräsentiert werden. Ressourcen sind i. d. R. kapazitätsbeschränkte Bestandteile des Systems, die entweder ortsgebunden oder ortsveränderlich sind. Ressourcen werden von Flussobjekten zur Ausführung von Operationen beansprucht. Eine Operation simuliert einen Zeitverbrauch, der durch einen Start- und Endzeitpunkt definiert ist. Der Endzeitpunkt kann hierbei fix vorbestimmt oder unbestimmt sein und stattdessen vom Eintritt einer Bedingung abhängen. Das Flussobjekt blockiert eine Kapazitätseinheit der Ressource über die gesamte Operationszeit. In der Produktionsablaufplanung werden z. B. Maschinen und Montagestationen durch ortsgebundene Ressourcen abgebildet, während bspw. Maschinenbediener*innen und Montagearbeiter*innen durch ortsveränderliche Ressource modelliert werden. Vor einer Ressource bilden sich Warteschlangen von Flussobjekten, sofern die Kapazitätsnachfrage von Flussobjekten das Kapazitätsangebot einer Ressource zu einem bestimmten Zeitpunkt übersteigt.

Die Art und Weise, wie DES-Modelle implementiert werden, hängt stark von der Entwicklungsumgebung ab. Vornehmlich werden grafische oder sprachliche Beschreibungsformen oder eine Kombination aus beiden verwendet (VDI 3633 Blatt 1). In grafischen Beschreibungsformen werden DES-Modelle vornehmlich als Flussdiagramme modelliert, welche den simulierten Prozess abbilden. Sprachliche Beschreibungsformen verwenden rudimentäre Programmier- oder spezifische Simulationssprachen, um Systeme und Prozesse zu modellieren.

Der Hauptvorteil der DES ist, dass Systeme und Prozesse in beliebiger Genauigkeit abgebildet werden können, da jeder Prozess durch beliebig viele Ereignisse modelliert werden kann (Kuhn und Rabe 1998, S. 4). Des Weiteren erlaubt die prozessgetreue Abbildung und Ausführung von DES-Modellen die internen Zusammenhänge und Abhängigkeiten eines komplexen Systems unter Berücksichtigung unterschiedlicher Eingangsdaten und Parameter zu untersuchen (Banks et al. 2010, S. 22).

Der Hauptnachteil von DES-Modellen ist die mitunter vergleichsweise hohe Rechenzeit, insbesondere bei komplexen Modellen, da alle relevanten Systemkomponenten als einzelne Objekte abgebildet werden (Law 2015, S. 2). Aus diesem Grund sollten DES-Modelle das zugrundeliegende System so einfach wie möglich und nur so komplex wie nötig abbilden. Eine effiziente und den Anforderungen entsprechende Modellbildung erfordert i. d. R. viel Erfahrung und Einarbeitungszeit (Banks et al. 2010, S. 24).

2.3.3 Konventionelle Lösungsverfahren

Im Folgenden werden die in der wissenschaftlichen Literatur am häufigsten beschriebenen und in der Industrie gebräuchlichsten Lösungsverfahren für die Produktionsablaufplanung skizziert. Diese lassen sich in ganzzahlige lineare Optimierungsverfahren, Heuristiken und Metaheuristiken unterteilen. Bezugnehmend auf Abbildung 1.2 in Abschnitt 1.1 sei darauf hingewiesen, dass Prioritätsregeln und spezifische Heuristiken in Abschnitt 2.3.3.2 zusammengefasst behandelt werden.

2.3.3.1 Ganzzahlige lineare Optimierung

Verfahren der ganzzahligen linearen Optimierung werden auch exakte Verfahren genannt, da sie in endlich vielen Schritten eine optimale Lösung garantieren (Domschke et al. 2015, S. 135). Grundsätzlich können diese Verfahren wie folgt klassifiziert werden (Dempe und Schreier 2006; Zimmermann 1988; Domschke et al. 2015, S. 134):

- Dekompositionsverfahren, welche sich wiederum aufteilen in:
  - Entscheidungsbaumverfahren
  - Dynamische Programmierung
- Schnittebenenverfahren

Die Grundidee aller Dekompositionsverfahren ist ein Problem in viele kleine Teilprobleme zu zerlegen. Aus den Optima aller Teilprobleme resultiert die optimale Lösung des Gesamtproblems.

Bei Entscheidungsbaumverfahren wird das Gesamtproblem in Form eines Entscheidungsbaums in Teilprobleme aufgeteilt. Die vollständige Enumeration des Lösungsraums lässt sich als Entscheidungsbaum formulieren. Ausgehend von

der ersten Entscheidungsvariable werden Zweige für alle möglichen Werte gebildet. Für jeden Wert der ersten Entscheidungsvariable werden Verzweigungen für alle möglichen Werte der zweiten Entscheidungsvariable gebildet. Das Prinzip setzt sich für alle weiteren Entscheidungsvariablen fort, sodass der vollständige Lösungsraum modelliert wird. Abbildung 2.4 illustriert den Entscheidungsbaum für drei binäre Variablen x_1, x_2 und x_3.

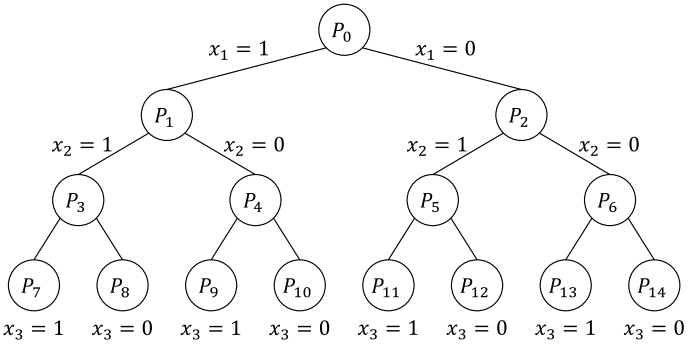


Abbildung 2.4 Beispielhafter Entscheidungsbaum für drei binäre Variablen

Bereits bei dieser geringen Anzahl von Entscheidungsvariablen resultieren 15 auszuwertende Teilprobleme. In diesem Fall steigt mit zunehmender Variablenanzahl die Menge an Teilprobleme überexponentiell, und zwar um $2 * (2^n) - 1$, wobei n die Anzahl der binären Entscheidungsvariablen repräsentiert. Aus diesem Grund ist für größere Probleme eine vollständige Enumeration i. d. R. nicht durchführbar. Stattdessen kann auf Verfahren der unvollständigen Enumeration zurückgegriffen werden. Das bekannteste Verfahren dieser Gruppe ist der Branch-And-Bound (B&B) -Algorithmus, welcher ursprünglich von Land und Doig (1960) entwickelt wurde. Die Idee des B&B-Verfahrens ist zunächst das Gesamtproblem durch geeignete Separationstechniken in Teilprobleme zu verzweigen (Branching). Des Weiteren verwendet das Verfahren obere und untere Schranken (Bounding), um frühzeitig unbrauchbare Lösungen zu identifizieren bzw. die relative Güte einer Lösung zu bewerten. Zielstellung ist es, bestimmte Verzweigungen im Entscheidungsbaum nicht mehr ausloten zu müssen und somit Rechenzeit einzusparen. Bei Minimierungsproblemen gilt die obere Schranke als Richtwert, ob eine Lösung weiterverfolgt werden soll (Lösung < obere Schranke) oder nicht. Die obere Schranke wird zu Beginn entweder mit $+\infty$ initialisiert oder mithilfe einer Heuristik berechnet. Immer dann, wenn B&B eine bessere

Lösung als die obere Schranke findet, wird diese durch die bisher beste gefundene Lösung überschrieben. Die untere Schranke hingegen ist ein Indikator für die beste zu erreichende Lösung und kann bspw. verwendet werden, um ein vorzeitiges Abbruchkriterium zu definieren. Die untere Schranke wird i. d. R. durch eine Relaxation des eigentlichen Problems ermittelt, das in Polynomialzeit optimiert werden kann. In einigen Fällen kann bspw. durch Weglassen der Ganzzahligkeitsbedingung eine optimale Lösung mithilfe des Simplex-Algorithmus berechnet werden (vgl. Dempe und Schreier 2006, S. 185 f.; Zimmermann 2008, S. 252 f.; Domschke et al. 2015, S. 140 f.). B&B ist kein generisches Verfahren, das ohne Anpassung auf jedes kombinatorische Optimierungsproblem angewandt werden kann. Vielmehr handelt es sich um ein allgemeines Handlungsprinzip, welches problemspezifisch zu implementieren ist. Konkret betrifft dies die Separationstechniken zum Aufbau des Entscheidungsbaums sowie die Methoden zum Ausloten von Verzweigungen sowie zur Berechnung von oberen und unteren Schranken. Aufgrund der hohen Rechenzeit beschränkt sich in der Ablaufplanung der Einsatz von B&B-Techniken i. d. R. auf vergleichsweise kleine Probleminstanzen mit wenigen Aufträgen und Ressourcen (Ruiz und Vázquez-Rodríguez 2010; Zhang et al. 2019).

Bei der dynamischen Programmierung (DP), deren Ursprünge auf Bellman (1951) zurückgehen, wird ein Gesamtproblem in eine Sequenz aufeinanderfolgender Teilprobleme unterteilt. Das Gesamtproblem wird durch stufenweise Berechnung der Teilprobleme optimiert. Jedes Teilproblem bzw. jede Optimierungsstufe wird durch eine Menge von Zuständen (Eingang) und Entscheidungen (Ausgang) beschrieben. Die Entscheidung einer Optimierungsstufe geht als Zustandsgröße in die darauffolgende Stufe ein. Analog zu B&B-Verfahren repräsentiert DP kein generisches Lösungsverfahren, sondern ein Handlungsprinzip, welches stets problemspezifisch konzeptioniert und implementiert werden muss (Dempe und Schreier 2006, S. 208; Zimmermann 2008, S. 233 f.; Domschke et al. 2015, S. 165). Laut Domschke et al. sind DP-Methoden weitaus schwerer zugänglich als bspw. lineare Optimierungsansätze. Womöglich finden sich aus diesem Grund nur wenige Anwendungsbeispiele für PPS-Probleme. Ruiz und Vázquez-Rodríguez (2010) berichten in ihrer umfangreichen Literaturrecherche zu hybriden Flow-Shop-Problemen lediglich von zwei Publikationen, die DP zur Lösung von kleinen Probleminstanzen einsetzen. In den bekannten Literaturstudien zu Job-Shop-Problemen (Chaudhry und Khan 2016; Xie et al. 2019; Zhang et al. 2019) wird von keinen Arbeiten berichtet, die DP als Lösungsverfahren einsetzen. Ungeachtet dessen existieren Arbeiten zur Anwendung von DP für Job-Shop-Probleme. Besondere Erwähnung soll die Arbeit von Gromicho et al. (2012) finden, die ein DP-Verfahren beschreibt, dass Job-Shop-Probleme optimal

löst und eine mindestens exponentiell geringere Laufzeitkomplexität als die vollständige Enumeration des Lösungsraums besitzt. Konkret löst das DP-Verfahren der Autoren Probleminstanzen mit fünf Ressourcen und zehn Aufträgen in ca. 16 bis 33 Minuten.

Die Grundidee des Schnittebenenverfahrens geht auf Gomory (1958) zurück. Zunächst wird ein gegebenes ganzzahliges Optimierungsproblem um dessen Ganzzahligkeitsbedingung vereinfacht, sodass die resultierende LP-Relaxation grafisch oder mithilfe des Simplex-Algorithmus gelöst werden kann. Das Optimum der LP-Relaxation gilt fortan als obere Schranke für die Ermittlung der ganzzahligen Lösung. Nun wird durch schrittweises Hinzufügen zusätzlicher Nebenbedingungen – sogenannter Schnittebenen – versucht, für jede Entscheidungsvariable ein ganzzahliges Optimum zu finden. Jede Schnittebene wird so formuliert, dass die aktuelle Lösung der LP-Relaxation außerhalb, jedoch alle zulässigen Punkte des ganzzahligen Optimierungsproblems innerhalb von ihr liegen. Jedes Mal, wenn eine neue Schnittebene eingeführt wird, erfolgt eine weitere Iteration mit dem Simplex-Algorithmus, sodass sich eine neue Lösung ergibt. Sofern diese ganzzahlig ist, handelt es sich um die optimale Lösung. Andernfalls wird die Lösung zur Konkretisierung der oberen Schranke verwendet. Die Prozedur wiederholt sich für jede nichtganzzahlige Entscheidungsvariable, bis eine ganzzahlige Lösung gefunden wird oder bis nachgewiesen werden kann, dass für das vorliegende Problem keine zulässige ganzzahlige Lösung existiert (vgl. Dempe und Schreier 2006, S. 197 ff.; Zimmermann 2008, S. 115 ff.). Schnittebenenverfahren werden heutzutage kaum noch als alleinige Methode zur Berechnung ganzzahliger Optimierungsprobleme verwendet. Bereits in den 1960er und 1970er Jahren stellte sich heraus, dass das schrittweise Hinzufügen von Nebenbedingungen und Lösen mit dem Simplex-Algorithmus kein effizientes Vorgehen darstellt (Zimmermann 2008, S. 310). Allerdings kann das Schnittebenenprinzip in Kombination mit B&B implementiert werden, um präzisere Schranken als Entscheidungsmaß für das Ausloten von Teilproblemen zu berechnen. Die Kombination beider Methoden ist als Branch-And-Cut (B&C) -Verfahren bekannt (Zimmermann 2008, S. 313). In den gängigen Literaturstudien zu Lösungsmethoden für Ablaufplanungsprobleme werden weder Arbeiten zu Schnittebenen- noch zu B&C-Verfahren gelistet (Ruiz und Vázquez-Rodríguez 2010; Chaudhry und Khan 2016; Xie et al. 2019; Zhang et al. 2019). Gleichwohl existieren einige Arbeiten, welche B&C-Verfahren für einfache Ablaufplanungsprobleme untersuchen. So präsentieren Stecco et al. (2008) ein B&C-Verfahren für ein Ein-Maschinen-Problem mit reihenfolge- und zeitabhängigen Rüstzeiten, während Kis und Kovács (2012) ein B&C-Verfahren für

ein Parallel-Maschinen-Problem untersuchen. Für komplexere Ablaufplanungsprobleme, wie bspw. Hybrid-Flow-Shops oder Job-Shops, finden sich in der wissenschaftlichen Literatur keine Anwendungsbeispiele.

2.3.3.2 Heuristiken

Eine Heuristik ist ein Algorithmus, der für ein gegebenes Optimierungsproblem eine zulässige Lösung berechnet. Wenn die resultierende Lösung für jede Eingabe optimal ist, handelt es sich gleichzeitig um ein exaktes Lösungsverfahren (Korte und Vygen 2018, S. 436), andernfalls um ein Näherungsverfahren (vgl. Dempe und Schreier 2006, S. 215). Das Hauptmerkmal jeder Heuristik ist, dass sie nach definierten Regeln und Schrittfolgen lediglich einen Teil des gesamten Lösungsraums analysiert. Bei Näherungsverfahren kann nicht nachgewiesen werden, dass sich das globale Optimum nicht im ausgeschlossenen Lösungsraum befindet, weswegen für die ermittelte Lösung ebenfalls kein Optimalitätsnachweis möglich ist (vgl. Zimmermann 2008, S. 273 f.).

Es existieren vergleichsweise wenige Heuristiken, die ein Problem optimal lösen können. Ihr Einsatz ist auf sehr spezifische und einfache Probleme limitiert. Für das Ein-Maschinen-Problem wurden problemspezifische Heuristiken von Moore (1968) und Lawler (1973) entwickelt. Moore's Algorithmus minimiert die Anzahl verspäteter Aufträge, während Lawler's Algorithmus die zeitabhängigen Produktionskosten minimiert, wobei zusätzlich Vorrangbedingungen zwischen den Aufträgen bestehen. Darüber hinaus existieren für bestimmte Flow-Shop-Konfigurationen exakte Heuristiken. Eine der ältesten und populärsten Heuristiken dieser Art stammt von Johnson (1954), welche die Gesamtdauer eines Ablaufplans für einen zweistufigen Flow-Shop mit genau einer Ressource pro Stufe minimiert. Aufbauend auf Johnson's Algorithmus präsentieren Campbell et al. (1970) eine Heuristik, welche für Flow-Shops mit beliebiger Anzahl von Stufen und jeweils einer Ressource pro Stufe einen Produktionsplan mit minimaler Gesamtdauer berechnet. Die Idee des Ansatzes ist ein n-stufiges Flow-Shop-Problem in n-1 zweistufige Flow-Shop-Probleme aufzuteilen und mit Johnson's Algorithmus zu lösen. Von den resultierenden Lösungen wird nun die beste Lösung für das Gesamtproblem gewählt. Für Parallel-Maschinen-Probleme ist die Heuristik von Sahni (1979) von größerer Bekanntheit. Diese findet einen Plan, der keine Fertigstellungsfristen von Aufträgen verletzt, unter der Annahme, dass die Bearbeitung von Aufträgen unterbrochen werden kann. Alle beschriebenen Heuristiken lösen das jeweilige Problem für eine beliebige Anzahl von Aufträgen in Polynomialzeit. Gleichbedeutend sind diese Probleme nicht der Komplexitätsklasse »NP« zuzuordnen, zu welcher bspw. viele Hybrid-Flow-Shop- und Job-Shop-Probleme gehören und für welche nach aktuellem Stand

der Wissenschaft und Technik keine exakten heuristischen Verfahren existieren. T'kindt und Billaut (2006) beschreiben eine Vielzahl weiterer Heuristiken, die spezifische Ablaufplanungsprobleme größtenteils optimal, teilweise non-optimal in Polynomialzeit lösen.

Ebenfalls zur Klasse der Heuristiken gehören sogenannte Prioritätsregeln, welche u. a. auch in etablierten PPS-Modellen zur Auftragssequenzierung vor Maschinen und Arbeitsstationen empfohlen werden (vgl. Schuh et al. 2012a, S. 53; Wiendahl und Wiendahl 2019, S. 334 f.). Obgleich Prioritätsregeln durch ihre einfache Anwendbarkeit und Echtzeitfähigkeit überzeugen, sind insbesondere bei hochdimensionalen und stark beschränkten Ablaufplanungsproblemen die Pläne weit entfernt von der optimalen Lösung (Chandra und Talavage 1991). Sofern Prioritätsregeln nicht statisch, sondern dynamisch-situativ angewandt werden, können sie ebenfalls für komplexere Probleme hochqualitative Lösungen erzielen. Dies wurde u. a. in zwei Vorarbeiten nachgewiesen, in denen mithilfe von genetischen Algorithmen die besten Prioritätsregeln für fixe Zeitintervalle gesucht wurden (Rolf et al. 2020a; Rolf et al. 2020b). Die in Kapitel 5 vorgestellte Methode zum Einsatz von bestärkenden Lernverfahren für die Produktionsablaufplanung kann u. a. für das Training eines intelligenten Agenten eingesetzt werden, der dynamisch die Prioritätsregel für eine Ressource anpasst, unter Einbeziehung aktueller Umgebungs- und Zustandsdaten. Vor diesem Hintergrund sollen im Folgenden einige der in der Literatur am häufigsten gelisteten Prioritätsregeln kurz vorgestellt werden (T'kindt und Billaut 2006, S. 21 f.; Pinedo 2016, S. 376 f.; Wiendahl und Wiendahl 2019, S. 334 ff.).

FIFO **First In – First Out** – Alle Aufträge werden in aufsteigender Reihenfolge gemäß ihres Eintrittszeitpunkts in das System bzw. – sofern alle Aufträge zum Zeitpunkt $t = 0$ verfügbar sind – in aufsteigender Reihenfolge gemäß ihrer Position in der Auftragsliste abgearbeitet.

SPT **Shortest Processing Time (Kürzeste Bearbeitungszeit)** – Alle Aufträge werden in aufsteigender Reihenfolge gemäß ihrer benötigten Prozesszeit bearbeitet. Diese Regel berechnet für das Ein-Maschinen-Problem, ohne weitere Restriktionen, den Ablaufplan mit der geringsten Gesamtdauer C_{max}.

LPT **Longest Processing Time (Längste Bearbeitungszeit)** – Alle Aufträge werden in aufsteigender Reihenfolge gemäß ihrer benötigten Prozesszeit bearbeitet.

EDD **Earliest Due Date (Früheste Fertigstellungsfrist)** – Alle Aufträge werden in aufsteigender Reihenfolge gemäß ihrer Fertigstellungsfrist bearbeitet. Diese Regel berechnet für das Ein-Maschinen-Problem, ohne weitere Restriktionen, den Ablaufplan mit der geringsten Gesamtverspätung T sowie mit der geringsten maximalen Verspätung L_{max} über alle Aufträge.

LST **Least Slack Time (Geringste Schlupfzeit)** – Die Schlupfzeit eines Auftrags ist die verbleibende Zeitspanne bis zu dessen Fertigstellungsfrist. Alle Aufträge werden in aufsteigender Reihenfolge gemäß ihrer Schlupfzeit sortiert.

2.3.3.3 Metaheuristiken

Metaheuristiken sind generische, iterative sowie gewöhnlich nichtdeterministische Algorithmen, welche unter Einsatz von einer oder mehreren untergeordneten Heuristiken, z. B. Nachbarschaftssuche, paarweiser Austausch, Rekombination von Lösungen u. v. m., den Lösungsraum eines Optimierungsproblems durchsuchen (vgl. Osman und Laporte 1996; Blum und Roli 2003; Hussain et al. 2019).

Oftmals sind die Suchstrategien von Metaheuristiken an Wirkprinzipien der Natur und Physik angelehnt. So imitieren bspw. evolutionäre (Rechenberg und Eigen 1973) und genetische Algorithmen (Holland 1975) das Prinzip von Darwins Evolutionstheorie um neue Lösungskandidaten zu erschließen und bestehende unattraktive Lösungen zu verwerfen. Währenddessen ist die Metaheuristik »Simulated Annealing« (Kirkpatrick et al. 1983) vom Prozess abkühlenden Metalls inspiriert, um mit fortschreitender Zeit die zufällige Exploration des Lösungsraums zu reduzieren. Schwarmalgorithmen repräsentieren eine weitere prominente Unterkategorie von Metaheuristiken. Es handelt sich um Multiagentensysteme, die den Lösungsraum eines gegebenen Optimierungsproblems dezentral durchsuchen, wobei die Agenten sich gegenseitig in ihrem Suchverhalten beeinflussen. Zwei populäre schwarmbasierte Metaheuristiken sind die Partikelschwarmoptimierung (Eberhart und Kennedy 1995) und der Ameisenalgorithmus (Dorigo et al. 1996).

Um das Wirkprinzip von Metaheuristiken zu versinnbildlichen, soll im Folgenden beispielhaft die Funktionsweise genetischer Algorithmen skizziert werden. Die Idee ist zunächst eine Menge (Population) von zufälligen Startlösungen (Individuen) zu generieren. Ein Individuum bzw. ein Lösungskandidat ist als Vektor kodiert. So könnte bspw. bei einem PPS-Ablaufplanungsproblem, in welchem eine Menge von Aufträgen einer Menge von Ressourcen zugeordnet werden soll, ein Individuum als n-dimensionaler Vektor kodiert werden, wobei n der

Anzahl zuzuordnender Aufträge entspricht und der Wert jedes Vektorelements dem Index der jeweiligen Maschine, zu welcher der entsprechende Auftrag zugeordnet werden soll. Über mehrere Iterationen (Generationen) werden zufällige Elemente der jeweiligen Lösungsvektoren zufällig verändert (Mutation) und / oder paarweise ausgetauscht (Inversion). Darüber hinaus können zwei Individuen, durch paarweisen Austausch untereinander, zu neuen Individuen kombiniert werden (Kreuzung). Mithilfe eines mathematischen Modells oder eines Simulationsmodells des Optimierungsproblems wird für jedes initiale und modifizierte Individuum ein sogenannter Fitnesswert berechnet. Hierbei handelt es sich i. d. R. um die zu optimierende Zielfunktion. Der Fitnesswert gilt als Indikator, ob ein Individuum beibehalten, modifiziert oder von der Population ausgeschlossen werden soll. Der Algorithmus terminiert, sobald ein benutzerdefiniertes Abbruchkriterium erreicht wird, z. B. wenn die Population über eine bestimmte Anzahl von Generationen evolviert oder ein Fitnessschwellenwert über- (Maximierung) bzw. unterschritten (Minimierung) wurde (vgl. Sivanandam und Deepa 2008, S. 83 ff.; Reeves 2010).

Das Angebot an Metaheuristiken ist zahlreich, vielfältig und kontinuierlich wachsend. Eine vollumfängliche Übersicht zu verfügbaren Metaheuristiken würde über den Rahmen dieser Arbeit hinausgehen. Ausführliche Darstellungen zu Metaheuristiken und ihren Funktionsweisen finden sich u. a. in (Blum und Roli 2003; Doerner et al. 2007; Bianchi et al. 2009; Gendreau und Potvin 2010; Burke und Kendall 2014; Du und Swamy 2016; Siarry 2016; Dokeroglu et al. 2019; Hussain et al. 2019).

Der Hauptvorteil aller Metaheuristiken ist, dass sie aufgrund ihrer generischen, nicht problembezogenen Suchstrategien vergleichsweise einfach für jede Art von Optimierungsproblem implementiert werden können (vgl. Weicker 2015, S. 25). Wie bereits in Abschnitt 1.1 beschrieben birgt der iterative, gewöhnlich randomisierte Suchprozess einer Metaheuristik ebenfalls Nachteile. So ist es bspw. nicht möglich abzuschätzen, wie viel Zeit eine Metaheuristik beansprucht, um eine Lösung von ausreichender Güte für ein bestimmtes Optimierungsproblem zu identifizieren. Mehr noch können bei zwei unterschiedlichen Instanzen ein und desselben Optimierungsproblems die Rechenzeit und erzielte Lösungsgüte stark voneinander abweichen (vgl. Chopard und Tomassini 2018, S. 191 f.). Vor diesem Hintergrund eignen sich Metaheuristiken nur bedingt zur Entscheidungsunterstützung bei zeitkritischen Optimierungsproblemen, bei denen in sehr kurzer Zeit eine den Anforderungen entsprechende Lösung ermittelt werden muss. Ebenfalls eignen sie sich nur bedingt für dynamische und stochastische Optimierungsprobleme, bei denen sich die Eingangsdaten der Optimierung zur Laufzeit ändern und nur schwer vorhersagbar sind.

Grundlagen des Bestärkenden Lernens 3

Im folgenden Kapitel soll das Themengebiet des bestärkenden Lernens theoretisch fundiert werden. Das bestärkende Lernen ordnet sich dem maschinellen Lernen unter, welches wiederum der Methodenwelt der künstlichen Intelligenz angehört. In Abschnitt 3.1 wird deshalb zunächst das bestärkende Lernen in die Themengebiete der künstlichen Intelligenz und des maschinellen Lernens eingeordnet. Zur besseren Abgrenzung werden zudem kurz die angrenzenden maschinellen Lernparadigmen »überwachtes Lernen« und »unüberwachtes Lernen« erläutert. Abschnitt 3.2 skizziert das Grundprinzip des bestärkenden Lernens und führt ferner eine Taxonomie ein, gemäß welcher die Methoden des bestärkenden Lernens in dieser Arbeit klassifiziert werden. Die beiden Hauptkategorien dieser Taxonomie bilden das gradientenabhängige und das gradientenfreie bestärkende Lernen, die wiederum in den Abschnitten 3.3 und 3.4 separat erläutert werden. Die im Rahmen dieser Arbeit untersuchten Lernalgorithmen werden für das Training von sogenannten künstlichen neuronalen Netzen verwendet. Ein Exkurs hinsichtlich des Aufbaus und der Funktionsweise von künstlichen neuronalen Netzen ist als Anhang A dem elektronischen Zusatzmaterial beigefügt.

Ergänzende Information Die elektronische Version dieses Kapitels enthält Zusatzmaterial, auf das über folgenden Link zugegriffen werden kann https://doi.org/10.1007/978-3-658-41751-2_3.

S. Lang, *Methoden des bestärkenden Lernens für die Produktionsablaufplanung*,
https://doi.org/10.1007/978-3-658-41751-2_3

3.1 Einordnung in die künstliche Intelligenz und in das maschinelle Lernen

Das bestärkende Lernen ist ein Teilgebiet des maschinellen Lernens, welches sich wiederum als Teilgebiet der künstlichen Intelligenz versteht (Nilsson 2010). Der Begriff »Künstliche Intelligenz« (KI) ist nicht eindeutig definiert. In der wissenschaftlichen Literatur finden sich eine Vielzahl von Erklärungsversuchen. Ertel (2016, S. 1 ff.) diskutiert einige Definitionen von KI, die im Laufe der Zeit entstanden sind. Gemäß dem Autor spiegelt die folgende Definition von Rich (1983, S. 2) am ehesten die moderne Vorstellung von KI wider: *„Artificial Intelligence is the study of how to make computers do things at which, at the moment, people are better"*. Sinngemäß übersetzt: Künstliche Intelligenz beschäftigt sich mit der Frage, wie Computer Aufgaben erledigen können, in denen Menschen derzeit noch besser sind. Diese Definition scheint angemessen, da sie ebenfalls aktuelle Trends in der KI-Forschung reflektiert, z. B. jüngste Fortschritte in der computergestützten Bild- und Spracherkennung. Bei beiden Aufgaben handelt es sich um Beispiele, bei welchen der Mensch dem Computer stets überlegen war. Neben maschinellen Lernen umfasst KI u. a. Methoden der Logik und Wissensrepräsentation (z. B. Aussage- und Prädikantenlogik, semantische Netze), des Suchens und Problemlösens (z. B. Tiefen- und Breitensuche, (Meta-)Heuristiken) sowie des probabilistischen Schließens (z. B. Bayes'sche Netze, Fuzzy-Logik) (Ertel 2016; Russell und Norvig 2016).

Der Begriff »Maschinelles Lernen« (ML) wurde erstmals von Samuel (1959) erwähnt, der ein Computerprogramm für das Spiel »Checkers« schrieb, welches aus vergangenen Spielzügen eine eigene Spielstrategie erlernt. Der Autor definiert ML als *„field of study that gives computers the ability to learn without being explicitly programmed"*. Sinngemäß übersetzt: ML beschäftigt sich mit Methoden, die einem Computer die Fähigkeit geben, Handlungsweisen zu erlernen, ohne diese explizit zu programmieren. Samuels Intention, ML zu nutzen, um den Programmieraufwand zu reduzieren, ist auch heute noch eines der Leitmotive der Forschung und Entwicklung von ML-Methoden. Eine aktuellere Definition von ML stammt von Mitchell (1997, S. 2): *„A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E."* Sinngemäß übersetzt: Ein Computerprogramm lernt aus Erfahrung E, in Bezug auf eine Aufgabe T und ein Leistungsmaß P, wenn sich seine Leistung bei T gemessen an P mit der Erfahrung E verbessert. In der Praxis bedeutet dies, dass mithilfe von ML-Methodens Daten analysiert werden können, um Muster zu erkennen und

Zusammenhänge zu erforschen und auf Basis dieser Erkenntnisse Prognosen aufzustellen sowie Entscheidungen zu treffen. Wie Abbildung 3.1 zeigt, existieren drei grundsätzliche ML-Paradigmen – darunter das bestärkende Lernen –, welche für unterschiedliche Probleme und Aufgaben angewandt werden. Im Folgenden sollen die Grundzüge der an das bestärkende Lernen angrenzenden Paradigmen »überwachtes Lernen« und »unüberwachtes Lernen« dargelegt werden.

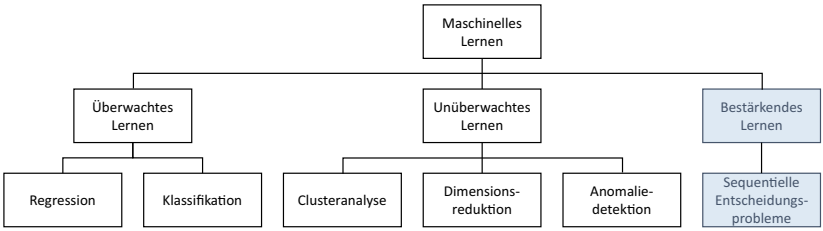


Abbildung 3.1 Paradigmen des maschinellen Lernens (in Anlehnung an Swamynathan 2019, S. 83)

3.1.1 Überwachtes Lernen als angrenzendes Paradigma

Grundvoraussetzung zur Anwendung einer überwachten Lernstrategie ist ein Trainingsdatensatz $X \cup Y$, bei dem jeder Eintrag i aus Merkmalen $x_i \in X$ (Eingangsgrößen) und einer zugehörigen Beobachtung $y_i \in Y$ (Ausgangsgröße, auch »Label« genannt) besteht. Auf Basis dieser Daten wird nun mithilfe eines Modells h_θ versucht eine Funktion zu approximieren, welche so präzise wie möglich von X auf Y abbildet $h_\theta : X \rightarrow Y$ (Mello und Ponti 2018, S. 5). Die Bezeichnung »Überwachtes Lernen« rührt daher, dass iterativ Prädiktionen mithilfe des anzulernenden Modells für beliebige Eingaben aus X generiert, diese mit der jeweils zugehörigen erwarteten Ausgabe aus Y verglichen und schließlich die Modellparameter θ, basierend auf den Fehler zwischen der Prädiktion $h_\theta(x_i)$ und erwarteter Ausgabe y_i, anpasst werden (z. B. mithilfe eines Gradientenabstiegsverfahrens). Abbildung 3.2 illustriert den Prozess des überwachten Lernens (a) und veranschaulicht diesen am Beispiel eines einfachen linearen Regressionsmodells (b).

Überwachte Lernstrategien können sowohl für Regressions- als auch Klassifikationsprobleme angewandt werden. Bei einem Regressionsproblem ist die

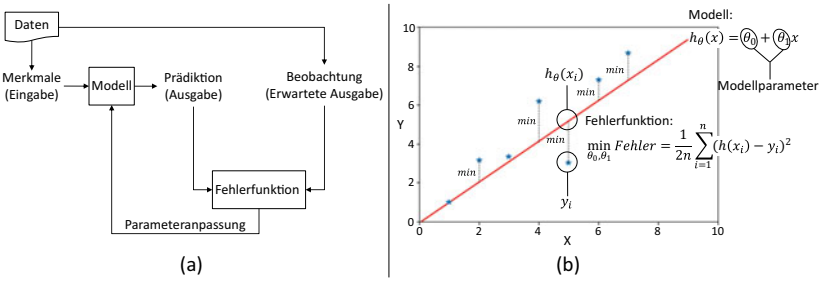


Abbildung 3.2 Prozess des überwachten Lernens (a) am Beispiel eines einfachen linearen Regressionsproblems (b)

Zielstellung, eine kontinuierliche Funktion zu approximieren, die möglichst präzise den Zusammenhang zwischen eingehenden Merkmalen und einer zu beobachtenden Ausgangsgröße beschreibt, z. B. die Vorhersage von Umsatzzahlen (Beobachtung) in Abhängigkeit von saisonalen, geografischen und / oder demografischen Daten (Merkmale). Hingegen wird bei einem Klassifikationsproblem die Approximation einer kontinuierlichen (bei binären Klassifikationsproblemen) oder diskreten Funktion (bei nichtbinären Klassifikationsproblemen) angestrebt, die auf Basis eingehender Merkmale eine möglichst eindeutige Klassenzuordnung aus einer definierten Menge an Klassen trifft. Ein Beispiel für eine binäre Klassifikationsaufgabe ist das Anlernen eines E-Mail-Spamfilters, welcher eine E-Mail als Spam (= 1) oder nicht Spam (= 0) klassifiziert. Ein Beispiel für eine nichtbinäre Klassifikationsaufgabe ist das Anlernen eines Bilderkennungsmodells, welches basierend auf Pixeldaten (Merkmale) und einer definierten Anzahl vorgegebener Klassen entscheiden soll, was auf dem Bild dargestellt wird (Swamynathan 2019, S. 83).

3.1.2 Unüberwachten Lernens als angrenzendes Paradigma

Beim unüberwachten Lernen besteht der Trainingsdatensatz lediglich aus Eingangsmerkmalen X, ohne dass eine erwartete Ausgabe Y bekannt ist. Die Lernaufgabe besteht darin, in den Eingangsmerkmalen Ähnlichkeiten und Differenzen zu erkennen und hierauf basierend eine Gruppierung der Daten abzuleiten. Da im Vergleich zum überwachten Lernen keine erwartete Ausgabe bekannt ist, wird beim unüberwachten Lernen mithilfe von Ähnlichkeits- und Distanzmaßen

die zugrundeliegende Struktur des Datensatzes untersucht. Auf diese Weise können unbekannte Relationen und Zusammenhänge in den Daten entdeckt werden (vgl. Celebi und Aydin 2016).

Typische Anwendungen für das unüberwachte Lernen sind Clusteranalysen, Dimensionsreduktion sowie Anomaliedetektion in Datensätzen. Bei einer Clusteranalyse werden Daten hinsichtlich ihrer (Un-)Ähnlichkeit bewertet und gruppiert. Beispielsweise können Kunden auf Basis hinterlegter Stammdaten (Geschlecht, Alter, Wohnort u. v. m.) segmentiert werden. Die in einer Clusteranalyse ermittelten Gruppen können darüber hinaus verwendet werden, um Anomalien in Datenströmen zu entdecken, z. B. durch Überprüfung, ob sich ein Datenpunkt außerhalb der kritischen Distanz zu allen bestehenden Clusterzentren befindet. Zielstellung einer Dimensionsreduktion ist es, einen großen multidimensionalen Eingangsdatensatz auf einen kleineren Dimensionsraum zu komprimieren, um z. B. Schlüsselvariablen mit dem größten Informationsgehalt zu identifizieren (Swamynathan 2019, S. 83 f.).

Abbildung 3.3 zeigt den Prozess des unüberwachten Lernens (a) am konkreten Beispiel einer Clusteranalyse (b). Weitgehend lässt sich das Prozessmodell ebenfalls auf die Anomaliedetektion sowie viele Dimensionsreduktionsverfahren übertragen. Analog zur Clusteranzahl muss bspw. beim Dimensionsreduktionsverfahren »Hauptkomponentenanalyse« die Anzahl der Hauptkomponenten festgelegt werden, auf welche ein Datensatz reduziert werden soll (Dunteman 1989).

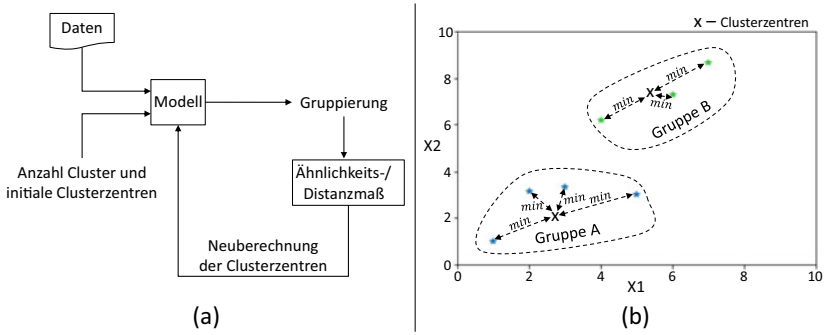


Abbildung 3.3 Prozess des unüberwachten Lernens (a) am Beispiel einer Clusteranalyse (b)

3.2 Grundprinzip und Taxonomie des bestärkenden Lernens

Das bestärkende Lernen (RL – vom englischen Begriff »Reinforcement Learning«) stellt neben dem überwachten und unüberwachten Lernen das dritte ML-Paradigma dar. RL hat einen psychologiewissenschaftlichen Ursprung im konditionierenden Lernen sowie ingenieurwissenschaftliche Wurzeln in der Kontrolltheorie und -optimierung (Sutton und Barto 2018, S. 13). Vor diesem Hintergrund ändern sich einige Begriffe, die für das überwachte und unüberwachte Lernen in Abschnitt 3.1.1 bzw. 3.1.2 eingeführt wurden. In RL werden ein anzulernendes Modell als »Agent«, in das Modell eingehende Daten als »Zustand« und die Modellausgabe als »Aktion« bezeichnet. Des Weiteren wird unter dem Agentenbegriff ebenfalls das RL-Verfahren mitsamt dessen Modellen, Untermethoden und Datenstrukturen subsumiert. Mit diesem Wissen kann nun das Grundprinzip von RL erläutert werden.

Die Grundidee von RL besteht darin, einen Agenten anzulernen, der mit seiner Umgebung interagiert. Der Agent wird durch ein ML-Modell repräsentiert, z. B. eine Tabelle, eine regelbasierte Logik, eine parametrisierbare Funktion oder ein künstliches neuronales Netz (KNN). Die modernsten und ebenfalls im Rahmen dieser Arbeit untersuchten RL-Methoden ordnen sich dem sogenannten tiefen bestärkenden Lernen (DRL – vom englischen Begriff »Deep Reinforcement Learning«) zu. DRL-Methoden zeichnen sich dadurch aus, dass der Agent stets durch ein KNN mit mehreren versteckten Schichten repräsentiert wird (vgl. François-Lavet et al. 2018, S. 219 f.). Ein Exkurs hinsichtlich des Aufbaus und der Funktionsweise von KNN ist als Anhang A dem elektronischen Zusatzmaterial beigefügt. Der Agent übersetzt Zustände seiner Umgebung (Eingangsgrößen) in Aktionen (Ausgangsgrößen). Jedes Mal, wenn der Agent eine Aktion ausführt, geht die Umgebung in einen Folgezustand über, für welchen der Agent die nächste Aktion auswählen muss. Im Gegensatz zum überwachten Lernen sind keine erwarteten Ausgaben für die Anwendung von RL-Methoden erforderlich. Jedoch verzichten RL-Methoden nicht gänzlich auf Trainingslabels wie das unüberwachte Lernen, sondern generieren diese während des Trainingsprozesses mithilfe einer sogenannten Belohnungsfunktion. Die Belohnungsfunktion bewertet die Aktionen eines Agenten mit positiven (Belohnung) oder negativen Werten (Bestrafung). Ziel des Agenten ist es, über die Zeit die erhaltene Belohnung zu maximieren bzw. die erhaltene Bestrafung zu minimieren. Auf diese Weise lernt der Agent, die beste Aktion für einen gegebenen Zustand zu prognostizieren (Sutton und Barto 2018, S. 48; François-Lavet et al. 2018, S. 234; Lorenz 2020, S. 4).

Um die Funktionsprinzipien von RL-Methoden im Detail darzulegen und in einem Prozessmodell analog zum überwachten (Abbildung 3.2 (a)) und unüberwachten Lernen (Abbildung 3.3 (a)) darzustellen, muss im Folgenden eine Fallunterscheidung getroffen werden. In dieser Arbeit werden RL-Methoden in gradientenabhängige und gradientenfreie Ansätze untergliedert. Bezogen auf die inhaltliche Ausrichtung der Übersichts- und Grundlagenliteratur, z. B. (Sutton und Barto 2018; Arulkumaran et al. 2017; François-Lavet et al. 2018; Lorenz 2020), repräsentieren gradientenabhängige Verfahren die traditionelle Sicht auf RL. Darüber hinaus werden im Rahmen dieser Arbeit ML-Methoden untersucht, die ebenfalls durch Belohnung / Bestrafung lernen, jedoch keine Gradienten erfordern, um die Parameter eines ML-Modells zu aktualisieren. Diese Methoden werden im Folgenden als gradientenfreie Verfahren bezeichnet. Gradientenabhängige und gradientenfreie Ansätze können ferner ansatzspezifisch klassifiziert werden. Abbildung 3.4 präsentiert eine Taxonomie von RL-Methoden, welche dieser Arbeit zugrunde gelegt wird. In den folgenden Abschnitten sollen die gradientenabhängigen und gradientenfreien RL-Ansätze zzgl. ihrer untergeordneten Kategorien näher beleuchtet und anhand von konkreten Methoden erklärt werden.

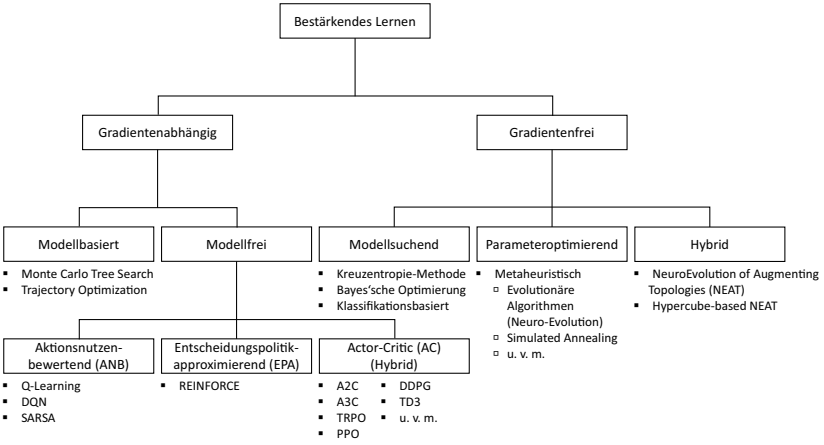


Abbildung 3.4 Taxonomie von bestärkenden Lernverfahren

3.3 Gradientenabhängiges bestärkendes Lernen

Gradientenabhängige RL-Verfahren trainieren einen Agenten in Richtung der Gradienten von dessen Parametern, die sich auf Basis der Differenz zwischen der Agentenausgabe und der aus der Belohnungsfunktion resultierenden erwarteten Ausgabe berechnen. Gradientenabhängige RL-Verfahren untergliedern sich in modellbasierte und modellfreie Ansätze.

In modellbasierten Ansätzen lernt der Agent ein Modell der Umgebung, mit welchem Entscheidungen vorausgeplant werden, bevor diese in der tatsächlichen Umgebung Anwendung finden. Ein populäres Verfahren dieser Art ist der Monte-Carlo-Tree-Search (MCTS) -Algorithmus (Coulom 2006). Ausgehend vom aktuellen Zustand konstruiert MCTS einen Entscheidungsbaum, wobei Knoten Zustände und Kanten Aktionen repräsentieren. Die kumulierten Belohnungen werden in den jeweiligen Knoten gespeichert, wodurch der Agent erfolgsversprechende Aktionen im Voraus kennt. MCTS funktioniert ebenfalls ohne den Einsatz von ML-Methoden, indem der Entscheidungsbaum durch zufällige Aktionen konstruiert wird. Bei Problemen mit großen Zustandsräumen kommt die konventionelle MCTS an ihre rechnerischen Grenzen, z. B. beim Brettspiel »Go«, in welchem ca. 250^{150} unterschiedliche Sequenzen von Spielzügen existieren. In diesem Fall kann mithilfe von ML-Verfahren die Effizienz der Exploration des Entscheidungsbaums erhöht werden. Zum einem, indem der Agent erfahrungsbasiert anstatt randomisiert Aktionen wählt, um zu entscheiden, ob und welche Baumverzweigungen ausgelotet werden, zum anderen, indem der Agent lernt, die Vorteilhaftigkeit eines Zustands zu bewerten, um frühzeitig zu entscheiden, ob Baumverzweigungen weiter ausgelotet werden sollen (Silver et al. 2016; Silver et al. 2017). Die MCTS ist nur für RL-Probleme mit diskreten Aktionsräumen anwendbar, d. h. dass das ML-Modell durch ein Klassifikator-KNN (vgl. Abbildung A-1 (b) in Anhang A im elektronischen Zusatzmaterial) oder ein anderes Klassifikationsmodell repräsentiert wird. Für RL-Probleme mit kontinuierlichem Aktionsraum, bei welchen das ML-Modell durch ein Regressor-KNN (vgl. Abbildung A-1 (a) in Anhang A im elektronischen Zusatzmaterial) oder ein anderes Regressionsmodell repräsentiert wird, kann bspw. der Trajectory-Optimization-Algorithmus als modellbasierter Ansatz verwendet werden. Sofern das RL-Problem als vollständig differenzierbares Modell abbildbar ist, kann eine Entscheidungspolitik analytisch berechnet werden, indem die erhaltenen Belohnungen entlang der Sequenz von getroffenen Aktionen zu beobachteten Zuständen rückpropagiert werden. Für stochastische Umgebungen, bei welchen der Folgezustand für eine gewählte Aktion variieren kann, ist es möglich ein Modell auf Basis eines Gauß-Prozesses oder eines tiefen KNN anzulernen, welches die

stochastische Dynamik der Umgebung approximiert und mit dessen Hilfe verschiedene Aktionssequenzen im Voraus evaluiert werden können (François-Lavet et al. 2018, S. 265 f.). Die Funktionsweise von modellbasierten Ansätzen trägt der Zielstellung dieser Arbeit keine Rechnung, RL-Methoden für die echtzeitfähige Produktionsablaufplanung verfügbar zu machen. Aufgrund der Tatsache, dass die mit der Produktionsablaufplanung verbundenen Allokations- und Sequenzierungsprobleme diskreter Natur sind, käme theoretisch MCTS als modellbasierter Ansatz in Frage. Praktisch erfordert die iterative Auswertung von Lösungen und hierauf basierend die Konstruktion eines Entscheidungsbaums eine hohe Rechenzeit und schließt somit den echtzeitfähigen Einsatz von MCTS aus. Aus diesem Grund werden modellbasierte RL-Ansätze von einer detaillierteren Betrachtung ausgeschlossen.

Hingegen lernen modellfreie Ansätze ausschließlich aufgrund der direkten Interaktion mit der Umgebung und der erhaltenen Belohnung. Gradientenabhängige modelfreie Ansätze untergliedern sich in aktionsnutzen-bewertende, entscheidungspolitik-approximierende und hybride Verfahren. Diese werden in den Abschnitten 3.3.3, 3.3.4 und 3.3.5 näher erläutert. Grundvoraussetzung für die Anwendung gradientenabhängiger Verfahren ist, dass die Umgebung, mit welcher der Agent interagiert, ein sogenanntes Markov-Entscheidungsproblem repräsentiert.

3.3.1 Markov-Entscheidungsproblem

Ein Markov-Entscheidungsproblem (MEP) formalisiert einen zeitdiskreten stochastischen Steuerungsprozess als mathematisches Modell. Es existieren verschiedene MEP-Definitionen, u. a. von Hu und Yue (2008), Chang et al. (2013) und François-Lavet et al. (2018). Allen Definitionen liegt zugrunde, dass ein MEP mindestens durch einen 4-Tupel $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$ beschrieben wird, wobei

- $\mathcal{S}$ die Menge möglicher Zustände repräsentiert,
- $\mathcal{A}$ die Menge möglicher Aktionen repräsentiert, wobei es sich um eine Obermenge der möglichen Aktionen für einen Zustand $S_t \in \mathcal{S}$ handeln kann $\mathcal{A} \supseteq \mathcal{A}(S_t)$,
- $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ die Übergangsfunktion repräsentiert, welche die Wahrscheinlichkeit angibt, von einem beliebigen Zustand $S_t \in \mathcal{S}$ durch die Auswahl einer Aktion $A_t \in \mathcal{A}$ in einen Folgezustand $S_{t+1} \in \mathcal{S}$ überzugehen, und

- $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ die Belohnungsfunktion repräsentiert, die den Übergang von einem beliebigen Zustand $S_t \in \mathcal{S}$ in einen Folgezustand $S_{t+1} \in \mathcal{S}$ durch Auswahl einer Aktion $A_t \in \mathcal{A}(S_t)$ bewertet.

Ferner liegt nur dann ein MEP vor, wenn die sogenannte »Markov-Eigenschaft« gilt, d. h. dass die Übergangswahrscheinlichkeit zu einem Folgezustand S_{t+1} und die erhaltene Belohnung R_{t+1} lediglich vom aktuellen Zustand S_t und der gewählten Aktion A_t abhängen, nicht jedoch von Zuständen und Aktionen beeinflusst werden, die weiter zurück in der Vergangenheit liegen. Es gilt somit $p(S_{t+1}, R_{t+1}|S_t, A_t)$ (Sutton und Barto 2018, S. 48). Vor diesem Hintergrund gestaltet sich die Interaktion zwischen Agent und Umgebung gemäß Abbildung 3.5.

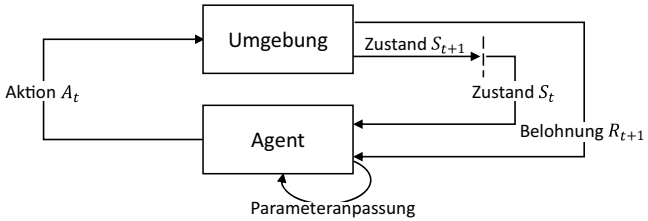


Abbildung 3.5 Prozess des gradientenabhängigen bestärkenden Lernens (in Anlehnung an Sutton und Barto 2018, S. 48)

Die Interaktion verläuft in diskreten Zeitschritten $t = 0, 1, 2, \ldots, T$. Zu jedem Zeitschritt t beobachtet der Agent den Zustand der Umgebung S_t und wählt eine Aktion $A_t \in \mathcal{A}(S_t)$. Durch Anwendung einer Aktion erhöht sich die Zeit inkrementell $t + 1$ und die Umgebung geht in den Folgezustand S_{t+1} über. Zeitgleich wird eine Belohnung R_{t+1} für die im vergangenen Zeitschritt gewählte Aktion A_t ausgegeben. Zum Zweck der Übersichtlichkeit und Verständlichkeit vernachlässigt Abbildung 3.5, dass zum Zeitpunkt t (sofern $t \neq 0$) eine Belohnung R_t für die Aktion A_{t-1} ausgegeben wird.

3.3.2 Nutzenfunktion

Im Allgemeinen verfolgen gradientenabhängige RL-Ansätze die Zielstellung, ausgehend von einem aktuellen Zeitpunkt t, die Summe der in Zukunft erhaltenen Belohnungen $G_t = R_{t+1} + R_{t+2} + \ldots + R_T$ durch Auswahl geeigneter Aktionen

zu maximieren. Hierbei werden gewöhnlich zukünftige Belohnungen mit einem Faktor $0 < \gamma \leq 1$ exponentiell zum Zeitschritt diskontiert, sodass gilt:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=1}^{\infty} \gamma^{k-1} R_{t+k} \tag{3.1}$$

Zum einen hat die Diskontierung einen mathematischen Hintergrund, weil in nichtterminierten Umgebungen, in welchen kein definierter Endzustand existiert (z. B. ein Produktionssystem mit rollierendem Auftragshorizont), die Summe erhaltener Belohnungen ohne Diskontierung gegen unendlich laufen würde. Damit ein RL-Verfahren konvergiert, ist es jedoch notwendig, dass die Summe erhaltener Belohnung gegen einen Grenzwert verläuft. Zum anderen werden oftmals auch in terminierten Umgebungen Belohnungen diskontiert, um Belohnungen in naher Zukunft gegenüber Belohnungen in ferner Zukunft zu priorisieren (Pardo et al. 2018; Sutton und Barto 2018, S. 54 f.)

Die von gradientenabhängigen RL-Ansätzen verfolgte Zielstellung stellt somit ein dynamisches Stufenoptimierungsproblem dar, welches theoretisch mithilfe von DP-Verfahren (siehe Abschnitt 2.3.3.1) exakt gelöst werden kann. Die optimale kumulierte Belohnung G_t^*, gemessen ab dem aktuellen Zeitschritt t, ergibt sich aus der Sequenz von Aktionen, welche die erhaltene Belohnung in Summe maximiert.

$$G_t^* = \max_{\{a_k\}_{k=t}^{\infty}} \sum_{k=t}^{\infty} \gamma^{k-1} \mathcal{R}(S_k, a_k) \tag{3.2}$$

Es sei darauf hingewiesen, dass diejenige Aktion in Zeitschritt k, die in $k+1$ die maximale Belohnung ausschüttet, nicht zwingend die optimale Auswahl darstellt. Aufgrund der Tatsache, dass aus unterschiedlichen Aktionen unterschiedliche Folgezustände resultieren, kann z. B. die Auswahl einer schlechteren Aktion zu einem Folgezustand führen, in dem wiederum die beste Aktion eine wesentlich höhere Belohnung verspricht als die jeweils maximal verfügbare Belohnung in anderen Zuständen. Hieraus resultiert ein komplexer Entscheidungsbaum, der im äußersten Fall $\mathcal{A}^T$ potenzielle Lösungen für den Zeithorizont T enthält, unter der Annahme, dass in jedem Zustand S_t alle Aktionen $\mathcal{A}$ zur Auswahl stehen. Der Lösungsraum wächst somit exponentiell in Abhängigkeit vom betrachteten Zeithorizont. In vielen Problemen sind der Zustands- und Aktionsraum sowie

der betrachtete Zeithorizont zu groß, als dass alle möglichen Entscheidungssequenzen zur Identifizierung des Optimums in annehmbarer Zeit durchprobiert werden können. Um dieses Problem zu lösen, machen sich DP-Verfahren im Allgemeinen und RL-Verfahren im Speziellen das Optimalitätsprinzip von Bellman zu Nutze, welches für alle dynamischen Optimierungsprobleme gilt, die als MEP modelliert werden können. Es besagt (Bellman 1957, S. 83): *„An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision"*. Sinngemäß übersetzt: Eine optimale Entscheidungspolitik besitzt die Eigenschaft, dass, unabhängig von dem initialen Zustand und der ersten Entscheidung, die verbleibenden Entscheidungen eine optimale Entscheidungspolitik in Abhängigkeit von der ersten Entscheidung des resultierenden Folgezustands bilden müssen. Hieraus folgt, dass sich eine optimale Entscheidungspolitik aus optimalen Teilpolitiken zusammensetzt (Lew und Mauch 2007, S. 5). Formel (3.2) kann somit wie folgt umformuliert werden, sodass die erste Entscheidung zum Zeitpunkt t von allen zukünftigen Entscheidungen ab $k = (t+1, t+2, \ldots, T)$ separiert wird.

$$G_t^* = \max_{a_t} \mathcal{R}(S_t, a_t) + \gamma \left(\max_{\{a_k\}_{k=t+1}^{\infty}} \sum_{k=t+1}^{\infty} \gamma^{k-1} \mathcal{R}(S_k, a_k) \right) \tag{3.3}$$

Bei Betrachtung von Formel (3.2) und (3.3) wird ersichtlich, dass Formel (3.2) dem rechten Term in Formel (3.3) für $k = t+1$ entspricht. Formel (3.2) für $k = t+1$ kann wiederum in Formel (3.3) überführt werden. Hieraus folgt, dass es sich bei Formel (3.3) um eine rekursive Funktion handelt, die wie folgt vereinfacht werden kann.

$$v_*(S_t) = \max_{a_t} \mathcal{R}(S_t, a_t) + \gamma v_*(S_{t+1}) \tag{3.4}$$

Es handelt sich bei Formel (3.4) um die sogenannte Bellman-Gleichung, wobei $v_*(S_t)$ für einen beobachteten Zustand S_t die Gesamtbelohnung in der Zukunft unter Einhaltung der optimalen Entscheidungspolitik wiedergibt. Die Funktion von $v_*(S_{t+1})$ wird auch als Zustandsnutzenfunktion (unter optimaler Entscheidungspolitik $*$) genannt, da sie den Nutzen V (vom englischen Begriff »Value«) eines Zustands bewertet. Die Bellman-Gleichung ist eine Funktionalgleichung (vgl. Dempe und Schreier 2006, S. 210; Domschke et al. 2015, S. 172), d. h. eine Gleichung, die aus einer endlichen Anzahl von unbekannten Funktionen und

unabhängigen Variablen besteht (Aczél 1960, S. 19). Die Lösung der Bellman-Gleichung bedeutet die unbekannte Funktion V zu ermitteln, welche ausgehend von einem Anfangszustand die maximal mögliche Belohnung in der Zukunft berechnet (Sutton und Barto 2018, S. 60).

Vor diesem Hintergrund versuchen gradientenabhängige RL-Verfahren durch Interaktion mit der Umgebung und auf Basis erhaltener Belohnungen die Nutzenfunktion zu approximieren. Es existieren zwei Varianten von Nutzenfunktionen (Sutton und Barto 2018, S. 58). Die Zustandsnutzenfunktion $v_\pi(s)$ schätzt die kumulierte zukünftige Belohnung, ausgehend von Zustand s und unter Anwendung von Entscheidungspolitik π.

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=1}^{\infty} \gamma^{k-1} R_{t+k}|S_t = s\right] \tag{3.5}$$

Hierbei definiert $\mathbb{E}_\pi[\cdot]$ den Erwartungswert für eine Zufallsvariable. Im Fall von Formel (3.5) handelt es sich um den Erwartungswert der kumulierten zukünftigen Belohnung ab Zeitpunkt t, unter der Voraussetzung, dass der Agent Entscheidungspolitik π anwendet. Die zweite Variante ist die Aktionsnutzenfunktion $q_\pi(s, a)$, die ausgehend von einem Zustand s die kumulierte zukünftige Belohnung für die Auswahl von Aktion a schätzt, unter der Annahme, dass nach Anwendung von Aktion a die Entscheidungspolitik π weiterverfolgt wird.

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=1}^{\infty} \gamma^{k-1} R_{t+k}|S_t = s, A_t = a\right] \tag{3.6}$$

Ob, welche und wie eine Nutzenfunktion angewandt wird, ist abhängig von der jeweiligen RL-Verfahrensklasse. Im Folgenden sollen diese erklärt und anhand einiger Beispiele demonstriert werden.

3.3.3 Aktionsnutzen-bewertende Verfahren

Die Zielstellung von aktionsnutzen-bewertenden (ANB) RL-Verfahren besteht darin die Aktionsnutzenfunktion $q_\pi(s, a)$ für ein sequenzielles Entscheidungsproblem zu erlernen. Die eigentliche Entscheidungspolitik wird von der gelernten Aktionsnutzenfunktion abgeleitet (François-Lavet et al. 2018, S. 242). In der

Regel wird die Aktion mit dem höchsten prognostizierten Aktionsnutzen ausgewählt, d. h. mit der höchsten prognostizierten kumulierten Belohnung in der Zukunft. Im Folgenden soll das Prinzip der ANB-Verfahren anhand des DQN-Algorithmus erklärt werden.

Der DQN (Deep Q-Networks) -Algorithmus wurde erstmals von Mnih et al. (2013) vorgestellt. Bekannter jedoch ist der Folgeartikel (Mnih et al. 2015), in welchem die Methode detaillierter erklärt wird. DQN baut konzeptionell auf dem Q-Learning-Algorithmus von Watkins (1989) auf. Sowohl DQN als auch das klassische Q-Learning beruhen auf der Annahme, dass der Aktionsraum eines Agenten endlich und diskret ist. Für einen gegebenen Zustand bewerten beide Algorithmen die Vorteilhaftigkeit jeder möglichen Aktion. Der Agent wählt entweder die Aktion mit dem höchsten Nutzen oder eine zufällige Aktion. Die Auswahl der Aktion mit dem höchsten Nutzen repräsentiert die von der Aktionsnutzenfunktion abgeleitete Entscheidungspolitik. Die Auswahl zufälliger Aktionen ist notwendig, um den Lösungsraum zu erkunden. Mit fortschreitender Zeit sinkt jedoch die Wahrscheinlichkeit, eine zufällige Aktion zu wählen. Der Hauptunterschied zwischen dem Q-Learning- und dem DQN-Verfahren liegt im verwendeten Agentenmodell. Beim Q-Learning wird der Agent i. d. R. durch eine Tabelle dargestellt (Watkins und Dayan 1992). Hingegen wird bei DQN der Agent durch ein tiefes KNN repräsentiert. Darüber hinaus besitzt DQN einige weitere Komponenten, die den Lernprozess unterstützen. Abbildung 3.6 illustriert die Architektur und den Ablauf des DQN-Algorithmus.

Der Trainingsprozess gestaltet sich wie folgt: Durch die Interaktion zwischen dem Agenten und seiner Umgebung werden Beobachtungen[1] generiert, die im Erfahrungsspeicher gesammelt werden. Eine Beobachtung ist ein 5-Tupel, bestehend aus dem Zustand S_t, der gewählten Aktion A_t, der erhaltenen Belohnung R_{t+1}, dem resultierenden Folgezustand S_{t+1} und dem Binärwert $ENDE_{t+1}$, der signalisiert, ob die Umgebung in S_{t+1} terminiert. Immer dann, wenn eine neue Beobachtung dem Erfahrungsspeicher hinzugefügt wird (und sofern der Erfahrungsspeicher bereits eine ausreichende Anzahl an Beobachtungen enthält), wird ein Trainingsschritt durchgeführt. Zu diesem Zweck wird ein Los mit zufälligen Beobachtungen aus dem Erfahrungsspeicher generiert. Die Zustände S_t und Folgezustände S_{t+1} werden vorwärts durch zwei unterschiedlich parametrisierte KNN $q_\pi(s, a; \theta)$ und $q_\pi\left(s, a; \theta^-\right)$ propagiert, um die entsprechenden Aktionsnutzen Q_t und Q_{t+1} zu bestimmen. Beide KNN haben zum Ziel, die Aktionsnutzenfunktion q_π zu approximieren. Das KNN

[1] Nicht zu verwechseln mit dem Begriff »Beobachtung« im überwachten Lernen, welche die erwartete Ausgabe / das Trainingslabel bezeichnet.

$q_\pi(s, a; \theta)$ repräsentiert den anzulernenden Agenten, der auch außerhalb des Trainingsprozesses eingesetzt werden soll. Bei $q_\pi\left(s, a; \theta^-\right)$ handelt es sich um eine Kopie von $q_\pi(s, a; \theta)$, dessen trainierbaren Parameter maximal c Zeitschritte in der Vergangenheit von $q_\pi(s, a; \theta)$ liegen. Das KNN $q_\pi\left(s, a; \theta^-\right)$ kommt lediglich während des Trainings zum Einsatz. Q_t und Q_{t+1} sind $|\mathcal{A}|$-dimensionale Vektoren, wobei jedes Vektorelement die erwartete kumulierte Belohnung in der Zukunft für die Auswahl der entsprechenden Aktion beschreibt. Q_t und Q_{t+1} werden für die Berechnung eines Fehlersignals verwendet, das mittels Backpropagation durch das KNN $q_\pi(s, a; \theta)$ rückgeführt wird. Vor diesem Hintergrund werden für die Berechnung von Q_t und Q_{t+1} unterschiedlich parametrisierte KNN verwendet, um unerwünschte Korrelationen zwischen Q_t und Q_{t+1} und damit einhergehend Oszillationen und Divergenzen im Trainingsprozess zu vermeiden. Im Vergleich zum konventionellen Q-Learning, das ohne einen zusätzlichen Ziel-Aktionsnutzenfunktion-Approximator $q_\pi\left(s, a; \theta^-\right)$ arbeitet und bei welchem eine Veränderung von Q_t oftmals zu einer ähnlichen und gleichgerichteten Veränderung von Q_{t+1} führt, wird auf diese Weise der Trainingsprozess stabilisiert.

Die Berechnung des Fehlersignals erfolgt in zwei Schritten. Im ersten Schritt wird die Zielgröße U_t für den Nutzen der gewählten Aktion Q_{t,A_t} mit der folgenden Formel geschätzt.

$$U_t = R_{t+1} + (1 - ENDE_{t+1})\gamma \max Q_{t+1} \tag{3.7}$$

Formel (3.7) entspricht der Bellman-Gleichung (3.4), wobei die Rekursivfunktion des zweiten Terms durch $\max Q_{t+1}$ angenähert wird. Des Weiteren bildet Formel (3.7) zwei Fälle ab. Sofern die Umgebung in S_{t+1} terminiert ($ENDE_{t+1} = 1$), wird lediglich die Belohnung R_{t+1} als Trainingslabel verwendet. Andernfalls wird zusätzlich die diskontierte maximal erwartete Belohnung für den Folgezustand S_{t+1} aufaddiert. Der Fehler der Ausgabeschicht L des Agenten berechnet sich wie folgt.

$$\delta^{(L)} = \left(U_t - Q_{t,A_t}\right)^2 \tag{3.8}$$

Durch das Backpropagation-Verfahren werden nun die Gradienten von $q_\pi(s, a; \theta)$ ermittelt und die Parameter des KNN aktualisiert. Das geschilderte Lernverfahren wird als »Temporal-Difference (TD) Learning« bezeichnet. Es zeichnet sich dadurch aus, dass für die Berechnung des Fehlersignals nicht die tatsächlich kumulierte Belohnung in der Zukunft G_t, sondern lediglich der prognostizierte

Aktionsnutzen im darauffolgenden Zeitschritt $t + 1$ berücksichtigt wird (Sutton und Barto 2018, S. 119). Der Lernprozess wiederholt sich für jeden Zeitschritt über mehrere Episoden[2]. Der Ziel-Aktionsnutzenfunktion-Approximator $q_\pi(s, a; \theta^-)$ wird aktualisiert, indem dessen Parameter nach n Zeitschritten (wobei n freiwählbar ist, jedoch größer eins sein sollte) durch die Parameter des Aktionsnutzenfunktion-Approximator $q_\pi(s, a; \theta)$ überschrieben werden.

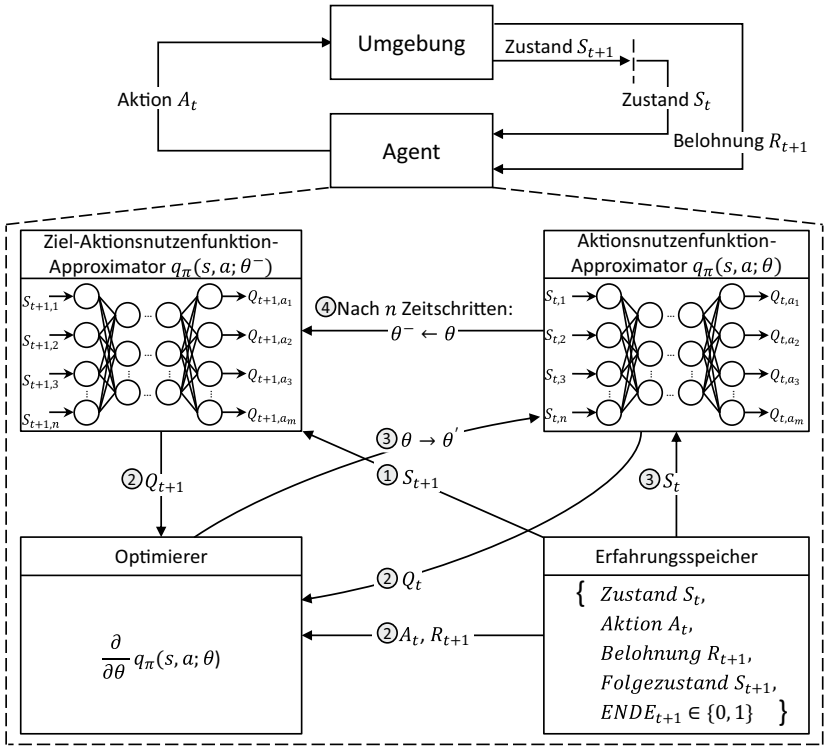


Abbildung 3.6 DQN-Architektur und algorithmischer Ablauf (in Anlehnung an Lang et al. 2020a)

[2] Sequenz von Zuständen und Aktionen, die eine Umgebung vom Anfangszustand S_0 in den Endzustand S_T überführt.

Bei DQN handelt es sich um eine sogenannte Off-Policy-Methode, d. h., dass der Algorithmus nicht mit Trainingsbeispielen lernt, die aus der aktuellen Entscheidungspolitik π resultieren. Wie bereits beschrieben generiert DQN stattdessen Trainingslose aus dem Erfahrungsspeicher, der ebenfalls Beobachtungen enthalten kann, die der Agent mit vergangenen Parametern θ^- gesammelt hat. Hingegen ist die ursprüngliche Version des SARSA-Algorithmus, die ebenfalls zu den ANB-Methoden zählt, als On-Policy-Methode klassifiziert, da diese auf einen Erfahrungsspeicher verzichtet und ausschließlich mit Beobachtungen trainiert, die mit den aktuellen Parametern θ gesammelt wurden.

3.3.4 Entscheidungspolitik-approximierende Verfahren

In entscheidungspolitik-approximierenden (EPA) Verfahren verfolgt das Agentenmodell die Zielstellung die Entscheidungspolitik direkt zu lernen. Hieraus resultiert, dass die Auswahl einer Aktion a nicht mehr ausschließlich vom aktuellen Zustand s, sondern ebenfalls von den momentanen Agentenparametern θ abhängt, sodass die Entscheidungspolitik als $\pi(a|s, \theta)$ definiert ist. Im Gegensatz zu ANB-Verfahren wird die Entscheidungspolitik nicht mehr implizit aus einer angelernten Aktionsnutzenfunktion abgeleitet, sondern explizit durch den Agenten repräsentiert. Die Ausgabe des Agenten entspricht somit nicht mehr der erwarteten zukünftigen kumulierten Belohnung für jede mögliche Aktion (Sutton und Barto 2018, S. 321). Stattdessen repräsentiert sie bei deterministischen Verfahren die unmittelbare Aktion. Hingegen wird bei stochastischen Verfahren die Agentenausgabe als Parameter einer Wahrscheinlichkeitsverteilung interpretiert, mit welcher zufällige Aktionen generiert werden (vgl. Silver et al. 2014). Bei Problemen mit diskreten Aktionsräumen werden i. d. R. die Parameter einer multinominalen Verteilung gesucht, während bei kontinuierlichen Problemen gewöhnlich die Parameter einer Normalverteilung prognostiziert werden (Arulkumaran et al. 2017).

Vor diesem Hintergrund sind die Lernkonzepte von ANB- und EPA-Verfahren grundlegend unterschiedlich. Während bei ANB-Verfahren der Fehler zwischen den erwarteten und prädiktierten Aktionsnutzen mittels Gradientenabstieg minimiert werden soll, haben EPA-Verfahren das Ziel, auf Basis einer agentenparameterabhängigen Performanzfunktion $J(\theta)$ die Güte einer Entscheidungspolitik per Gradientenaufstieg zu maximieren.

$$\theta' = \theta + \alpha\widehat{\nabla J(\theta)} \tag{3.9}$$

Hierbei entsprechen θ bzw. θ' den Agentenparametern vor bzw. nach der Aktualisierung, α der Lernrate und $\widehat{\nabla J(\theta)}$ dem Gradienten einer geschätzten Performanzfunktion (Sutton und Barto 2018, S. 322). Das Hauptproblem von EPA-Verfahren ist, dass die Performanzfunktion $J(\theta)$ unbekannt ist. Gesucht wird eine Funktion, welche die kumulierte Belohnung über alle Zustände und Aktionen in Abhängigkeit von den Agentenparametern θ maximiert, sodass eine gradientenabhängige Änderung der Agentenparameter zu einer Verbesserung der Entscheidungspolitik führt. Anders ausgedrückt wird eine θ-abhängige Variante von Formel (3.3) gesucht, welche die optimale Gesamtbelohnung G_t^* ermittelt. Für stochastische EPA-Verfahren adressieren Sutton et al. (1999) dieses Problem mit dem »Policy Gradient Theorem«, das den folgenden mathematischen Zusammenhang beweist.

$$\nabla J(\theta) \propto \sum_s \left(\mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s;\theta) \right) \tag{3.10}$$

Formel (3.10) besagt, dass sich der Gradient der unbekannten Performanzfunktion $\nabla J(\theta)$ proportional verhält zum Summenprodukt der Eintrittswahrscheinlichkeit eines Umgebungszustands $\mu(s)$ sowie des inneren Summenprodukts der prognostizierten Aktionsnutzen $q_\pi(s,a)$ und der Gradienten der Entscheidungspolitik $\nabla\pi(a|s;\theta)$. Um nun eine Entscheidungspolitik $\pi(a|s;\theta)$ zu approximieren, ist es nicht notwendig, die Eintrittswahrscheinlichkeiten des kompletten Zustandsraums (äußere Summe über s), geschweige denn den Nutzen jeder Aktion in jedem Zustand (innere Summe über a) zu kennen. Zunächst kann sich die Tatsache zu Nutze gemacht werden, dass sich die Wahrscheinlichkeit für den Eintritt eines Zustands $\mu(s)$ folgendermaßen berechnet (Sutton und Barto 2018, S. 199).

$$\mu(s) = \frac{\eta(s)}{\sum_{s'} \eta(s')} \tag{3.11}$$

Hierbei repräsentiert $\eta(s)$ eine Funktion, die wiedergibt, wie oft ein Zustand s eingetreten ist. Demzufolge wirkt $\mu(s)$ als gewichtender Koeffizient, der sich über die äußere Summe von Formel (3.10) zu eins aufaddiert. Die folgende Formel (3.12) verdeutlicht, dass die rechte Seite von Formel (3.10) einem gewichteten Mittelwert über den Zustandsraum $\mathcal{S}$ entspricht. Ferner folgt hieraus in Formel (3.13), dass sich ein auf Zustandsstichproben (S_t) basierender Erwartungswert $\mathbb{E}_\pi[\cdot]$ ebenfalls proportional zum Performanzgradienten verhalten muss (vgl. Sutton und Barto 2018, S. 326).

$$\nabla J(\theta) \propto \sum_s \left(\mu(s) \sum_a q_\pi(s,a) \nabla \pi(a|s;\theta) \right) \tag{3.10}$$

$$\propto \sum_s \left(\frac{\eta(s) \sum_a q_\pi(s,a) \nabla \pi(a|s;\theta)}{\sum_{s'} \eta(s')} \right) \tag{3.12}$$

$$\propto \mathbb{E}_\pi \left[\sum_a q_\pi(S_t,a) \nabla \pi(a|S_t;\theta) \right] \tag{3.13}$$

Darüber hinaus kann auch die innere Summe über a durch Formulierung eines Erwartungswerts eliminiert werden. Analog zu Formel (3.12) ist hierfür eine gewichtete Mittelung über den Zustandsraum $\mathcal{A}$ erforderlich. Als Gewichtungskoeffizient eignet sich der Term $\pi(a|s;\theta)$, siehe Formel (3.14). Nun kann, wie in Formel (3.15) dargestellt, ein auf Aktionsstichproben (A_t) basierender Erwartungswert kalkuliert werden, der sich ebenfalls proportional zum Performanzgradienten verhält. Aus Formel (3.6) ist bereits bekannt, dass für einen eingetretenen Zustand S_t und eine gewählte Aktion A_t der Aktionsnutzen $q_\pi(S_t, A_t)$ dem Erwartungswert der kumulierten Gesamtbelohnung $\mathbb{E}_\pi[G_t]$ ab Zeitpunkt t entspricht. Sofern die Umgebung irgendwann in einem finalen Zustand S_T terminiert, kann die Gesamtbelohnung G_t durch Anwendung der Entscheidungspolitik π ab S_t bis S_T beobachtet werden. Vor diesem Hintergrund ist das Training eines Approximators der Aktionsnutzenfunktion $q_\pi(s,a)$ nicht notwendig, siehe Formel (3.16). Ausgehend von Formel (3.13) kann die rechte Seite des ursprünglichen Policy Gradient Theorem (3.10) wie folgt vereinfacht werden (vgl. Sutton und Barto 2018, S. 327).

$$\nabla J(\theta) \propto \mathbb{E}_\pi \left[\sum_a q_\pi(S_t,a) \nabla \pi(a|S_t;\theta) \right] \tag{3.13}$$

$$\propto \mathbb{E}_\pi \left[\sum_a \left(\frac{\pi(a|S_t;\theta) q_\pi(S_t,a) \nabla \pi(a|S_t;\theta)}{\sum_{a'} \pi(a'|S_t;\theta)} \right) \right] \tag{3.14}$$

$$\propto \mathbb{E}_\pi \left[q_\pi(S_t,A_t) \frac{\nabla \pi(A_t|S_t;\theta)}{\pi(A_t|S_t;\theta)} \right] \tag{3.15}$$

$$\propto \mathbb{E}_\pi \left[G_t \frac{\nabla \pi(A_t|S_t;\theta)}{\pi(A_t|S_t;\theta)} \right] \tag{3.16}$$

Formel (3.16) findet u. a. im REINFORCE-Algorithmus (Williams 1988, 1992) Anwendung, um die Parameter eines EPA-Agenten zu aktualisieren. Bezugnehmend auf Formel (3.9) definiert sich die Aktualisierungsvorschrift im REINFORCE-Algorithmus gemäß Formel (3.17), wobei der Gradient in Abhängigkeit vom Zeitschritt mit γ^t diskontiert wird. Der Einfachheit halber wird in der Literatur der Gradient oftmals in der verkürzten logarithmischen Form angegeben, siehe Formel (3.18). Dies ist zulässig, da $\nabla \ln x$ und $\nabla x/x$ äquivalent zueinander sind (Sutton und Barto 2018, S. 327 f.).

$$\theta_{t+1} = \theta_t + \alpha \gamma^t G_t \frac{\nabla \pi(A_t|S_t;\theta)}{\pi(A_t|S_t;\theta)} \tag{3.17}$$

$$= \theta_t + \alpha \gamma^t G_t \nabla \log \pi(A_t|S_t;\theta) \tag{3.18}$$

Der Ablauf des REINFORCE-Algorithmus ist vergleichsweise einfach. Zunächst wird die Entscheidungspolitik $\pi(a|s;\theta)$ (Agent) mit beliebigen θ-Werten initialisiert. Danach werden die folgenden Schritte wiederholt, bis der Agent zu einer lokalen optimalen Entscheidungspolitik konvergiert: *(i)* Der Agent und die Umgebung interagieren über eine vollständige Episode (von $t = 0$ bis T) miteinander, wobei der Agent die aktuelle Entscheidungspolitik $\pi(a|s;\theta)$ verfolgt. Die dabei resultierende Trajektorie $S_0, A_0, R_1, S_1, A_1, R_2, \ldots, S_{T-1}, A_{T-1}, R_T$ wird gespeichert. *(ii)* Die Trajektorie wird für jeden Zeitschritt durchlaufen, wobei in jedem Zeitschritt die kumulierte Belohnung G_t gemäß Formel (3.1) berechnet wird und die Agentenparameter θ gemäß Formel (3.18) aktualisiert werden. *(iii)* Die Umgebung wird in den Anfangszustand S_0 zurückversetzt.

Der erste Nachteil von REINFORCE ist dessen geringe Konvergenzgeschwindigkeit. Das Verfahren muss gewöhnlich eine hohe Anzahl von Episoden durchlaufen, da die Monte-Carlo-basierte Schätzung des Gradienten einer hohen Varianz unterliegt (Sutton et al. 1999; François-Lavet et al. 2018, S. 256).

Der zweite Nachteil von REINFORCE ist, dass der Agent lediglich eine stochastische Entscheidungspolitik erlernen kann. Wie bereits erwähnt gilt das Policy Gradient Theorem in Formel (3.10) nur für stochastische EPA-Verfahren. Darüber hinaus existiert das Policy Gradient Theorem für deterministischen EPA-Verfahren, welches von Silver et al. (2014) hergeleitet und in Formel (3.19) dargestellt ist. Um eine proportionale Verhältnismäßigkeit zum Performanzgradienten herzustellen, wird zusätzlich der Gradient der Aktionsnutzenfunktion benötigt.

$$\nabla J(\theta) \propto \mathbb{E}_\pi[\nabla \pi(s;\theta) \nabla q_\pi(s,a)] \tag{3.19}$$

Da REINFORCE die Aktionsnutzen nicht funktional schätzt, sondern direkt durch Interaktion mit der Umgebung ermittelt, besitzt das Verfahren keine Informationen über die Aktionsnutzenfunktion (respektive ihres Gradienten) und kann demzufolge nicht für das Anlernen von deterministischen Entscheidungspolitiken eingesetzt werden.

Im Folgenden soll eine dritte Art von gradientenabhängigen modellfreien RL-Verfahren diskutiert werden, welche die beiden Nachteile adressiert.

3.3.5 Actor-Critic-Verfahren

Actor-Critic (AC) -Verfahren kombinieren die Vorteile von ANB- und EPA-Verfahren, indem sie zwei ML-Modelle lernen: Das sogenannte »Actor«-Modell, das die Entscheidungspolitik und gleichzeitig den mit der Umwelt interagierenden Agenten repräsentiert, sowie das sogenannte »Critic«-Modell, das verfahrensabhängig entweder die Zustandsnutzenfunktion $v_\pi(s;\theta)$ oder die Aktionsnutzenfunktion $q_\pi(s,a;\theta)$ erlernt. Im Gegensatz zu klassischen EPA-Verfahren wie REINFORCE sind AC-Verfahren zum Lernen nicht auf die tatsächliche Gesamtbelohnung G_t angewiesen. Sie müssen demzufolge keine vollständige Episode der Umgebung beobachten. Stattdessen verwenden sie das Critic-Modell, um die kumulierte Belohnung in der Zukunft zu prognostizieren, sowie TD-Techniken zur Generierung von Lernsignalen. Somit sind AC-Verfahren nicht auf episodische RL-Probleme beschränkt, sondern können ebenfalls für Probleme angewandt werden, bei welchen die Umgebung in keinem finalen Zustand S_T terminiert. Erste Steuerungsansätze, die nach dem AC-Prinzip arbeiten, werden bereits in den Arbeiten von Witten (1977) und Barto et al. (1983) diskutiert. Die Bezeichnung »Actor-Critic«, die als Oberbegriff ebensolche hybriden RL-Verfahren zusammenfasst, wurde mutmaßlich durch Williams und Baird III (1993) eingeführt und insbesondere durch die Arbeiten von Sutton et al. (1999) sowie von Konda und Borkar (1999) bzw. Konda und Tsitsiklis (1999) etabliert.

Die im ersten Absatz referenzierten Arbeiten beziehen sich im weitesten Sinne auf das ursprüngliche AC-Konzept, in welchem das Critic-Modell die Zustandsnutzenfunktion $v_\pi(s;\theta_C)$ und das Actor-Modell eine stochastische Entscheidungspolitik $\pi(a|s;\theta_A)$ lernt. Im Folgenden werden diese Verfahren als stochastische AC-Verfahren bezeichnet. Wie eingangs erwähnt sind AC-Verfahren im Allgemeinen nicht auf die tatsächliche Gesamtbelohnung

G_t zur Verbesserung der Entscheidungspolitik angewiesen. Im Kontrast zum REINFORCE-Algorithmus verwenden stochastische AC-Verfahren im Speziellen das vereinfachte Policy Gradient Theorems in Formel (3.13), um die Parameter des Actor-Modells zu optimieren. Ferner nutzen stochastische AC-Verfahren gewöhnlich »N-Step-Bootstrapping« (Sutton und Barto 2018, S. 141) zur Generierung der erwarteten Aktionsnutzen U_k (entspricht dem ersten Term $q_\pi(S_t; a)$ in Formel (3.13)). N-Step-Bootstrapping repräsentiert eine Verallgemeinerung des TD-Lernverfahrens, welches bereits für den DQN-Algorithmus in Abschnitt 3.3.3 beschrieben wurde. Ausgehend von einem beliebigen Zustand S_t erstellt das Actor-Modell eine partielle Trajektorie über n Zeitschritte S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}, R_{t+2}, …, $S_{t+(n-1)}$, $A_{t+(n-1)}$, R_{t+n}, S_{t+n}. Das Critic-Modell prognostiziert für alle beobachteten Zustände die Zustandsnutzen $V_k = v_\pi(S_k) \forall k = (t, \ldots, t+n)$. Aufgrund der Tatsache, dass für die Zeitschritte t bis $t + (n-1)$ die gewählten Aktionen des Actor-Modells bekannt sind, können durch Rückwärtsrechnung, ausgehend vom letzten Zustandsnutzen V_{t+n}, die Zielwerte der Aktionsnutzen wie folgt ermittelt werden.

$$U_k = \begin{cases} V_k, & falls\ k = t+n \\ R_{k+1} + \gamma U_{k+1} & sonst \end{cases} \quad \forall k = (t+n, \ldots, t) \tag{3.20}$$

Von den Ziel-Aktionsnutzen U_k werden die prognostizierten Zustandsnutzen V_k subtrahiert. Das Ergebnis ist der sogenannte »Advantage«. Auf Basis der Advantage-Werte können nun die Lernsignale für das Actor- und Critic-Modell generiert werden. Wie bereits erwähnt basiert die Parameteroptimierung des Actor-Modells auf dem Policy Gradient Theorem in Formel (3.13). Anstelle der prognostizierten Aktionsnutzen $q_\pi(S_t, a)$ werden die Advantage-Werte zur Berechnung des Lernsignals verwendet. Hierbei repräsentieren die Advantage-Werte nichts Anderes als die Aktionsnutzen, die durch Subtraktion der prognostizierten Zustandsnutzen normalisiert wurden, um (im Gegensatz zum konventionellen REINFORCE-Algorithmus) die Varianz des Lernsignals zu reduzieren und somit das Konvergenzverhalten während des Trainings zu verbessern. Analog zur Berechnung des Fehlers der Agentenausgabe im DQN-Algorithmus (Formel (3.8)) ermittelt sich das Lernsignal des Critic-Modells durch Quadrieren des Advantages. Das geschilderte Lernprinzip wird ebenfalls von modernen DRL-Algorithmen verwendet, z. B. dem »Advantage Actor-Critic« (A2C) -Algorithmus sowie dessen asynchron parallelisierte Variante, dem A3C-Algorithmus (Mnih et al. 2016). Eine ausführliche Beschreibung des A2C-Algorithmus ist als Anhang B dem elektronischen Zusatzmaterial beigefügt.

Darüber hinaus existieren weitere stochastische AC-Verfahren, welche nach demselben Prinzip Advantage-Werte berechnen und das Critic-Modell aktualisieren, jedoch andere Konzepte zum Anlernen des Actor-Modells verfolgen. Dazu zählen z. B. der »Trust Region Policy Optimization« (TRPO) -Algorithmus (Schulman et al. 2015) sowie der auf derselben Idee basierende, jedoch mathematisch einfachere »Proximal Policy Optimization« (PPO) -Algorithmus (Schulman et al. 2017). Im Gegensatz zum A2C- und A3C-Algorithmus ändern TRPO und PPO die Parameter des Actor-Modells nicht ausschließlich auf Basis der aktuellen Entscheidungspolitik. Beide Verfahren basieren auf der Überlegung, dass bereits kleinste Veränderungen der Agentenparameter zu großen Veränderungen in der Entscheidungspolitik führen können. Vor diesem Hintergrund vergleichen TRPO und PPO für dieselbe Sequenz von Zuständen die Eintrittswahrscheinlichkeiten der jeweils gewählten Aktion A_t unter der aktuellen Entscheidungspolitik $\pi(a|s;\theta)$ und unter den veralteten Entscheidungspolitik $\pi(a|s;\theta^-)$. Für die Gradientenbestimmung werden die Eintrittswahrscheinlichkeiten $\pi(A_t|S_t;\theta)$ und $\pi(A_t|S_t;\theta^-)$ ins Verhältnis gesetzt und mit dem Advantage gewichtet. Das Ergebnis ist der sogenannte »Surrogate-Advantage«, welcher anstelle des Policy Gradient Theorem in Formel (3.13) für die Generierung des Actor-Lernsignals verwendet wird. Konkret verfolgen das TRPO- und PPO-Verfahren unterschiedliche Berechnungsvorschriften zur Bildung des Lernsignals. Der TRPO-Algorithmus löst näherungsweise mittels Taylor-Reihenentwicklung ein nichtlineares Optimierungsproblem, welches die Maximierung des Surrogate-Advantage postuliert, unter der Nebenbedingung, dass der Abstand zwischen $\pi(A_t|S_t;\theta)$ und $\pi(A_t|S_t;\theta^-)$, gemessen über die Kullback-Leibler-Divergenz (Kullback und Leibler 1951), kleiner gleich einer benutzerdefinierten Schranke ist. Hingegen wendet der PPO-Algorithmus ein einfaches Clipping-Verfahren auf den Surrogate-Advantage an, um eine zu starke positive bzw. negative Differenz zwischen $\pi(a|S_t;\theta)$ und $\pi(a|S_t;\theta^-)$ durch eine konstante obere bzw. untere Schranke zu begrenzen. Der PPO-Algorithmus wird in Kapitel 6 für ein Problem der Produktionsablaufplanung untersucht. Aus diesem Grund ist eine ausführliche Beschreibung des PPO-Algorithmus als Anhang C dem elektronischen Zusatzmaterial beigefügt.

Neben den stochastischen AC-Verfahen hat sich aufbauend auf dem deterministischen Policy Gradient Theorem (Silver et al. 2014) ein weiteres AC-Konzept etabliert, bei welchem das Actor-Modell eine deterministische Entscheidungspolitik $\pi(s;\theta)$ und das Critic-Modell die Aktionsnutzenfunktion $q_\pi(s,a;\theta)$ erlernt. Das erste und wohl populärste Verfahren dieser Art, das mehrschichtige KNN zur Repräsentation des Actor- und Critic-Modells verwendet, ist der »Deep Deterministic Policy Gradient« (DDPG) –Algorithmus (Lillicrap

et al. 2015). Stellvertretend für die deterministischen EPA-Verfahren wird der DDPG-Algorithmus in Anhang D im elektronischen Zusatzmaterial detailliert dargelegt.

3.4 Gradientenfreies bestärkendes Lernen

Gemäß der dieser Arbeit zugrundeliegenden Taxonomie (Abbildung 3.4) existieren neben gradientenabhängigen RL-Verfahren ebenfalls gradientenfreie RL-Verfahren, deren Eigenschaften im Folgenden dargelegt werden. Es sei an dieser Stelle erwähnt, dass es keinen wissenschaftlichen Konsens hinsichtlich der Einteilung von RL-Verfahren in gradientenabhängige und gradientenfreie Ansätze zu geben scheint. In einigen Grundlagen- und Übersichtswerken finden gradientenfreie RL-Ansätze keine Erwähnung, z. B. (Sutton und Barto 2018; François-Lavet et al. 2018; Lorenz 2020), während andere populäre Arbeiten im Bereich diese explizit aufführen, z. B. (Schulman et al. 2015; Arulkumaran et al. 2017). Ein ausführlicher Übersichtsartikel zu gradientenfreien RL-Verfahren wurde erst jüngst von Qian und Yu (2021) unter der Bezeichnung »Derivative-Free Reinforcement Learning« (zu Deutsch: „Ableitungsfreies Bestärkendes Lernen“) veröffentlicht. Nach besten Wissen handelt es sich hierbei um die derzeit einzige Übersichtsarbeit zu diesem Thema. Wie in Kapitel 6 dieser Arbeit noch aufgezeigt wird, scheinen für einige Probleme der Produktionsablaufplanung gradientenfreie gegenüber gradientenabhängigen Methoden überlegen zu sein. Aus diesem Grund werden gradientenfreie RL-Ansätze in dieser Arbeit ebenfalls berücksichtigt.

Ihrer Bezeichnung entsprechend verzichten gradientenfreie RL-Verfahren auf die Berechnung von Gradienten für das Training von Agenten. Stattdessen verwenden sie stochastische bzw. stochastisch-heuristische Verfahren, mit welchen sie den Parameterraum größtenteils zufällig durchsuchen, anstatt sich – wie bei gradientenabhängigen Verfahren üblich – gerichtet zu einer (lokalen) optimalen Lösung zu bewegen. Ein weiterer Unterschied ist die Art und Frequenz der Belohnungsvergabe. Während in gradientenabhängigen RL-Ansätzen der Agent für jede Aktion eine Belohnung erhält, wird in gradientenfreien Ansätzen lediglich am Ende der Episode eine Belohnung vergeben. Der Grund ist, dass alle gradientenabhängigen RL-Methoden ein differenzierbares Lernsignal benötigen, von welchem sie die Gradienten der Agentenparameter ableiten, um diese zu optimieren. Die Zielfunktionen zur Optimierung von Produktionsablaufplänen (siehe Abschnitt 2.3.1.3) sind i. d. R. nicht differenzierbar. Stattdessen setzen gradientenabhängige RL-Methoden zur Generierung eines Lernsignals auf

TD-Lernverfahren (siehe S. 45), bei welchen die Gradienten auf Basis der Differenz zwischen der erwarteten kumulierten Gesamtbelohnung V_t, ausgehend vom aktuellen Zustand S_t, und der erwarteten kumulierten Gesamtbelohnung V_{t+1}, ausgehend vom darauffolgenden Zustand S_{t+1}, berechnet werden. Das Optimierungsproblem wird auf diese Weise auf das Teilproblem reduziert, den Prognosefehler zwischen der erwarteten kumulierten Gesamtbelohnung zum aktuellen und zum darauffolgenden Zeitpunkt zu minimieren.

Weil hingegen gradientenfreie RL-Verfahren die Parameter eines Agenten stochastisch aktualisieren, sind diese nicht auf ein differenzierbares Lernsignal angewiesen und können somit eine Lösung unmittelbar anhand des eigentlichen Optimierungskriteriums bewerten. Dementsprechend unterscheidet sich ebenfalls die Interaktion zwischen Agent und Umwelt. Obgleich zur Lösungsgenerierung weiterhin eine Sequenz von Zuständen und Aktionen durchlaufen wird, ist es nicht notwendig für jede Aktion eine Belohnung zu berechnen. Wie in Abbildung 3.7 dargestellt, bewertet die Umgebung stattdessen die Güte von vollständigen Lösungskandidaten mit einer sogenannten »Fitness«, wobei diese i. d. R. kongruent mit der zu optimierenden Zielfunktion ist (Qian und Yu 2021).

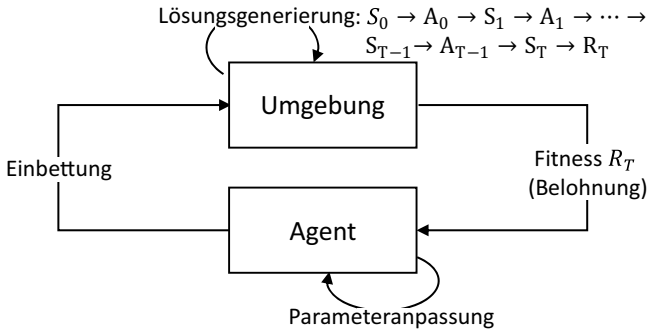


Abbildung 3.7 Prozess des gradientenfreien bestärkenden Lernens

Hierfür wird zu Beginn jeder Episode eine Kopie des Agentenmodells mit dem aktuellen Parameterbild als Steuerungslogik in die Umgebung eingebettet. Darauffolgend wird die Episode durchlaufen und die Zielfunktion berechnet. Das Ergebnis wird als Fitness-Wert an den Agenten zurückgemeldet, worauf der Agent seine Parameter aktualisiert. Die Aktualisierung der Parameter ist stets zufallsbasiert und entweder unabhängig oder teilweise abhängig vom

Fitness-Wert. Im zweiten Fall kann der Fitness-Wert bspw. als Indikator dienen, um den Bereich des Lösungsraums einzugrenzen, aus welchem die nächste Zufallsstichprobe gezogen werden soll.

Ein weiterer Unterschied zu gradientenabhängigen RL-Verfahren ist, dass die Probleme, auf die gradientenfreie RL-Verfahren angewandt werden, keinem Markov-Entscheidungsproblem entsprechen müssen, weil das Training unabhängig von der Transitionsdynamik stattfindet.

Entsprechend der Taxonomie bestärkender Lernverfahren (Abbildung 3.4) werden in dieser Arbeit gradientenfreie RL-Verfahren in modellsuchende und parameteroptimierende Verfahren kategorisiert (in Anlehnung an Qian und Yu 2021) sowie ferner in hybride Verfahren, welche die Eigenschaften der beiden erstgenannten Verfahrensklassen vereinen. Auf die ersten beiden Verfahrensarten soll im folgenden Unterabschnitt lediglich kurz eingegangen werden. Darauffolgend wird in Abschnitt 3.4.2 der Algorithmus »NeuroEvolution of Augmenting Topologies« (NEAT) als Vertreter der hybriden Verfahren detailliert erläutert. NEAT vereint alle charakteristischen Eigenschaften modellsuchender und parameteroptimierender Verfahren, sodass der Algorithmus geeignet ist, um die Arbeitsweise gradientenfreier RL-Verfahren im Allgemeinen zu veranschaulichen.

3.4.1 Modellsuchende und parameteroptimierende Verfahren

Modellsuchende Verfahren können als Pendant der gradientenabhängigen modellbasierten RL-Verfahren betrachtet werden. Sie zeichnen sich dadurch aus, dass sie iterativ die Parameter eines stochastischen Modells anpassen, welches entweder der Generierung von Lösungskandidaten dient oder welches die Agentenumgebung auf bestimmte Weise approximiert. Mithilfe der komplexeren und laufzeitaufwändigeren Agentenumgebung wird die wahre Lösungsgüte der Kandidaten bewertet und das stochastische Modell aktualisiert. Die Kreuzentropie-Methode, die Bayes'sche Optimierung und klassifikationsbasierte Ansätze sind typische Vertreter für modellsuchende Verfahren.

Die Kreuzentropie-Methode (Rubinstein 1997; 1999) verwendet als stochastisches Modell eine beliebige Wahrscheinlichkeitsverteilung mit zufällig initialisierten Parametern. Hierbei kann es sich bspw. um eine (multivariate) Normalverteilung mit den Parametern μ (Mittelwert) und σ (Standardabweichung) oder um eine Multinominalverteilung mit den Parametern $n_1, \dots n_K$ (Anzahl Beobachtungen für die Klassen 1 bis K) bzw. $p_1, \dots p_K$ (Wahrscheinlichkeit

für die Klassen 1 bis K) handeln. Das stochastische Modell dient der Generierung von Lösungskandidaten. Über mehrere Iterationen werden jeweils mehrere zufällige Lösungskandidaten durch die gewählte Wahrscheinlichkeitsverteilung generiert und mithilfe der Agentenumgebung bewertet. Die besten Lösungskandidaten einer Iteration werden zur Aktualisierung der Modellparameter verwendet. In der darauffolgenden Iteration wird das aktualisierte Modell für die Generierung neuer Lösungskandidaten verwendet.

Die Bayes'sche Optimierung (BO) (Mockus 1975; 1989) konstruiert ein probabilistisches Modell der Agentenumgebung, das der Vorevaluierung von Lösungskandidaten dient. Das stochastische Modell wird gewöhnlich durch einen Gauß-Prozess repräsentiert. Analog zur Kreuzentropie-Methode aktualisiert BO das probabilistische Modell durch die stichprobenartige Evaluation von Lösungskandidaten mithilfe der Agentenumgebung. Das BO-Verfahren unterscheidet sich von der Kreuzentropie-Methode insbesondere durch zwei Aspekte. Erstens dient das BO-konstruierte probabilistische Modell nicht der Generierung von Lösungskandidaten, sondern der Prognose der Lösungsgüte von Kandidaten. Im Gegensatz zur Kreuzentropie-Methode werden Lösungskandidaten bspw. durch eine Rastersuche über den Eingabedatenraum (Bergstra et al. 2011), via Monte-Carlo-Simulation (Snoek et al. 2012) oder mittels anderer Verfahren (Shahriari et al. 2016) generiert. Zweitens werden Lösungskandidaten nicht in arbiträrer Reihenfolge durch die Agentenumgebung evaluiert, sondern mithilfe einer sogenannten Akquisitionsfunktion ausgewählt. Die Akquisitionsfunktion bestimmt die nächste zu evaluierende Lösung aus der Menge von Lösungskandidaten, z. B. gemäß der höchsten Wahrscheinlichkeit für eine Verbesserung (Kushner 1964) oder gemäß der höchsten erwarteten Verbesserung (Mockus et al. 1978). Das BO-Verfahren ist ebenfalls im Rahmen der in Kapitel 5 vorgestellten Methode von Bedeutung. Neben anderen Verfahren wird es für die Optimierung von Hyperparametern im Zuge des Agententrainings (Abschnitt 5.3.6) vorgeschlagen. Ergänzend zu dieser kurzen Erklärung ist aus diesem Grund eine detaillierte Beschreibung des BO-Verfahrens als Anhang E dem elektronischen Zusatzmaterial beigefügt.

Klassifikationsbasierte Ansätze bilden die dritte Kategorie modellsuchender Verfahren. Ähnlich zur Kreuzentropie-Methode werden Lösungskandidaten zufällig generiert. Das gesuchte Modell dient jedoch nicht der Generierung, sondern der Klassifikation von qualitativ hochwertigen und minderwertigen Lösungskandidaten. Das Klassifikationsmodell wird anhand der Evaluation von Lösungskandidaten in der Agentenumgebung trainiert. Im Einsatz soll das Klassifikationsmodell Lösungen herausfiltern, deren Evaluation durch die berechnungsaufwändige

Agentenumgebung nicht rentabel sind. Des Weiteren soll anhand der Kategorisierung prognostiziert werden, in welchen Regionen des Lösungsraums sich die erfolgversprechendsten Lösungskandidaten befinden. Auf diese Weise steuert das Klassifikationsmodell die Suche nach neuen Lösungen (Qian und Yu 2021). Ein Beispiel für klassifikationsbasierte Verfahren ist der RACOS (Randomized Coordinate Shrinking) -Algorithmus (Yu et al. 2016) bzw. dessen Variation SRACOS (Hu et al. 2017), welche speziell für sequenzielle Entscheidungsprobleme entwickelt wurde.

Hingegen verfolgen parameteroptimierende Verfahren die Zielstellung, geeignete Parameter für ein bekanntes Agentenmodell (z. B. KNN, Polynomfunktion u. v. m.) zu identifizieren, um auf diese Weise ein gegebenes Problem bestmöglich zu lösen. In den letzten Jahren wurden diesbezüglich insbesondere evolutionäre Algorithmen untersucht. Diese Ansätze werden unter dem Begriff »Neuro-Evolution« zusammengefasst. Bereits in den 1990er Jahren wurden Neuro-Evolution-Methoden für das Training einfacher KNN-Topologien untersucht. Stellvertretend sei auf die Arbeiten von Belew et al. (1990), Kitano (1990) und Whitley et al. (1990) verwiesen. Darüber hinaus wurden in den letzten Jahren Algorithmen entwickelt, welche sich ebenfalls zur Parametrisierung von tiefen mehrschichtigen KNN eignen (Such et al. 2017; Gangwani und Peng 2018). Neben evolutionären Algorithmen wurden in der Vergangenheit auch andere Metaheuristiken für die Parametrisierung von KNN untersucht, z. B. Simulated Annealing (Sexton et al. 1999). Eine ausführliche Übersicht zum Einsatz von Metaheuristiken für die Optimierung von KNN-Parametern findet sich in (Sharda et al. 2006).

3.4.2 Hybride Verfahren – NeuroEvolution of Augmenting Topologies

Gradientenfreie hybride RL-Verfahren können sowohl Agentenmodelle konstruieren als auch deren Parameter trainieren. Ferner sind sie ebenfalls in der Lage, die Struktur und / oder Parameter von gegebenen Modellen zu optimieren. Die bekanntesten hybriden Verfahren basieren auf evolutionären Algorithmen, dienen der Konstruktion sowie Parametrisierung von KNN-basierten Agentenmodellen und werden unter der Abkürzung »TWEANN« (Topology and Weight Evolving Artificial Neural Network) zusammengefasst (Stanley und Miikkulainen 2002b). Analog zu rein parameteroptimierenden Neuro-Evolution-Ansätzen reichen erste TWEANN-Verfahren ebenfalls bis in die 1990er Jahre zurück (Dasgupta und

McGregor 1992; Mandischer 1993; Zhang und Mühlenbein 1993; Angeline et al. 1994; u. v. m.).

Eines der populärsten TWEANN-Verfahren ist »NeuroEvolution of Augmenting Topologies« (NEAT), das von Stanley und Miikkulainen (2002a; 2002b, 2002c) entwickelt wurde. Im Rahmen dieser Arbeit soll NEAT als Stellvertreter der gradientenfreien RL-Verfahren untersucht werden. NEAT gehört zur Familie der genetischen Algorithmen, deren Grundprinzipien bereits in Abschnitt 2.3.3.3 skizziert wurden. Insbesondere aufgrund der folgenden vier Eigenschaften gilt NEAT als der erste Neuro-Evolution-Algorithmus, der die Topologie eines KNN effizient adaptieren kann (Stanley 2004):

1) Flexible Enkodierung von Genomen zur Laufzeit

Im Unterschied zu anderen neuroevolutionären Ansätzen ist die Kodierung von NEAT-Genomen dynamisch erweiterbar, d. h. nicht auf eine feste Dimension beschränkt. Jedes Genom besteht aus einer Reihe von neuronalen und synaptischen Genen. Ein neuronales Gen enthält alle Informationen eines Neurons, respektive dessen Bias-Wert und Aktivierungsfunktion. Darüber hinaus besitzt jedes neuronale Gen eine ID, über welche ebenfalls Rückschluss darüber gezogen werden kann, ob es sich um ein Eingabeneuron (I), ein verstecktes Neuron (H) oder ein Ausgabeneuron (O) handelt. Ein synaptisches Gen besitzt ebenfalls eine ID, aus der die miteinander verbundenen Neuronen abgeleitet werden können. Des Weiteren besitzt jedes synaptische Gen eine Gewichtung und einen Status (aktiviert/ deaktiviert). Ferner besitzt es eine Innovationsnummer. Die Innovationsnummer dient als Indikator, ob zwei Genome miteinander gekreuzt werden können. Das Konzept des Kreuzung und die Rolle der Innovationsnummer wird im Rahmen der zweiten Eigenschaft genauer erläutert. NEAT wendet zwei Arten von Mutationen an, um Genome zu erweitern und auf diese Weise die Struktur von KNN anzupassen. Eine synaptische Mutation erzeugt eine neue synaptische Verbindung zwischen zwei zuvor unverbundenen Neuronen, während eine neuronale Mutation eine bestehende Verbindung deaktiviert und stattdessen ein neues Neuron hinzufügt, das jeweils eine synaptische Verbindung zu den zuvor verbundenen Neuronen pflegt.

Abbildung 3.8 veranschaulicht die Kodierung von Genomen in NEAT sowie ein Beispiel für eine neuronale und eine synaptische Mutation. Im gezeigten Beispiel erfährt das ursprüngliche Genom (linke Abbildung) zunächst eine neuronale Mutation, durch welche zwischen dem Eingabeneuron $I2$ und dem Ausgabeneuron $O1$ ein neues verstecktes Neuron $H1$ entsteht und die Verbindungen $(I1, H1)$ und $(H1, O1)$ einhergehen (mittlere Abbildung). Nachfolgend durchläuft das Genom eine synaptische Mutation, durch welche die zusätzliche Verbindung

($I2$, $H1$) geschaffen wird (rechte Abbildung). Die dynamische Enkodierung von NEAT-Genomen zeichnet sich durch eine hohe Effizienz und Skalierbarkeit aus, weil die Größe eines Genoms der exakten Menge enkodierter Informationen entspricht und das Genom beliebig um weitere Gene erweitert werden kann, wenn in Folge von genetischen Operationen neue Informationen hinzukommen. Diesbezüglich unterscheidet sich NEAT von älteren Neuro-Evolution-Verfahren, bei welchen die Genomgröße fix ist und für nicht verwendete Gene Speicherplatz vorreserviert wird.

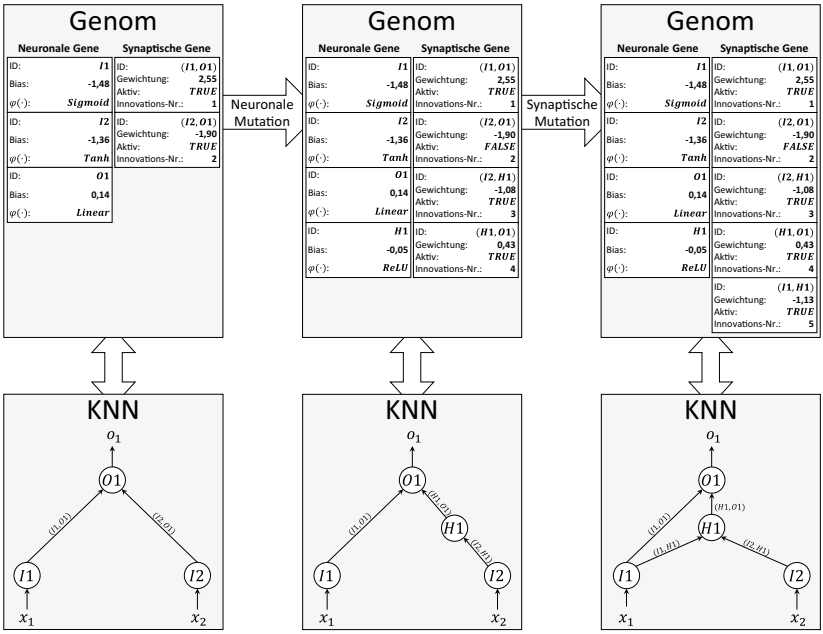


Abbildung 3.8 Genetische Enkodierung und Mutationen in NEAT (in Anlehnung an Lang et al. 2021b)

2) Rekombination von unterschiedlichen KNN-Topologien

Eine Kreuzung von Genomen, die unterschiedliche KNN-Topologien repräsentieren, ist nicht immer realisierbar. Sofern sich zwei Genome hinsichtlich ihrer Neuronen-Anzahl unterscheiden, können im Zuge von deren Kreuzung u. U.

ebensolche Gene ausgewählt werden, aus welchen eine nicht realisierbare KNN-Topologie resultieren würde. Diese Situation tritt bspw. dann auf, wenn das Nachkommen-Genom von einem der beiden Eltern-Genome ein synaptisches Gen erbt, zu dem es nicht die beiden neuronalen Gene besitzt, welche die zu verbindenden Neuronen repräsentieren. Um unzulässige Kreuzungen zu vermeiden, wird in NEAT der Ursprung jedes synaptischen Gens durch eine eindeutige Innovationsnummer nachgehalten. Darüber hinaus ist es nicht notwendig, ebenfalls den Ursprung von neuronalen Genen zu überwachen, da die ID eines synaptischen Gens bereits alle notwendigen Informationen über die Topologie eines KNN enthält, d. h. welche Neuronen existieren und wie diese untereinander verbunden sind. Bei der Kreuzung zweier Genome wählt NEAT nach dem Zufallsprinzip übereinstimmende synaptische Gene aus, d. h. Gene, die in beiden Elterngenomen vorhanden sind. Für die übrigen Gene, die sich unterscheiden, berücksichtigt NEAT nur die Gene des Elterngenoms mit der besseren Fitness. Falls beide Elternteile die gleiche Fitness besitzen, fügt NEAT die nicht übereinstimmenden synaptischen Gene beider Elternteile dem Nachkommen-Genom hinzu. Abbildung 3.9 veranschaulicht eine beispielhafte Kreuzung zwischen zwei Genomen, die sich partiell hinsichtlich ihrer synaptischen Gene unterscheiden. Im Beispiel stimmen die synaptischen Gene mit den Innovationsnummern 1 und 2 überein. Es wird angenommen, dass NEAT zufällig beide übereinstimmenden synaptischen Gene ausgewählt hat, um das Nachkommen-Genom zu konstruieren. Des Weiteren berücksichtigt NEAT alle nicht übereinstimmenden Gene im Nachkommen-Genom, was darauf hindeutet, dass die Eltern-Genome die gleiche Fitness besitzen. Die gekreuzten synaptischen Genome implizieren, dass das Nachkommen-Genom das neuronale Gen $H1$ vom ersten Eltern-Genom sowie das neuronale Gen $H2$ vom zweiten Eltern-Genom übernehmen muss.

3) Spezifizierung der Genom-Population

Sofern ein Genom eine topologische Veränderung durchläuft, ist es möglich, dass dieses nicht mit der Fitness von solchen Genomen konkurrieren kann, die über mehrere Generationen keine strukturellen Veränderungen durchliefen und ihre synaptischen Gewichte für dieselbe Topologie optimieren konnten. Um zu verhindern, dass topologisch veränderte Genome fitnessbedingt sofort aus der Population eliminiert werden, unterteilt NEAT die Gesamtpopulation in verschiedene Spezies gemäß von topologischen Ähnlichkeiten. Auf diese Weise konkurrieren nur topologisch ähnliche Genome innerhalb ihrer Spezies miteinander. Die (Un-)Ähnlichkeit zwischen zwei Genomen wird über ein Distanzmaß quantifiziert, wobei die Berechnung der genetischen Distanz von

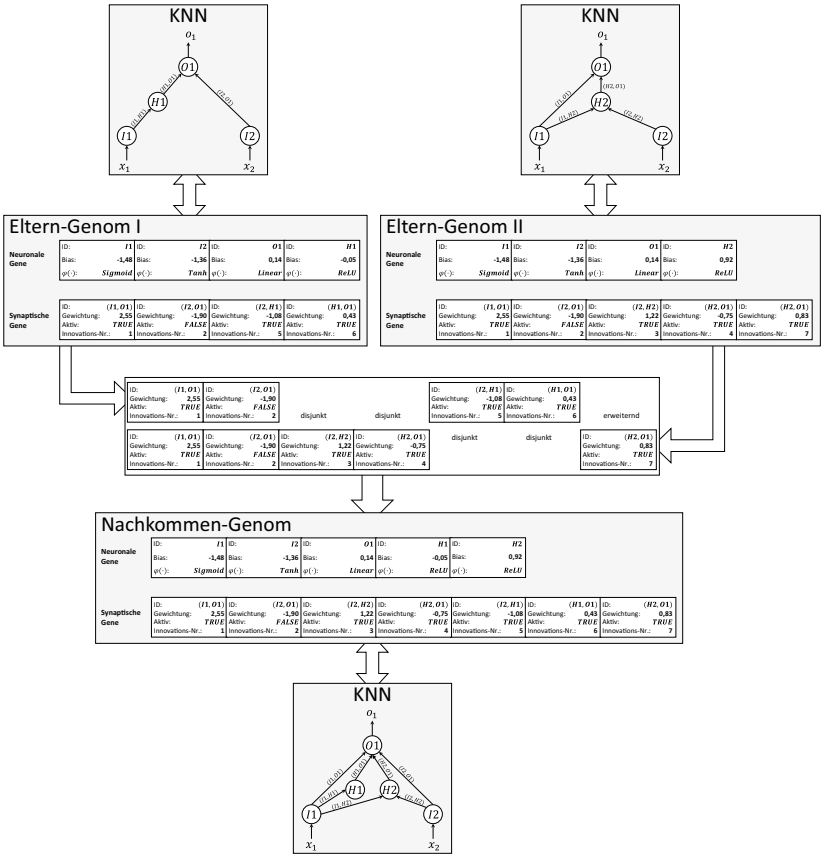


Abbildung 3.9 Beispiel für die Kreuzung von zwei Genomen (in Anlehnung an Lang et al. 2021b)

der jeweiligen Implementierung abhängt. Im Folgenden werden die Berechnungsvorschriften der Open-Source-Implementierung »NEAT-Python« (McIntyre et al. 2017a) angeführt, welche ebenfalls für die Experimente in Kapitel 6 verwendet wird.

$$D = d_n + d_s \tag{3.21}$$

$$d_n = \frac{1}{\max(N_1, N_2)} * \left(c_{sim} * \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} Dist_n(n_i, n_j) + c_{diff} * \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} Diff(n_i, n_j) \right) \tag{3.22}$$

$$d_s = \frac{1}{\max(S_1, S_2)} * \left(c_{sim} * \sum_{i=1}^{S_1} \sum_{j=1}^{S_2} Dist_s(s_i, s_j) + c_{diff} * \sum_{i=1}^{S_1} \sum_{j=1}^{S_2} Diff(s_i, s_j) \right) \tag{3.23}$$

Hierbei bezeichnen D, d_n und d_s die gesamte, neuronale und synaptische genetische Distanz. Die neuronale und die synaptische genetische Distanz werden auf die gleiche Weise berechnet. Beide Distanzen resultieren im Wesentlichen aus zwei Termen, die durch die benutzerdefinierten Koeffizienten c_{sim} und c_{diff} gewichtet werden. Beide Terme iterieren mittels einer Doppelsumme über alle neuronalen bzw. synaptischen Gene der zu vergleichenden Genome. Die Intention des ersten Terms ist die Berechnung und Summation der Distanzen zwischen denjenigen Genen, die jeweils die gleiche ID besitzen. Innerhalb der Doppelsumme wird zu diesem Zweck eine neuronale bzw. synaptische Distanzfunktion $Dist_n(\cdot, \cdot)$ bzw. $Dist_s(\cdot, \cdot)$ aufgerufen. Die exakten Berechnungsvorschriften beider Distanzfunktionen sind für das weitere Verständnis nicht notwendig und werden deshalb im Folgenden nur kurz beschrieben. Zusammengefasst berechnen beide Funktionen nur dann eine Distanz ungleich null, wenn die verglichenen Gene die gleiche ID besitzen. In diesem Fall basiert die Distanz in $Dist_n(\cdot, \cdot)$ u. a. auf dem absoluten Betrag zwischen den Bias-Werten der verglichenen neuronalen Gene. Ferner wird die Distanz um jeweils eins inkrementiert, sofern sich bspw. die Aktivierungsfunktion oder Netzeingabefunktion unterscheiden. $Dist_s(\cdot, \cdot)$ kalkuliert im Wesentlichen den absoluten Betrag zwischen den synaptischen Gewichten der verglichenen Gene. Auch hier wird die Distanz für jede nichtnumerische Eigenschaft um jeweils eins inkrementiert, sofern diese bei beiden Genomen unterschiedlich ausgeprägt ist. Die Intention des zweiten, durch den Koeffizienten c_{diff} gewichteten Terms ist die Berechnung der Anzahl derjenigen Gene, die nicht die gleiche ID besitzen und somit grundsätzlich verschieden sind. Zu diesem Zweck gibt die Funktion $Diff(\cdot, \cdot)$ den Wert eins bzw. null zurück, wenn die verglichenen Gene unterschiedliche bzw. gleiche IDs besitzen. Die lineare Kombination beider Terme wird schließlich durch die maximale Anzahl der Neuronen bzw. Synapsen der verglichenen Genome dividiert. In jeder Generation berechnet NEAT für jedes Genom die Zugehörigkeit zu einer Spezies neu. Für jede Spezies bleibt nur das Genom mit der höchsten Fitness als Referenzvertreter der Spezies übrig. Für alle anderen Genome iteriert NEAT über die

vorhandenen Spezies und berechnet die genetische Distanz zwischen dem Genom und dem Vertreter jeder Spezies. Das Genom wird der Spezies zugeordnet, wenn der genetische Abstand eine vom Benutzer festgelegte Kompatibilitätsschwellenwert überschreitet. Kann ein Genom keiner bestehenden Spezies zugeordnet werden, erstellt NEAT eine neue Spezies mit dem Genom als Referenzvertreter.

4) Inkrementelle Komplexifizierung von Genomen

Andere neuroevolutionäre Algorithmen weisen eine geringe Recheneffizienz auf, da sie bereits in der Anfangspopulation verschiedene Topologien berücksichtigen müssen: Zum einen, weil sie keine dynamische Erweiterung der Vektorrepräsentation der Genome zur Laufzeit erlauben, zum anderen, weil jede Veränderung der KNN-Topologie zu einem Fitnessverlust und damit einhergehend zu einem sofortigen Ausschluss des Genoms von der Population führen kann. Hierdurch entstehen bereits während der Initialisierungsphase hochdimensionale Genome. In NEAT ist die genetische Enkodierung von Genomen zur Laufzeit erweiterbar (erste Eigenschaft) und die Population gemäß topologischer Ähnlichkeiten in Spezies untergliedert (zweite Eigenschaft). Aufgrund beider Eigenschaften müssen in NEAT keine verschiedenen KNN-Topologien während der Initialisierung der Population berücksichtigt werden. Die anfänglichen Genome enthalten keine versteckten Neuronen und unterscheiden sich lediglich hinsichtlich ihrer nichtstrukturellen Parameter, z. B. hinsichtlich ihrer synaptischen Gewichte, neuronalen Bias-Werte und neuronalen Aktivierungsfunktionen. NEAT erweitert die Topologien von KNN lediglich inkrementell, wobei diejenigen Genome beibehalten werden, die mit anderen Genomen derselben Spezies konkurrieren können. Die Anzahl von versteckten Neuronen, die durch genetische Operationen hinzugefügt werden, wird durch NEAT nicht begrenzt. Demzufolge ist der Algorithmus auf Problemen beliebiger Komplexität anwendbar.

Abbildung 3.10 veranschaulicht die Funktionsweise von NEAT als Programmablaufplan. Eine Erklärung der Programmablaufplan-Notation ist als Anhang F dem elektronischen Zusatzmaterial beigefügt. Analog der Berechnungsvorschriften für die genetische Distanz in den Formeln (3.21–23) orientiert sich das Ablaufschema ebenfalls an der Open-Source-Implementierung »NEAT-Python«. Im Folgenden soll der Programmablaufplan kurz erläutert werden. Während der Initialisierungsphase liest NEAT die benutzerdefinierten Hyperparameter aus einer Konfigurationsdatei ein. Darüber hinaus wird an den Algorithmus eine benutzerdefinierte Fitnessfunktion übergeben. Die Hyperparameter sind maßgeblich für die Steuerung von Kreuzungs- und Mutationsprozessen. Hingegen bewertet die Fitnessfunktion die Lösungsqualität von Genomen in Abhängigkeit getroffener Planungsentscheidungen. Sowohl die Art und Konfiguration

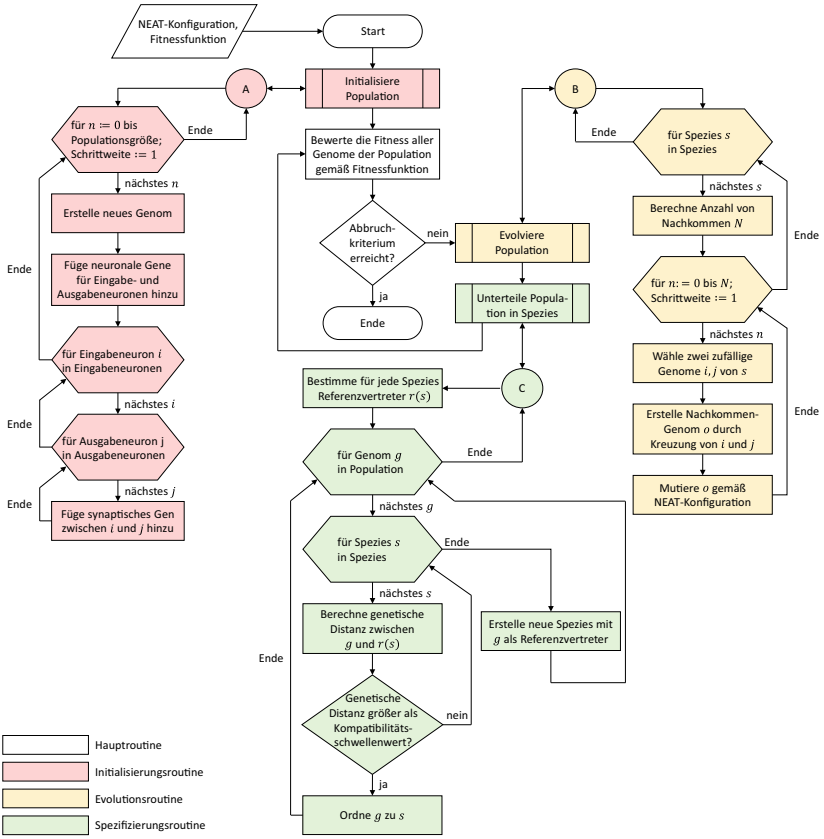


Abbildung 3.10 NEAT-Algorithmus als Programmablaufplan (in Anlehnung an Lang et al. 2021b)

der Hyperparameter als auch die Fitnessfunktion werden im Rahmen der zu entwickelnden Methode in Kapitel 5 ausführlich geschildert.

Bei der Initialisierung der Population erzeugt NEAT nur Genome mit Eingabe- und Ausgabeneuronen. Gemäß der Standardkonfiguration von NEAT-Python sind alle Eingabe- und Ausgabeneuronen über Synapsen miteinander verbunden. Darüber hinaus erlaubt NEAT-Python, dass zu Beginn nicht alle, sondern lediglich zufällige (oder gar keine) Eingabe- und Ausgabeneuronen miteinander verbunden sind. Zum Zweck der Übersichtlichkeit wird eine unvollständige Verbindung von

Eingabe- und Ausgabeneuronen im Programablaufplan von Abbildung 3.10 nicht berücksichtigt. Nach der Bewertung der Fitness jedes Genoms bricht der Algorithmus ab, sofern ein benutzerdefiniertes Abbruchkriterium erfüllt ist, z. B. das Überschreiten einer bestimmten Anzahl von Generationen oder eines Schwellenwerts bezogen auf das Fitnesskriterium. Andernfalls evolviert NEAT die Population, indem der Algorithmus zunächst mehrere zufällige Kreuzungen zwischen Genomen durchführt, gefolgt von mehreren zufälligen Mutationen in den resultierenden Nachkommen. Neben strukturellen Mutationen (siehe Abbildung 3.8), können an Genomen auch nichtstrukturelle Mutationen vollzogen werden, z. B. hinsichtlich ihrer synaptischen Gewichte, neuronalen Bias-Werte oder neuronalen Aktivierungsfunktionen. Die Mutationen von synaptischen Gewichten und neuronalen Bias-Werten unterliegen jeweils einer um den Wert null zentrierten Normalverteilung mit einer benutzerdefinierten Standardabweichung. Im Fall einer Mutation entnimmt NEAT der jeweiligen Normalverteilung eine Zufallsstichprobe und addiert diese auf den aktuellen Parameterwert. Mögliche neuronale Aktivierungsfunktionen werden als Liste in der Konfigurationsdatei hinterlegt. Im Zuge einer Mutation wird zufällig eine neue Aktivierungsfunktion aus der Liste gewählt. Bevor der Algorithmus in die nächste Generation übergeht, wird die Population in neue Spezies entsprechend des genetischen Abstands der Genome zu den jeweiligen Referenzvertretern aufgeteilt.

4 Stand der Wissenschaft und Technik: Bestärkendes Lernen in der Produktionsablaufplanung

Der Einsatz von RL für die Produktionsablaufplanung findet in der Wissenschaft noch vergleichsweise geringe Beachtung. Ungeachtet dessen erschienen in den vergangenen Jahrzehnten einige themenbezogene Veröffentlichungen, die im Folgenden diskutiert werden sollen. Anhand einer Literaturanalyse soll aufgezeigt werden, auf welche Art und Weise RL-trainierte Agenten für die Produktionsablaufplanung eingesetzt werden können. Die Literaturstudie dient somit im Wesentlichen der Beantwortung der ersten Forschungsfrage aus Abschnitt 1.2. Ferner sollen die Ergebnisse der Literaturanalyse in die Entwicklung der Methode (siehe Kapitel 5) einfließen, um diese so allgemeingültig wie möglich zu gestalten. Zuletzt sollen mithilfe der Literaturstudie Forschungs- und Entwicklungsbedarfe aufgezeigt werden, die von den bisherigen Veröffentlichungen nicht oder nur unzureichend adressiert werden. Entsprechend der Organisation des theoretischen Grundlagenkapitels zu RL werden die recherchierten Arbeiten in gradientenabhängige und gradientenfreie Verfahren für die Produktionsablaufplanung untergliedert.

4.1 Gradientenabhängige Verfahren für die Produktionsablaufplanung

Während den Arbeiten an der vorliegenden Dissertation wurden insgesamt 57 Publikationen analysiert, die den Einsatz von gradientenabhängigen RL-Verfahren für die Produktionsablaufplanung untersuchen. Wie in Abbildung 4.1 dargestellt, lassen sich die recherchierten Veröffentlichungen in fünf Kategorien untergliedern, welche unterschiedliche Einsatzarten von RL-Methoden für die

S. Lang, *Methoden des bestärkenden Lernens für die Produktionsablaufplanung*,
https://doi.org/10.1007/978-3-658-41751-2_4

Produktionsablaufplanung adressieren. Im Folgenden sollen die fünf Einsatzarten beschrieben und anhand einiger ausgewählter Publikationen veranschaulicht werden.

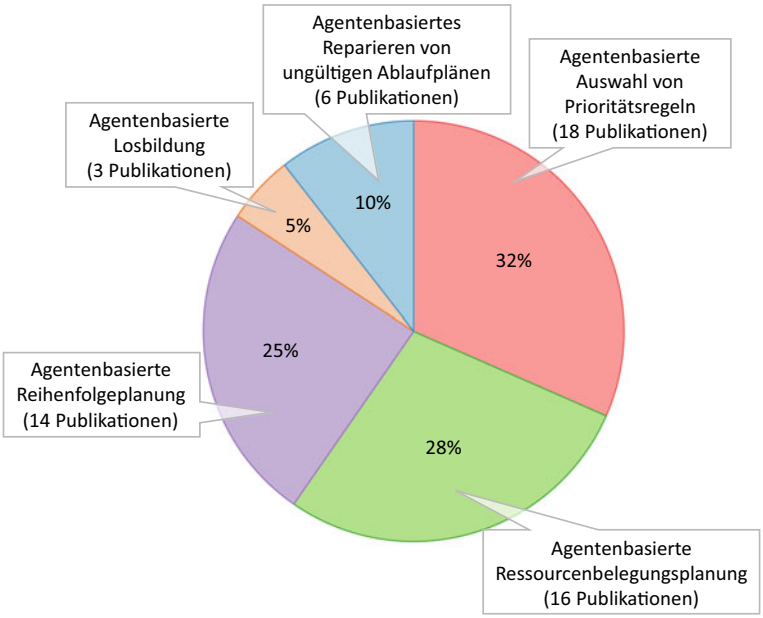


Abbildung 4.1 Kategorisierung der recherchierten Forschungsarbeiten, die gradientenabhängige Methoden des bestärkenden Lernens für die Produktionsablaufplanung untersuchen

4.1.1 Agentenbasierte Auswahl von Prioritätsregeln

Die Mehrheit der relevanten Forschungsarbeiten verwendet RL-Verfahren, um einen Agenten hinsichtlich der Auswahl von Prioritätsregeln in Abhängigkeit vom aktuellen Zustand des Produktionssystems zu trainieren. In Abschnitt 2.3.3.2 wurden bereits einige populäre Prioritätsregeln eingeführt. Das zugrundeliegende Entscheidungsproblem besitzt einen diskreten Aktionsraum, wobei jede Aktion mit der Auswahl einer Prioritätsregel assoziiert wird. Der Agent kann somit bspw. durch ein Klassifikator-KNN repräsentiert werden.

Die erste Veröffentlichung, die diesen Ansatz beschreibt, stammt von Rabelo et al. (1994). Die Autoren nutzen Q-Learning, um für ein stochastisches Job-Shop-Problem ein Ensemble von 15 KNN zu trainieren, wobei jedes KNN mit einer bestimmten Prioritätsregel assoziiert wird. Jedes KNN besitzt genau ein Ausgabeneuron, das die Wahrscheinlichkeit für die Auswahl der assoziierten Prioritätsregel ausgibt. Die Autoren entscheiden sich bewusst gegen die Verwendung eines KNN mit mehreren Ausgabeneuronen. Auf diese Weise vermeiden sie, dass bei der Aktualisierung eines Aktionsnutzens, aufgrund von gemeinsamen versteckten Neuronenschichten, die anderen Aktionsnutzen ebenfalls beeinflusst werden. Der Zustand, der in jedes KNN eingeht, umfasst Informationen zu Bearbeitungszeiten, Prozesszeiten, Rüstzeiten und Fertigstellungsfristen von Aufträgen in den entsprechenden Warteschlangen. Das Optimierungsziel der Autoren ist die Minimierung des Umlaufbestands, wobei Q-Learning die besten Ergebnisse erzielt, verglichen zur Anwendung von einzelnen Prioritätsregeln.

Aydin und Öztemel (2000) sowie Wang und Usher (2004, 2005) wenden ebenfalls Q-Learning für ein statisches Job-Shop- bzw. Ein-Maschinen-Problem an. Beide Arbeiten verzichten auf den Einsatz von KNN, um die Wahrscheinlichkeit für die Auswahl einer Prioritätsregel zu bestimmen. Stattdessen werden in der Aktionsnutzentabelle die verschiedenen Zustände der jeweiligen Produktionsumgebung direkt mit den auswählbaren Prioritätsregeln in Beziehung gesetzt. Beide Autoren berücksichtigen die Warteschlangenlänge bei der Bildung von Zuständen. Aydin und Öztemel berücksichtigen darüber hinaus die durchschnittliche Schlupfzeit der Aufträge in der Warteschlange, während Wang und Usher für jede Entscheidung eine Schätzung der Gesamtverspätung miteinbeziehen.

Shiue et al. (2018) analysieren den Einsatz von Q-Learning zur Auswahl von Prioritätsregeln in einem flexiblen Fertigungssystem, bestehend aus drei Ressourcen mit unterschiedlichen Kapazitäten, die durch drei fahrerlose Transportsysteme aus einem zentralen Puffer beliefert werden. Die Autoren berücksichtigen 30 Systemattribute, um den Zustand des Produktionssystems zu beschreiben, sowie fünf Prioritätsregeln als mögliche Aktionen. Der Agent wird durch eine Tabelle repräsentiert. Um die Dimensionalität der Tabelle zu reduzieren, verwenden die Autoren eine selbstorganisierende Karte (Kohonen 1990), die sich den Methoden des unüberwachten Lernens zuordnet. Konkret werden mithilfe der selbstorganisierenden Karte Systemattribute nach Ähnlichkeit gruppiert und zu aggregierten Zuständen konsolidiert. Die Autoren untersuchen drei Optimierungskriterien, nämlich die Maximierung des Systemdurchsatzes, die Minimierung der mittleren Durchlaufzeit und die Minimierung der Anzahl verspäteter Aufträge. Der Q-Learning-Ansatz der Autoren leistet für alle drei Optimierungskriterien die besten Ergebnisse. Dabei ist hervorzuheben, dass der Ansatz bessere Ergebnisse

erzielt als ein genetischer Algorithmus, der ebenfalls auf das Problem angewandt wurde. In einer nachfolgenden Veröffentlichung untersuchen die Autoren denselben Ansatz anhand einer weiteren Fallstudie aus der Halbleiterindustrie, unter weitgehender Berücksichtigung derselben Aktionen und Zustände (Shiue et al. 2020).

Ebenfalls wurden in jüngster Zeit DRL-Ansätze für die Auswahl von Prioritätsregeln beforscht. Sowohl Lin et al. (2019) als auch Luo (2020) untersuchen den DQN-Algorithmus zur Auswahl von Prioritätsregeln in einem Job-Shop-Problem. Lin et al. verfolgen als Optimierungsziel die Minimierung der Gesamtdauer des Ablaufplans. Sie berücksichtigen insgesamt zehn Attribute, um den Zustand des Produktionssystems zu definieren, u. a. die Anzahl der Ressourcen, die Anzahl der Aufträge, die Summe aller Prozesszeiten über alle Operationen und Aufträge, die maximale und minimale Prozesszeit über alle Operationen und Aufträge u. v. m. Hierbei kann der Agent zwischen sieben verschiedenen Prioritätsregeln wählen. In Experimenten auf mehreren Probleminstanzen erzielt DQN im Mittel bessere Ergebnisse als jede einzeln angewandte Prioritätsregel. Luo verfolgt die Minimierung der Gesamtverspätung über alle Aufträge als Optimierungskriterium. Der aktuelle Zustand des Produktionssystems setzt sich aus *(i)* dem Mittelwert und *(ii)* der Standardabweichung der Ressourcenauslastung, *(iii)* dem mittleren Ressourcendurchsatz, *(iv)* dem Mittelwert und *(v)* der Standardabweichung des Systemdurchsatzes sowie *(vi)* der prognostizierten und *(vii)* der wahren Anzahl verspäteter Aufträge zusammen. Die Aktionen des Agenten umfassen sechs selbstentwickelte Prioritätsregeln. In mehr als 50 Prozent der Experimente leistet der DQN-Ansatz von Luo bessere Ergebnisse als die alleinige Anwendung einer einzelnen Prioritätsregel.

Die diskutierten Beiträge bilden lediglich einen Teil der Arbeiten ab, die RL-trainierte Agenten zur Auswahl von Prioritätsregeln untersuchen. Sie wurden stellvertretend beschrieben, um das Funktionsprinzip zu verdeutlichen. Darüber hinaus existiert eine Vielzahl weiterer artverwandter Arbeiten, bei welchen der Agent zustandsabhängig zwischen verschiedenen Prioritätsregeln wechselt, z. B. (Heger und Voss 2020), welche neuronales Q-Learning für ein stochastisches Job-Shop-Problem zur Reduzierung der mittleren Verspätung untersuchen, (Wang et al. 2020), welche einen zweistufigen Q-Learning-Ansatz zur Minimierung von Stillstandzeiten und der Auslastungsimbalance von Ressourcen sowie zur Minimierung der Kosten für zu früh produzierte Aufträge für ein Job-Shop-Problem vorstellen, u. v. m. (Wei und Zhao 2005; Wei et al. 2009a, 2009b; Zhang et al. 2013; Bouazza et al. 2015; Yuan et al. 2016; Bouazza et al. 2017; Wang et al. 2017).

4.1.2 Agentenbasierte Ressourcenbelegungsplanung

Einige Arbeiten präsentieren Ansätze, in welchen ein RL-Agent Aufträge zur weiteren Bearbeitung auf stationäre Produktionsressourcen verteilt bzw. mobile Produktionsressourcen stationären Produktionsressourcen zuweist, um eine bestimmte Operation an einem Auftrag durchzuführen. Auch hier besitzt das zugrundeliegende Entscheidungsproblem einen diskreten Aktionsraum, wobei jede Aktion mit einer Ressource assoziiert wird.

Arviv et al. (2016) verwenden Q-Learning für das Training von zwei Agenten, die jeweils einen Transportroboter in einem Flow-Shop steuern. Jede Ressource des Flow-Shops besitzt einen Eingangs- und Ausgangspuffer. Die Ressourcen sind nicht fest miteinander verkettet, wodurch der vollständige Materialfluss über beide Transportroboter realisiert wird. Ungeachtet dessen müssen alle Produkte alle Ressourcen in derselben Reihenfolge durchlaufen. Immer dann, wenn ein Transportroboter verfügbar ist, entscheidet der jeweilige Agent, welcher nachfolgende Auftrag vom Ausgangspuffer einer Ressource i zum Eingangspuffer der Ressource $i + 1$ transportiert werden soll. Aufträge, die auf der letzten Ressource bearbeitet wurden, verlassen das System, ohne auf einen weiteren Transport angewiesen zu sein. Folglich umfasst der Aktionsraum für m Ressourcen $m - 1$ Aktionen. Für die Bestimmung der nächsten Aktion berücksichtigt jeder Agent den Zustand jedes Puffers (leer $= 0$ oder belegt $= 1$) und jeder Ressource (unproduktiv $= 0$, produktiv $= 1$). Für die Bestrafung von Aktionen werden die Zeitanteile, in denen die Roboter keine Aufträge transportieren, sowie die Wartezeiten von Aufträgen in Puffern gemessen. Das übergeordnete Optimierungskriterium ist die Minimierung der Gesamtdauer des Ablaufplans. Um ihre Ergebnisse vergleichen zu können, schätzen die Autoren eine untere Schranke für die optimale Lösung. Hierbei erreicht das Multiagentensystem eine Lösungsgüte von bis zu 99,8 Prozent der unteren Schranke.

Arinez et al. (2017) betrachten ein Produktionssystem mit zwei Fertigungszellen und einem zentralen Puffer, in welchem Produktionsaufträge durch einen Portalkran transportiert werden. Die Autoren verwenden Q-Learning, um eine agentenbasierte Steuerungsstrategie für den Portalkran zu optimieren. In jedem Zustand hat der Agent fünf Aktionen zur Auswahl, nämlich die erste oder zweite Fertigungszelle zu Be- oder Entladen oder alternativ keine Aktion auszuführen und im Ruhezustand zu verweilen. Für die Entscheidungsfindung berücksichtigt der Agent den Zustand der Fertigungszellen (produzierend, inaktiv, beladend, entladend, auf Portalkran wartend), die verbleibende Zeit der in den Fertigungszellen laufenden Operationen sowie den aktuellen Pufferbestand. Das Optimierungskriterium der Autoren ist die Maximierung der Produktivität. Diesbezüglich leistet

die agentenbasierte Steuerung bessere Ergebnisse als eine Steuerung nach dem »First-Come-First-Serve«-Prinzip (FIFO-Regel). Ou et al. (2018) verfolgen denselben Ansatz für ein sehr ähnliches Problem unter Berücksichtigung desselben Optimierungskriteriums. Auch hier ist ein Portalkran für den Transport von Aufträgen verantwortlich. Das betrachtete Produktionssystem umfasst jedoch vier Ressourcen und drei Puffer. Bezugnehmend auf dieselbe Problemart analysieren Ou et al. (2019) den Einfluss von verschiedenen Belohnungsfunktionen auf das Agententraining.

Qu et al. (2018) trainieren mithilfe des REINFORCE-Algorithmus ein Multiagentensystem hinsichtlich der Zuweisung von Aufträgen zu Produktionsressourcen in einem mehrstufigen Fertigungssystem. Hierbei unterliegen die Art, Systemeintrittszeit und Bearbeitungszeiten von Aufträgen stochastischen Schwankungen. Jeder Produktionsstufe ist ein Agent zugeordnet. In Abhängigkeit von Informationen der jeweils vorgelagerten Puffer (Eingangszustand) entscheidet jeder Agent für die jeweilige Produktionsstufe, welcher Auftrag auf welcher Ressource bearbeitet werden soll (Aktionen). Die zu optimierende Zielfunktion und die Belohnungsfunktion berücksichtigen u. a. die Höhe der Umlaufbestände sowie die fristgerechte Fertigstellung von Aufträgen. Der Ansatz der Autoren leistet im Mittel bessere Ergebnisse als die Anwendung von statischen Prioritätsregeln.

Stricker et al. (2018) trainieren mit dem Q-Learning-Algorithmus einen Agenten, der durch ein KNN repräsentiert wird. Die Aufgabe des Agenten ist die Verteilung von Auftragslosen auf Produktionsressourcen in einem Halbleiter-Produktionssystem. Das Produktionssystem beinhaltet acht Ressourcen, drei schienengeführte Transport-Shuttles und eine mobile Servicetechnik-Fachkraft, welche für die Behebung von zufällig auftretenden Störungen an den Ressourcen verantwortlich ist. Auftragslose werden auf genau einer Ressource bearbeitet und verlassen danach das System mithilfe eines Transport-Shuttles. Jedes Shuttle kann bis zu fünf Auftragslose transportieren. Zudem besitzt jede Ressource einen Eingangspuffer, in welchem einige wenige Lose zwischengelagert werden können. Der Agent berücksichtigt 32 Informationen zur Bildung von Eingangszuständen, u. a. die Anzahl bearbeiteter und unbearbeiteter Auftragslose auf jeder Maschine, die Zielorte der auf den Transport-Shuttles geladenen Auftragslose sowie die aktuelle Position der Servicetechnik-Fachkraft. Eine Besonderheit bei Stricker et al. ist, dass der RL-trainierte Agent selbst als mobile Ressource betrachtet wird und seinen Standort entweder zu einer Ressource oder zu einem Transport-Shuttle verlagern kann. In Abhängigkeit vom Standort ändern sich die Bezugsobjekte, die der Agent durch seine Aktionen beeinflusst. Der Agent kann entweder ein Auftragslos einer Produktionsressource zuweisen, ein bereits alloziertes Los wieder auf ein Transport-Shuttle laden, seine eigene Position zu einem anderen Standort

verlagern oder keine Aktion ausführen, um auf den nächsten Zustand zu warten. Die Autoren messen die Leistung des Agenten anhand der mittleren Systemauslastung und der mittleren Durchlaufzeit eines Auftrags. Der Ansatz der Autoren kann im Vergleich zu einer regelbasierten Heuristik die mittlere Systemauslastung um acht Prozent erhöhen und die mittlere Durchlaufzeit von 125 auf 118 Minuten reduzieren.

Darüber hinaus existieren einige weitere Arbeiten, die den Einsatz von RL-trainierten Agenten für die Ressourcenbelegungsplanung untersuchen (Martínez et al. 2011; Xue et al. 2018; Han et al. 2019; Kuhnle et al. 2019; Guo et al. 2020; Park et al. 2020; Zhou et al. 2020; Shi et al. 2020; Kuhnle et al. 2021; Park et al. 2021; Zhou et al. 2021).

4.1.3 Agentenbasierte Reihenfolgeplanung

Gemessen an der Anzahl der recherchierten Publikationen bildet die agentenbasierte Reihenfolgeplanung die dritte Haupteinsatzart von RL-Methoden für die Produktionsablaufplanung. Im Rahmen der Reihenfolgeplanung entscheidet der Agent, in welcher Sequenz die Aufträge in einer Warteschlange abgearbeitet werden. Wie die folgenden Arbeiten aufzeigen, kann sich die Art und Weise der Formulierung des jeweiligen Sequenzierungsproblems stark unterscheiden. Aus diesem Grund ist der vorliegende Abschnitt umfangreicher als die anderen Abschnitte zu den jeweiligen Einsatzarten. Bis auf die Arbeit von Gabel und Riedmiller (2012) haben jedoch alle diskutierten Arbeiten gemeinsam, dass die Aktionsräume der Agenten diskreter Natur sind. Im Kontext von DRL würde der Agent somit durch ein KNN mit mehreren Ausgabeneuronen repräsentiert werden.

Tanaka und Yoshida (1999) verwenden ein einfaches TD-Lernverfahren, um Aufträge in einem Flow-Shop mit zwei bzw. drei Ressourcen zu sequenzieren. Das Optimierungskriterium ist die Minimierung der Gesamtdauer des Ablaufplans. Die Auftragssequenz wird vor der ersten Ressource festgelegt und ist zwischen den Ressourcen nicht permutierbar. Der Agent wird durch ein KNN mit einer versteckten Schicht repräsentiert. Die Grundüberlegung der Autoren ist, dass mit Johnsons Algorithmus (1954) bereits eine exakte Heuristik existiert, die ein Flow-Shop-Problem mit zwei Ressourcen optimal löst. Die Zielstellung der Autoren ist, eine Agentenrepräsentation von Johnsons Algorithmus zu finden, die ebenfalls für Flow-Shops mit mehr als zwei Ressourcen nahezu optimale Lösungen erzielt. Vor diesem Hintergrund berücksichtigt der Agent für seine Entscheidungen diverse Informationen, die ebenfalls Johnsons Algorithmus in

den Lösungsprozess miteinbezieht, z. B. die Anzahl der Aufträge, bei denen die Prozesszeit auf der ersten Ressource größer als auf der zweiten Ressource ist. Die Autoren untersuchen drei verschiedene Kodierungsvarianten des Aktionsraums, wobei die erste und zweite Variante ebenfalls an Johnsons Algorithmus angelehnt sind, respektive an den Operationen, die der Algorithmus zur Lösungsbestimmung durchführt. Den Aktionen der ersten beiden Varianten liegen stets zwei Auftragslisten zugrunde. Die erste Auftragsliste umfasst alle Aufträge, bei denen die Prozesszeit auf der ersten Stufe kleiner gleich der Prozesszeit auf der zweiten Stufe ist, während die zweite Liste alle verbleibenden Aufträge umfasst. In der ersten Variante kann der Agent entweder aus der ersten Liste den Auftrag mit der minimalen Prozesszeit auf der ersten Stufe oder aus der zweiten Liste den Auftrag mit der maximalen Prozesszeit auf der zweiten Stufe wählen. Die zweite Variante ist eine Erweiterung der ersten Variante und umfasst acht mögliche Aktionen. Konkret kann der Agent aus der ersten bzw. zweiten Liste den Auftrag mit der minimalen bzw. maximalen Prozesszeit der ersten bzw. zweiten Stufe wählen. In der dritten Variante entspricht die Anzahl auswählbarer Aktionen der Anzahl einzuplanender Aufträge, sodass die Auswahl einer Aktion mit der Auswahl eines Auftrags assoziiert wird. Die Autoren stellen fest, dass der Agent optimale Auftragsreihenfolgen auf denjenigen Probleminstanzen findet, die ebenfalls mit Johnsons Algorithmus optimal gelöst werden können. Bei Probleminstanzen, die mit Johnsons Algorithmus nicht nachweisbar optimal gelöst werden können, war der Agent nicht in der Lage bessere Lösungen als Johnsons Algorithmus zu erzielen.

Hong und Prabhu (2001; 2004) untersuchen den Einsatz von Q-Learning, um einen Agenten hinsichtlich der Auswahl der nächsten zu produzierende Auftragsfamilie zu trainieren. Als Optimierungskriterium verfolgen die Autoren die Minimierung der mittleren Unpünktlichkeit, wobei sich verfrühte als auch verspätete Aufträge gleichermaßen negativ auf die Zielfunktion auswirken. Für die Entscheidung berücksichtigt der Agent *(i)* die Anzahl von Aufträgen, die sich von jeder Familie in der Warteschlange befinden, *(ii)* die zuletzt produzierte Auftragsfamilie sowie *(iii)* die Anzahl bisher produzierter Aufträge. Die Autoren bewerten die Aktionen des Agenten über die Summe der Wartezeiten über alle noch nicht bearbeiteten Aufträge. Der Ansatz der Autoren leistet in allen Probleminstanzen bessere Ergebnisse als die EDD- und LST-Regel.

Ramirez-Hernandez und Fernandez (2007) betrachten eine Linienfertigung mit zwei Arbeitsstationen und vier kapazitätsbegrenzten Puffern. Jeder Auftrag muss ein weiteres Mal auf der ersten Arbeitsstation bearbeitet werden, wenn dieser bereits die erste und zweite Arbeitsstation durchlaufen hat. Der erste Puffer beinhaltet Aufträge, die noch nicht freigegeben wurden. Der zweite und dritte Puffer

sind der ersten Arbeitsstation zugeordnet. Der zweite Puffer speichert neue freigegebene Aufträge, die das System noch nicht durchlaufen haben. Der dritte Puffer beinhaltet ausschließlich Aufträge, die zuvor von der zweiten Arbeitsstation bearbeitet wurden. Der vierte Puffer umfasst diejenigen Aufträge, die als nächstes an der zweiten Arbeitsstation bearbeitet werden müssen. Aufträge kommen kontinuierlich mit zufällig exponentialverteilten Zwischenankunftszeiten in das System. Das Puffern von Aufträgen ist mit Kosten verbunden, die pro Auftrag und Zeiteinheit anfallen. Die Zielstellung ist, die Lagerhaltungskosten für Umlaufbestände über die Zeit zu minimieren. Die Autoren untersuchen einen agentenbasierten Ansatz zur Steuerung der Auftragsfreigabe sowie der Auftragssequenzierung auf der ersten Stufe. Der Agent wird durch ein Actor-Critic-Modell repräsentiert, das mithilfe eines TD-Lernverfahrens trainiert wird. Der Agent berücksichtigt alle Pufferbestände als Eingangszustand. Sowohl die Auftragsfreigabe als auch -sequenzierung können über einen binären Aktionsraum abgebildet werden. Bei der Auftragsfreigabe entscheidet der Agent, ob der nächste Auftrag (FIFO-Prinzip) des ersten Puffers in den zweiten Puffer eingespeist werden soll. Immer dann, wenn die erste Arbeitsstation verfügbar wird, entscheidet der Agent im Zuge der Auftragssequenzierung, ob der nächste Auftrag aus dem zweiten oder dritten Puffer nach dem FIFO-Prinzip entnommen werden soll. Verglichen mit der globalen optimalen Entscheidungspolitik erzielt der Agent eine Lösungsgüte von über 98 Prozent.

Gabel und Riedmiller (2012) stellen ein Multiagentensystem vor, das für die Sequenzierung von Aufträgen in einem Job-Shop eingesetzt und mithilfe des REINFORCE-Algorithmus angelernt wird. Das Optimierungskriterium ist die Minimierung der Gesamtdauer des Ablaufplans. Jeder Ressource ist ein eigener Agent zugeordnet, der entscheidet, welche Operation von welchem Auftrag als nächstes bearbeitet wird. Der jeweilige Agent berücksichtigt als Zustand alle durchführbaren Operationen der Aufträge, die sich zum Zeitpunkt der Entscheidung in der Warteschlange der entsprechenden Ressource befinden. Diesbezüglich entspricht der Zustandsraum dem Aktionsraum. Zusätzlich kann jeder Agent von anderen Agenten eine Benachrichtigung erhalten, falls auf den umliegenden Ressourcen gerade ein Auftrag bearbeitet wird, der nachfolgend zu der Ressource des benachrichtigten Agenten gelangt. Auf diese Weise kann der Agent ebenfalls entscheiden, dessen Ressource im Leerlauf zu halten, um Kapazitäten für Aufträge mit hoher Priorität zu reservieren. Die Entscheidungspolitik eines Agenten wird durch eine Menge von trainierbaren Parametern repräsentiert, die der Menge der zu bearbeitenden Aufträge entspricht. Die Auswahlwahrscheinlichkeit eines Auftrags wird somit über exakt einen Parameter gesteuert. Sobald ein Auftrag auf

einer Ressource eine Operation beendet, wird dieser einer nachfolgenden Ressource zugeordnet. Im Zuge dessen wird die soeben fertiggestellte Operation aus dem lokalen Zustands- und Aktionsraum des Agenten der vorausgegangenen Ressource entfernt. Gleichzeitig wird der Zustands- und Aktionsraum des Agenten der nachfolgenden Ressource um die nächste auszuführende Operation des Auftrags erweitert. In deterministischen Probleminstanzen erreicht der Ansatz der Autoren eine Lösungsgüte von knapp 94 Prozent der optimalen Lösung. In stochastischen Probleminstanzen leistet das Multiagentensystem bessere Ergebnisse als diverse Prioritätsregeln (FIFO, LPT, SPT, etc.).

Unlängst wurden auch DRL-Verfahren für die Sequenzierung von Aufträgen beforscht. Zwei vielzitierte Arbeiten stammen von Waschneck et al. (2018a; 2018b). Die Autoren präsentieren ein Multiagentensystem für ein Job-Shop-Problem aus der Halbleiterindustrie. Jedem Bearbeitungszentrum ist ein Agent zugeordnet, der über die Bearbeitungsreihenfolge der ihm zugewiesenen Fertigungslose entscheidet. Ein Agent wird durch ein KNN mit einer versteckten Schicht repräsentiert. Die Autoren verwenden den DQN-Algorithmus, um den Agenten zu trainieren. Im Zustandsraum berücksichtigt jeder Agent die aktuelle Verfügbarkeit und Rüstung des jeweiligen Bearbeitungszentrums. Zudem erhält der Agent die Information, welche Fertigungsoperationen von Aufträgen im Bearbeitungszentrum ausgeführt werden können. Des Weiteren analysiert der Agent für seine Entscheidungen diverse Daten der wartenden Aufträge, z. B. die Produkttypen, die Fertigstellungsfristen u. v. m. Der Aktionsraum des Agenten entspricht der Anzahl der Pufferplätze des jeweiligen Bearbeitungszentrums. Darüber hinaus existiert eine zusätzliche Aktion, welche impliziert, dass im aktuellen Zustand kein neues Los bearbeitet werden soll. Das Training der Agenten erfolgt in drei Phasen. Zunächst wird ein Agent mittels überwachten Lernens vortrainiert. Die Trainingslabels stammen von einem regelbasierten Expertensystem, welches das Unternehmen hinter der Problemstellung derzeit für die Steuerung der Produktion verwendet. In der zweiten Phase wird derselbe Agent für lediglich ein Bearbeitungszentrum mithilfe des DQN-Algorithmus trainiert, während die Auftragssequenzierung für die anderen Bearbeitungszentren durch eine einfache Heuristik erfolgt. In der dritten Phase werden alle Bearbeitungszentren durch Instanzen des vortrainierten DQN-Agenten gesteuert, wobei jeder Agent das Training separat fortführt. Die Vergabe von Belohnungen basiert maßgeblich auf der aktuellen Auslastung der Bearbeitungszentren. Ungültige Aktionen, z. B. die Auswahl eines nicht belegten Pufferplatzes für die Produktion, sowie der Abbruch der Simulation durch Systemüberlast werden mit einer Bestrafung von (-1) bzw. (-2) quittiert. Der DQN-Ansatz erzielt im Mittel die gleiche Lösungsgüte wie das bisher verwendete regelbasierte Expertensystem.

Weitere Arbeiten, die RL-trainierte Agenten für die Sequenzierung von Aufträgen und Operationen einsetzen, stammen u. a. von Tuncel et al. (2014), Fonseca-Reyna et al. (2015), Qu et al. (2016b), Méndez-Hernández et al. (2019), Tan et al. (2019), Zheng et al. (2019), Altenmüller et al. (2020) und Lee et al. (2020).

4.1.4 Agentenbasierte Losbildung

Drei der recherchierten Arbeiten betrachten die Produktionsablaufplanung insbesondere als Losgrößenproblem. Grundvoraussetzung ist, dass die zu produzierenden Aufträge Rüstfamilien angehören. Sofern keine auftragsbezogenen Fertigstellungsfristen betrachtet werden, kann das Optimierungsproblem auf die Entscheidung reduziert werden, zu welchem Zeitpunkt eine Ressource auf welche Produktfamilie umgerüstet wird, bspw. um die Produktionskosten oder die Zykluszeit des Produktionsprogramms zu minimieren. Der Agent handelt in einem diskreten Aktionsraum, wobei jede Aktion mit einer Produktfamilie assoziiert wird, auf welche die jeweilige Ressource gerüstet werden kann.

Paternina-Arboleda und Das (2005) betrachten die Ablaufplanung in einer Ein-Maschinen-Umgebung mit unterschiedlichen Produkttypen als stochastisches Losgrößenproblem. Mithilfe eines selbstentwickelten RL-Verfahrens trainieren sie einen KNN-Agenten, zustandsabhängig zu entscheiden, ob und auf welchen Produkttypen die Ressource gerüstet werden soll. Der Agent berücksichtigt für seine Entscheidungen den aktuellen Lagerbestand, den Produktionsrückstand der verschiedenen Produkttypen sowie die aktuelle Rüstung der Maschine. Das Optimierungskriterium ist die Minimierung der Gesamtkosten, die sich aus Lagerhaltungskosten, Umrüstungskosten sowie Vertragsstrafen für Produktionsrückstände zusammensetzt. Die Autoren vergleichen ihre Ergebnisse mit einer deterministischen Rüstungspolitik, die zyklisch nach demselben Muster die unterschiedlichen Produkttypen rüstet. Der RL-trainierte Agent kann in allen Experimenten zwischen einem bis sieben Prozent bessere Ergebnisse als die deterministische Entscheidungspolitik erzielen.

Qu et al. (2016a) verfolgen einen ähnlichen Ansatz, betrachten jedoch ein mehrstufiges Flow-Shop-Problem und berücksichtigen zusätzlich Fertigungspersonal, welches für die Bearbeitung der Produkte auf den Ressourcen erforderlich ist. Die Autoren verfolgen zwei Optimierungsziele: Zum einen die Maximierung der Produktivität des Fertigungspersonals, zum anderen die Minimierung der Personalkosten. Um beide Optimierungskriterien zu adressieren, lernen die Autoren mittels Q-Learning einen Agenten für die Personalplanung und einen Agenten für

die Ablaufplanung in derselben Produktionsumgebung an. Der erste Agent entscheidet für jede Arbeitskraft welche Operation an welcher Ressource ausgeführt werden soll. Der Agent für die Ablaufplanung entscheidet für jede Maschine, ob sie auf einen bestimmten Produkttyp rüstet und diesen darauffolgend produziert oder ob sie inaktiv ist. Für die Entscheidungsfindung beobachten beide Agenten die Länge der Warteschlangen sowie verschiedene Informationen der Arbeitsstationen und Fertigungsarbeitskräfte. Die Autoren weisen nach, dass beide Agenten sich über die Zeit verbessern und zu einer Entscheidungspolitik konvergieren. Bereits ein Jahr zuvor haben die Autoren ein ähnliches Problem betrachtet (Qu et al. 2015), jedoch ohne die Berücksichtigung von Fertigungspersonalressourcen, sodass sich der Lösungsansatz auf das Ablaufplanungsproblem reduziert.

4.1.5 Agentenbasiertes Reparieren von ungültigen Ablaufplänen

Durch unvorhersehbare Ereignisse können in einem Fabriksystem Störungen auftreten, wodurch ein vorbestimmter Produktionsablaufplan nicht realisiert werden kann. Vor diesem Hintergrund präsentierten einige Arbeiten Ansätze, bei welchen ein RL-trainierter Agent schrittweise einen ungültigen Ablaufplan zu einem neuen gültigen Ablaufplan umbaut, der ein gegebenes Optimierungskriterium möglichst gut erfüllt. Hierbei wird in allen recherchierten Arbeiten für den jeweiligen Agenten ein diskreter Aktionsraum definiert, wobei sich die Art und Wirkung der Agentenaktionen teilweise kaum und teilweise deutlich zwischen den unterschiedlichen Arbeiten unterscheiden.

Die erste Arbeit, welche diesen Ansatz vorstellt, stammt wahrscheinlich von Zhang und Dietterich (1995) für ein Job-Shop-Problem mit der Zielstellung die Gesamtdauer des Ablaufplans zu minimieren. Das Job-Shop-Problem bildet die Spaceshuttle-Abfertigung der NASA ab. Der Prozess involviert verschiedene Abfertigungsstationen und Fachkräfteteams (Ressourcen-Pools), welche für die Instandsetzung der Spaceshuttles erforderlich sind. Der RL-Agent wird durch ein KNN mit 40 versteckten Neuronen repräsentiert. Als Eingangszustand berücksichtigen die Autoren diverse Attribute, welche den initialen Ablaufplan charakterisieren, etwa den Mittelwert und die Standardabweichung der freien Kapazitäten von denjenigen Fachkräfteteams, die als Engpass bekannt sind, den Mittelwert und die Standardabweichung der Schlupfzeiten von Operationen, die an den Spaceshuttles auszuführen sind, die Anzahl überlasteter Ressourcen u. v. m. Der Agent kann entweder eine Operation einem anderen Ressourcen-Pool zuordnen oder sie zu einem früheren oder späteren Zeitpunkt neu terminieren. Um

die Aktionen des Agenten zu bewerten, wird in jedem Zeitschritt überprüft, ob sich die Überlastung von Ressourcen im Vergleich zum vergangenen Zeitschritt verbessert hat. Der Agent wird durch ein einfaches TD-Lernverfahren trainiert. Die Autoren vergleichen ihren Ansatz mit einer früheren Arbeit, in welcher die Metaheuristik »Simulated Annealing« für das Problem untersucht wurde. Hierbei konnte der RL-Ansatz in allen Probleminstanzen bessere Ergebnisse erzielen.

Einen ähnlichen Ansatz verfolgen Palombarini und Martínez (2012). Die Autoren stellen ein RL-basiertes Werkzeug vor, mit dessen Hilfe ungültige Ablaufpläne repariert werden können. Die Autoren testen ihr Werkzeug an einem Job-Shop-Problem, in welchem Verspätungen von Aufträgen möglichst vermieden werden sollen. Der Agent wird durch eine Tabelle repräsentiert und mittels Q-Learning trainiert. Eine Besonderheit ist, dass Zustände und Aktionen durch Konstrukte der Prädikatenlogik zweiter Stufe beschrieben werden. Jeder Zustand, den der Agent für seine Entscheidungen analysiert, besteht aus einem vollständigen Ablaufplan sowie mindestens einer verletzten Restriktion. Der Ablaufplan umfasst für jede Produktionsressource eine Liste, in welcher die an den zugeordneten Aufträgen durchzuführenden Operationen chronologisch geordnet sind. Eine verletzte Restriktion wird durch eine Operation eines Auftrags und dessen Ressourcenzuordnung beschrieben. Der Agent kann Operationen von Aufträgen auf derselben Ressource um eine Position verschieben, sie an den Anfang oder an das Ende einer Bearbeitungssequenz setzen oder die Positionen von zwei Auftragsoperationen untereinander vertauschen. Darüber hinaus kann der Agent Auftragsoperationen ebenfalls anderen Ressourcen zuweisen, indem er sie an den Anfang oder an das Ende einer Bearbeitungssequenz setzt oder mit einer anderen Operation vertauscht. In der Belohnungsfunktion wird zum einen berücksichtigt, dass eine Verringerung der Gesamtverspätung über alle Aufträge belohnt wird, zum anderen, dass eine hohe Anzahl von Änderungen im Ablaufplan bestraft wird.

Zhao et al. (2018) verfolgen einen prioritätsregel- und reallokationsbasierten Ansatz, um ungültige Ablaufpläne in einem flexiblen Job-Shop mit zufälligen Ressourcenausfällen zu reparieren. Das Job-Shop-Problem wird durch acht Ressourcen, fünfzehn Aufträge und zehn Operationen definiert. Der initiale Ablaufplan wird mithilfe eines genetischen Algorithmus und eines Modells des Job-Shop-Problems generiert, das Ressourcenausfälle vernachlässigt. Danach wird der Ablaufplan im ursprünglichen Job-Shop-Modell mit Ressourcenausfällen simuliert. Immer dann, wenn eine Ressource ausfällt, entscheidet ein Agent, der mit Q-Learning trainiert wird, mit welcher Prioritätsregel (SPT, EDD oder FIFO) der nächste Auftrag aus der Warteschlange der ausgefallenen Ressource

ausgewählt werden soll. Der Auftrag wird dann von derjenigen Ressource bearbeitet, die zum frühestmöglichen Zeitpunkt verfügbar ist. Um eine Entscheidung zu treffen, analysiert der Agent die erwartete Dauer des Ressourcenausfalls im Verhältnis zur kumulierten Bearbeitungszeit der Aufträge im vorgelagerten Puffer. Der Ansatz der Autoren leistet bessere Ergebnisse als die alleinige Anwendung der durch den Agenten auswählbaren Prioritätsregeln.

Weitere Ansätze, die mithilfe von RL eine agentenbasierte Reparatur von Ablaufplänen umsetzen, können u. a. in (Palombarini und Martinez 2018; Palombarini und Martínez 2019; Minguillon und Lanza 2019) nachgelesen werden.

4.2 Gradientenfreie Verfahren für die Ablaufplanung im Allgemeinen

Verglichen zu den gradientenabhängigen Verfahren sind gradientenfreie RL-Ansätze für die Produktionsablaufplanung in der wissenschaftlichen Literatur nur spärlich vertreten. Obgleich die Literaturrecherche für gradientenfreie RL-Verfahren nicht auf die Produktionsablaufplanung beschränkt wurde, konnten insgesamt nur 14 relevante Publikationen identifiziert werden. Im Gegensatz zu Abschnitt 4.2 werden die recherchierten Arbeiten nicht nach ihrer Einsatzart, sondern gemäß der RL-Methoden kategorisiert, welche für die jeweiligen Ablaufplanungsprobleme zum Einsatz kommen. Eine entsprechende Kategorisierung der relevanten Arbeiten ist in Abbildung 4.2 dargestellt.

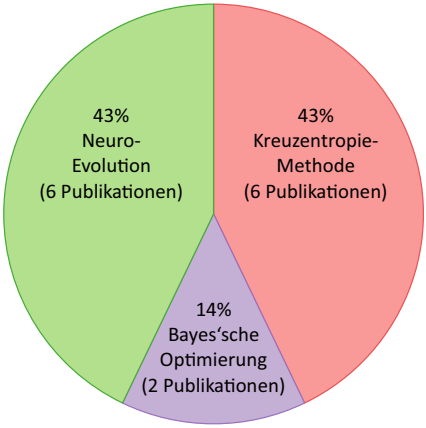


Abbildung 4.2 Kategorisierung der recherchierten Forschungsarbeiten, die gradientenfreie Methoden des bestärkenden Lernens für die Ablaufplanung im Allgemeinen untersuchen

Die Tatsache, dass gradientenfreie Verfahren ebenfalls in der RL-Grundlagenliteratur nicht gleichermaßen Erwähnung, könnte womöglich ein Grund für die geringe Anzahl relevanter Publikationen sein. Im Folgenden sollen die Rechercheergebnisse im Detail diskutiert werden.

4.2.1 Einsatz der Kreuzentropie-Methode in der Ablaufplanung

Derzeit existieren nur wenige Arbeiten, welche die Kreuzentropie-Methode zur Lösung von Ablaufplanungsproblemen untersucht haben. In den meisten relevanten Publikationen wird die Kreuzentropie-Methode verwendet, um vollständige Lösungen für das jeweilige Ablaufplanungsproblem zu generieren. In manchen Publikationen wird die Methode eingesetzt, um bestimmte Parameter für ein i. d. R. stochastisches Ablaufplanungsproblem zu konkretisieren. Die ersten themenbezogenen Veröffentlichungen stammen von Bendavid und Golany (2008, 2009), die ein Projektablaufplanungsproblem mit ressourcengebundene Projektaktivitäten und stochastischen Aktivitätszeiten betrachten. Aufgrund der teilweise zufälligen Zeitspannen von Projektaktivitäten können zwei Probleme auftreten. Zum einen kann eine Aktivität früher fertig sein als geplant und die Ressource für die Folgeaktivität noch nicht verfügbar sein, sodass die Weiterbearbeitung des Projekts unnötig verzögert wird. Zum anderen kann die umgekehrte Situation eintreten, dass die Bearbeitung einer Projektaktivität länger andauert als geplant. In diesem Fall verweilen bereits reservierte Ressourcen in einem unproduktiven Zustand. Darüber hinaus wird der Start von anderen Projektaktivitäten verzögert, welche auf die Ressource der verspäteten Aktivität angewiesen sind. Die Autoren schlagen vor, mithilfe der Kreuzentropie-Methode untere Schranken für die Startzeiten der Projektaktivitäten zu bestimmen, um die Allokation von Ressourcen effizienter zu gestalten. Zu diesem Zweck formulieren die Autoren ein Optimierungsproblem, in welchem Projekte, die auf eine Ressource zur weiteren Bearbeitung warten, mit Umlaufkosten bewertet werden, während Projektaktivitäten, die verspätet starten, Strafkosten erhalten. Über mehrere Iterationen optimiert die Kreuzentropie-Methode die Parameter einer multivariaten Normalverteilung, mit der zufällige untere Schranken für die Startzeiten aller Projektaktivitäten generiert werden. Hierbei generiert die Kreuzentropie-Methode kostengünstigere untere Schranken als die Auswahl der frühest bzw. spätest möglichen Startzeitpunkte als untere Schranken. Ferner erzielt die Kreuzentropie-Methode bessere Ergebnisse als die Metaheuristik »Simulated Annealing«.

Wang et al. (2015) betrachten die Produktionsablaufplanung einer Gießerei. In der ersten Produktionsstufe wird das Metall zu Chargen verflüssigt. Darauffolgend durchläuft jede Charge in Abhängigkeit von technologischen Anforderungen eine bestimmte Anzahl von Veredelungsprozessen (z. B. Entgasung, Desoxidation, Entschwefelung, Entfernung von Einschlüssen u. v. m.) an unterschiedlichen Arbeitsstationen. Im letzten Produktionsschritt wird die Charge in einer Stranggussanlage in ihre finale Form gegossen. Die Zielstellung ist die Ermittlung eines Ablaufplans, der den gesamten Energieverbrauch der Gießerei minimiert. Der Energieverbrauch wird insbesondere durch ungeplante Wartezeiten der Chargen beeinflusst. Je höher die Wartezeiten ausfallen, desto mehr Energie muss investiert werden, um das Metall nicht vorzeitig abkühlen zu lassen. Somit korreliert der Energieverbrauch mit den Wartezeiten, die durch eine geeignete Ablaufplanung minimiert werden können. Die Zuordnung von Chargen zu Produktionsstufen ist aufgrund der technologischen Reihenfolge gegeben. Die Bearbeitungsreihenfolge der Aufträge auf jeder Ressource wird mithilfe einer Rückwärtsterminierung ermittelt, da jede Charge eine obligatorische Startzeit auf der Stranggussanlage in der letzten Produktionsstufe besitzt. Einige Produktionsstufen besitzen jedoch mehrere Ressourcen, woraus eine Menge lokaler Allokationsprobleme resultiert. An dieser Stelle verwenden die Autoren die Kreuzentropie-Methode, um eine Wahrscheinlichkeitsverteilung für jede Produktionsstufe, die mehrere Bearbeitungskapazitäten besitzt, zu optimieren. Die Verteilung gibt an, mit welcher Wahrscheinlichkeit ein Auftrag einer bestimmten Bearbeitungskapazität zugeordnet wird. Die Arbeit der Autoren lässt sich somit der Einsatzart »Agentenbasierte Ressourcenbelegungsplanung« zuordnen (siehe Abschnitt 4.1.2). Ferner präsentieren die Autoren einige problembezogene Verbesserungen der Kreuzentropie-Methode. Zum Beispiel werden in jeder Iteration eine bestimmte Anzahl von Lösungen nicht mit der zu optimierenden Wahrscheinlichkeitsverteilung, sondern stattdessen mithilfe von heuristischen Planungsregeln generiert. Die Autoren vergleichen ihren Ansatz mit einem genetischen Algorithmus. Sowohl die klassische als auch verbesserte Kreuzentropie-Methode erzielen bessere Ergebnisse als der genetische Algorithmus. Darüber hinaus generiert die verbesserte Kreuzentropie-Methode energieeffizientere Ablaufpläne als die klassische Kreuzentropie-Methode.

Lv und Liu (2016) verwenden die Kreuzentropie-Methode um eine kombinierte Prozess- und Ablaufplanung für ein Job-Shop-Problem zu realisieren. Im Rahmen der Prozessplanung wird für jeden Auftrag eine Bearbeitungssequenz aus einer Menge möglicher Bearbeitungssequenzen ausgewählt. Im Zuge der Ablaufplanung wird eine Bearbeitungsreihenfolge über alle Operationen und Aufträge

ermittelt. Die Arbeit kann somit der Einsatzart »Agentenbasierte Reihenfolgeplanung« zugerechnet werden (siehe Abschnitt 4.1.3). Ein Lösungskandidat wird durch zwei Vektoren repräsentiert. Der erste Vektor bildet auf die Menge einzuplanender Aufträge ab. Jedes Vektorelement enthält einen ganzzahligen Wert, der den Index der gewählten Bearbeitungssequenz des jeweiligen Auftrags repräsentiert. Der zweite Vektor bildet auf die Menge einzuplanender Operationen über alle Aufträge ab. Hierbei repräsentiert jedes Vektorelement eine Position in der Bearbeitungssequenz. Um die Länge des Vektors zu bestimmen, wird die Summe über die maximale Anzahl an Operationen aus allen verfügbaren Bearbeitungssequenzen je Auftrag gebildet. Für jeden Auftrag werden die Längen aller verfügbaren Bearbeitungssequenzen vereinheitlicht. Das bedeutet, dass Sequenzen, die weniger Operationen erfordern als die längste Bearbeitungssequenz des jeweiligen Auftrags, um eine entsprechende Anzahl von Pseudooperationen ohne Bearbeitungszeiten erweitert werden. Ferner wird angenommen, dass die Operationen auftragsübergreifend aufsteigend indiziert sind, jedoch Operationen von unterschiedlichen Bearbeitungssequenzen desselben Auftrags denselben Index besitzen, sofern sie an der gleichen Position der jeweiligen Auftragssequenz berücksichtigt werden. Auf diese Weise kann nun mittels Zufallsstichprobe jedem Vektorelement eine Operation zugeordnet werden. In vier Experimenten und anhand verschiedener Job-Shop-Probleme vergleichen die Autoren ihre Methode mit verschiedenen Lösungsansätzen, (genetischer Algorithmus u. v. m.). Allen Experimenten wird die Minimierung des Ablaufplans als Optimierungskriterium zugrunde gelegt. In den meisten Experimenten liefert die Kreuzentropie-Methode die besten Ergebnisse.

Ferner existieren wenige weitere Arbeiten, welche die Kreuzentropie-Methode für Ablaufplanungsprobleme untersuchen, z. B. (Chen et al. 2016) für die Signalverarbeitung in Telekommunikationsnetzwerken oder (Liu et al. 2018) für die Aufgabenverwaltung in hybriden Cloud- und Edge-Computing-Netzwerken. Beide Arbeiten können der Einsatzart »Agentenbasierte Ressourcenbelegungsplanung« zugeordnet werden (siehe Abschnitt 4.1.2).

4.2.2 Einsatz von Bayes'scher Optimierung in der Ablaufplanung

Viele wissenschaftliche Beiträge, die im Titel oder Abstrakt Bayes'sche Optimierung referenzieren, beziehen sich auf ein anderes Verfahren namens Bayes'scher Optimierungsalgorithmus (BOA), z. B. (Li und Aickelin 2003; Yang et al. 2011; Sun et al. 2015; Biswas et al. 2016; Muhuri und Biswas 2020). BOA ist ein

genetischer Algorithmus, der iterativ für eine gegebene Probleminstanz eine Population von Bayes'schen Netzen parametrisiert, welche jeweils eine Lösung des Problems repräsentieren (Pelikan et al. 1999). Dem Ansatz liegt die Idee zugrunde, dass jedes kombinatorische Optimierungsproblem als Graph dargestellt werden kann, wobei eine Lösung durch einen Pfad im Graph repräsentiert wird. Ein Bayes'sches Netz ist ein Graph, dessen Knoten Zufallsvariablen und dessen Kanten Abhängigkeiten zwischen den Variablen repräsentieren. BOA sucht für ein gegebenes Optimierungsproblem ein Bayes'sches Netz, das eine möglichst optimale Lösung des Problems über eine gemeinsame Wahrscheinlichkeitsverteilung darstellt. In der initialen Population basiert die Struktur jedes Bayes'schen Netzes auf dem spezifischen Problemgraphen der betrachteten Probleminstanz. Über mehrere Iterationen aktualisiert BOA die Wahrscheinlichkeiten in den Knoten und fügt womöglich neue Relationen zwischen Knoten hinzu. Für viele Probleme der Produktionsablaufplanung (z. B. Job-Shop-Probleme) ist es i. d. R. nicht möglich ein für eine Probleminstanz optimiertes Bayes'sches Netz zur Lösung von anderen Probleminstanzen zu verwenden, weil sich bspw. die Vorrangbedingungen und die Anzahl der Ressourcen für die Bearbeitung von Operationen zwischen den Probleminstanzen unterscheiden. Aus diesem Grund werden diejenigen Arbeiten, welche BOA für die Produktionsablaufplanung untersuchen, im Folgenden nicht berücksichtigt. Ferner existieren einige Arbeiten, welche die Bayes'sche Optimierung für die Feinjustierung von Hyperparametern eines anderen Optimierungsverfahrens anwenden, z. B. (Roman et al. 2016; Kim et al. 2020). Diese Arbeiten finden ebenfalls keine Berücksichtigung, weil die Bayes'sche Optimierung nicht als primäres Lösungsverfahren zum Einsatz kommt. Zusammengefasst verbleiben lediglich zwei Arbeiten, die im Folgenden diskutiert werden sollen.

Dolatnia et al. (2016) betrachten Probleme, in welchen die Durchführung von realen Experimenten äußerst zeit- und kostspielig ist. Als Beispiel nennen die Autoren u. a. die Optimierung der Ausgangsleistung von mikrobiellen Brennstoffzellen, die mithilfe von Bakterien Energie erzeugen. Die Menge der erzeugten Energie ist von vielen Parametern abhängig, u. a. der Art und Kombination verschiedener Bakterien, den Oberflächeneigenschaften der Anode u. v. m. Die Durchführung von realen Experimenten ist zeitaufwändig, weil jedes Experiment im Voraus die Züchtung eines Bakterienstamms erfordert. Vor diesem Hintergrund formulieren die Autoren ein Optimierungsproblem, das den Experimentvorbereitungs- und -durchführungsprozess mit dessen Ressourcen und Aktivitäten modelliert. Konkret beinhaltet das Problem eine Menge von Präparationsstationen, um die für das Experiment erforderlichen Utensilien vorzubereiten, sowie eine Menge von Laboratorien, in denen die Experimente durchgeführt

werden. Darüber hinaus erfordert die Durchführung von Experimenten speziell qualifiziertes Personal. Die Vorbereitungs- und Durchführungszeiten sowie das erforderliche Personal sind für jedes Experiment spezifisch. Somit handelt es sich im Kern um ein Hybrid-Flow-Shop-Problem mit auftragsspezifischen Prozesszeiten und zusätzlichen Ressourcenbedingungen auf der zweiten Stufe. Für die Lösung des Problems schlagen die Autoren einen echtzeitfähigen Planungsalgorithmus vor, der auf einer Baumsuche basiert. Da eine vollständige Enumeration des Entscheidungsbaums zeitlich nicht möglich ist, trainieren die Autoren mithilfe von Bayes'scher Optimierung einen Agenten, der in jedem Baumknoten entscheidet, welches Experiment als nächstes durchgeführt werden soll. Vor diesem Hintergrund ordnet sich die Arbeit der Einsatzart »Agentenbasierte Reihenfolgeplanung« zu. Der Agent berücksichtigt für seine Entscheidungen u. a. die Menge bereits komplettierter Experimente, die freien Kapazitäten jeder Ressource sowie die verbleibende Zeit der aktuell laufenden Präparations- und Experimentoperationen. Für die Konstruktion des Entscheidungsbaums berücksichtigt der Agent vergangene Beobachtungen und schätzt für jede konstruierbare Verzweigung die beste zu erwartende Verbesserung. Die Autoren vergleichen ihren Ansatz mit mehreren Prioritätsregeln. Das Optimierungskriterium ist die Maximierung der Stickstoffproduktion der Bakterienstämme, die aufgrund unterschiedlicher Experimentkonfigurationen variiert. Zusammenfassend erzielt der Bayes'sche Optimierungsansatz signifikant bessere Ergebnisse als die Verwendung von Prioritätsregeln.

Candelieri et al. (2018) verwenden Bayes'sche Optimierung, um die Kosten eines Wasserverteilungssystems zu optimieren. Das zugrundeliegende Optimierungsproblem ist ein MILP, in welchem zwei Varianten von Pumpen berücksichtigt werden. Für Pumpen der ersten Variante muss zu jedem Zeitpunkt entschieden werden, ob sie in Betrieb genommen werden sollen. Jede Pumpe der ersten Variante kann somit über eine binäre Entscheidungsvariable abgebildet werden. Bei der zweiten Pumpenvariante kann die Durchflussrate flexibel gesteuert werden. Pumpen der zweiten Variante werden somit über kontinuierliche Entscheidungsvariablen abgebildet. Die Optimierung der Betriebskosten erfolgt unter mehreren Nebenbedingungen, u. a. dass die Wassernachfrage der Verbraucher befriedigt, dass zu jedem Zeitpunkt der Wasserdruck von jeder Pumpe in einem bestimmten Wertebereich liegt oder dass die Füllmengen der Wasserreservoirs am Ende des Tages den kritischen Bestand nicht unterschreiten. Eine Besonderheit der Arbeit ist, dass der Agent nicht durch einen Gauß-Prozess repräsentiert wird, sondern durch eine Menge von nicht korrelierten Entscheidungsbäumen, die über einen Random-Forest-Algorithmus generiert werden. Die Autoren motivieren diese Entscheidung damit, dass der Gauß-Prozess schlecht

mit zunehmender Anzahl von Entscheidungsvariablen skaliert. Die Autoren testen ihren Algorithmus anhand einer realen Fallstudie, die das Wasserverteilungssystem von Mailand darstellt. Sie vergleichen ihre Ergebnisse mit der cloudbasierten Optimierungssoftware »SigOpt« sowie mit den Softwarepaketen »ALAMO« (zur Generierung eines Metamodells des Optimierungsproblems zur schnelleren Evaluation von Lösungskandidaten) und »BARON« (für die eigentliche Optimierung). Hierbei gelingt es den Autoren, in allen Experimenten bessere Ergebnisse zu erzielen als die kommerziellen Softwarepakete. Hinsichtlich der Rechenzeit sei jedoch angemerkt, dass die Bayes'sche Optimierung mindestens zwei Stunden für die Lösung einer Probleminstanz benötigt. Die Arbeit gibt keinen Aufschluss darüber, ob das angelernte Modell auf andere Probleminstanzen angewandt werden kann.

4.2.3 Einsatz von Neuro-Evolution in der Ablaufplanung

Obgleich bereits Anfang der 1990er Jahre erste Neuro-Evolution-Algorithmen vorgestellt wurden, existieren bislang nur sehr wenige Arbeiten, welche diese Verfahren für Ablaufplanungsprobleme untersucht haben. Da im Rahmen dieser Arbeit ein neuroevolutionärer Algorithmus stellvertretend für alle gradientenfreien Ansätze für die Produktionsablaufplanung untersucht wird, werden die folgenden recherchierten Arbeiten gleichermaßen ausführlich behandelt.

Gomez et al. (2001) untersuchen Neuro-Evolution, um eine Steuerungsstrategie für die Speicherverwaltung in Multiprozessorsystemen zu erlernen. Der Agent wird durch ein KNN repräsentiert, wobei der Ansatz der Autoren nur dessen Parameter, jedoch nicht dessen Struktur anpasst. Konkret soll der Agent zu jedem Zeitpunkt entscheiden, wie viel Kapazität des zentralen Speichers dem jeweiligen Prozessorkern zur Verfügung steht. Zusätzlich besitzt jeder Prozessorkern noch einen kleineren lokalen Speicher. Die Arbeit lässt sich somit der Einsatzart »Agentenbasierte Ressourcenbelegungsplanung« (siehe Abschnitt 4.1.2) zurechnen. Für die Entscheidungsfindung berücksichtigt der Agent die aktuelle Verarbeitungsrate jedes Prozessorkerns sowie deren Fehlerraten für Zugriffsoperationen (Lesen und Schreiben) auf den lokalen und zentralen Speicher. Obgleich das KNN mehrere Ausgabeneuronen besitzt (entsprechend der Anzahl von Prozessorkernen), handelt der Agent in einem kontinuierlichen Aktionsraum. Die Anregung jedes Ausgabeneurons wird als die Anzahl von Kapazitätseinheiten des zentralen Speichers interpretiert, die dem jeweiligen Prozessorkern zugewiesen

wird. Im Vergleich zu einer statischen Allokation von zentralen Speicherkapazitäten zu Prozessorkernen kann mithilfe des neuroevolutionären Ansatzes die Gesamtverarbeitungsrate eines Vierkernprozessors um 16 Prozent erhöht werden.

Whiteson und Stone (2006) präsentieren ein neues hybrides Lernverfahren, das NEAT und Q-Learning miteinander kombiniert. Im Kontrast zum in Abbildung 3.7 (Abschnitt 3.4) skizzierten Prozess des gradientenfreien bestärkenden Lernens, erfordert das Verfahren, dass die Umgebung als MEP modelliert wird. Der Q-Learning-Algorithmus wird innerhalb der Evaluationsroutine des NEAT-Algorithmus eingebettet. Immer dann, wenn NEAT die Fitness eines zufällig gewählten Genoms bewertet, wird zusätzlich für jeden besuchten Zustand der resultierende Agent durch ein TD-Lernverfahren trainiert. Erst danach werden die Struktur und Parameter des Agenten durch NEAT evolviert. Die Autoren testen ihren Ansatz u. a. anhand eines Lastverteilungsproblems auf einem Computerserver. Im konkreten Fall der Autoren gilt es zu entschieden, in welcher Reihenfolge der Server offene Anfragen bearbeiten soll. Im Wesentlichen handelt es sich um ein Ein-Maschinen-Problem. Die Art und Weise, wie die Autoren das Problem adressieren, entspricht der Einsatzart »Agentenbasierte Reihenfolgeplanung« (siehe Abschnitt 4.1.3). In den Experimenten sind initial 100 offene Anfragen registriert, wobei in den ersten 100 Zeitschritten je Zeitschritt eine neue Anfrage hinzukommt. Insgesamt existieren vier unterschiedliche Anfragetypen. Die Autoren diskretisieren sowohl den Zustands- als auch den Aktionsraum, sodass nicht jeder Auftrag in jedem Zeitschritt separat betrachtet werden muss. Maximal können 200 Anfragen während einer Episode im System registriert sein. Die Autoren teilen diese Menge in vier Untermengen auf und gliedern ferner jede Untermenge gemäß den unterschiedlichen Anfragetypen. Hieraus resultieren 16 Variablen, die den beobachtbaren Zustand der Umgebung bilden. Gleichermaßen hat der Agent 16 Aktionen zur Auswahl, welche die Gliederung des Zustandsraums widerspiegeln. Jede Anfrageart ist mit einer individuellen Nutzenfunktion hinterlegt, die eine zu späte Verarbeitung unterschiedlich bestraft. Das Ziel ist die Maximierung des Nutzens der bearbeiteten Anfragen bzw. die Minimierung der Bestrafung durch zu hohe Bedienzeiten. Hierbei erzielt NEAT + Q bessere Ergebnisse als wenn beide Verfahren jeweils einzeln ausgeführt werden.

Mao et al. (2007) verwenden einen Neuro-Evolution-Algorithmus, der lediglich die Parameter, nicht jedoch die Struktur eines Agenten adaptiert, um eine Steuerungsstrategie für eine Flugzeugwartung zu erlernen. Ankommende Flugzeuge müssen in einer Station enteist werden, bevor sie wieder einsatzbereit sind. Die Enteisungsstation hat eine bestimmte Anzahl von Parzellen zur Verfügung, wodurch mehrere Flugzeuge parallel enteist werden können. Allgemein betrachtet handelt es sich somit um ein Parallel-Maschinen-Problem. Die Autoren

lösen das Problem über einen spieltheoretischen Ansatz. Jedes Mal, wenn eine Parzelle der Enteisungsstation frei wird, können Flugzeuge entscheiden, ob sie die Parzelle für einen bestimmten Zeitpunkt reservieren oder nicht. Die geplante Ankunft von Flugzeugen unterliegt Schwankungen, sodass in manchen Fällen reservierte Zeiten nicht eingehalten werden können. In diesem Fall muss das Flugzeug den reservierten Termin aufgeben und die Airline des Flugzeugs Rücktrittskosten entrichten. Vor diesem Hintergrund sind Flugzeuge angehalten eine Parzelle so spät wie möglich zu reservieren, sodass die Unsicherheit bzgl. der avisierten Ankunftszeit so gering wie möglich ist. Auf der anderen Seite kann eine zu späte Reservierung dazu führen, dass die verbleibenden Reservierungszeitfenster keine fristgerechte Durchführung des Folgeflugs erlauben. In diesem Fall entstehen sogenannte Verspätungskosten. Das Ziel des Agenten ist somit, eine Strategie zu finden, bei welchen die Rücktritts- und Verspätungskosten minimiert werden. Für jede Entscheidung berücksichtigt jeder Agent als Eingangszustand die verbleibende Zeit bis die nächste Parzelle verfügbar wird, die individuelle verbleibende Zeit bis zum nächsten Abflug sowie die Anzahl von konkurrierenden Flugzeugen, welche ebenfalls die Parzelle buchen können. Die Ausgabeschicht jedes Agenten wird lediglich durch ein Neuron mit Tanh-Aktivierungsfunktion repräsentiert. Ausgaben größer null implizieren, dass die Parzelle gebucht werden soll. Die Autoren vergleichen ihre Ergebnisse mit einer FIFO-Steuerung sowie mit einem eigenen entwickelten Ansatz, der Entscheidungen auf Basis statistischer Vergangenheitsdaten fällt. Hierbei verursacht der neuroevolutionäre Ansatz der Autoren lediglich 20 Prozent der Kosten einer FIFO-Steuerung und lediglich die Hälfte der Kosten des auf statistischer Analyse beruhenden Ansatzes.

Mokhtari (2014) kombiniert Neuro-Evolution mit einer variablen Nachbarschaftssuche, um Aufträge in einem Job-Shop-Problem mit zwei Ressourcen zu sequenzieren. Jeder Auftrag muss genau zwei Operationen durchlaufen. Die erste Operation muss entweder auf der ersten oder zweiten Ressource durchgeführt werden, wohingegen die zweite Operation auf der jeweiligen anderen Ressource durchgeführt werden muss. Eine Besonderheit ist, dass Aufträge, welche die erste Operation durchlaufen haben, ohne Verzögerung die zweite Operation auf der anderen Ressource beginnen müssen. Die Bearbeitungszeiten aller Operationen sind für jeden Auftrag individuell. Das Optimierungskriterium ist die Minimierung der Gesamtdauer des Ablaufplans. Zunächst werden durch ein Sortierverfahren Auftragspaare gebildet, die jeweils gemeinsam bearbeitet werden. Die Auftragspaare sind so gestaltet, dass beide Aufträge ihre Bearbeitung auf unterschiedlichen Ressourcen beginnen. Aufgrund der unterschiedlichen Bearbeitungszeiten kann die Situation eintreten, dass ein Auftrag seine erste Operation später starten muss, sodass beide Aufträge zeitgleich fertig werden. Des Weiteren

kommt es gewöhnlich vor, dass ein Auftrag auf die Fertigstellung der zweiten Operation des anderen Auftrags warten muss, bevor das Auftragspaar das System verlassen kann. Diese Zeiten werden im Folgenden als Überschusszeiten bezeichnet. Mittels Neuro-Evolution werden die Parameter eines KNN-basierten Agenten mit fixer Struktur optimiert, der für die Reihenfolgebildung der Auftragspaare verantwortlich ist. Die Arbeit lässt sich somit der Einsatzart »Agentenbasierte Reihenfolgeplanung« zurechnen (siehe Abschnitt 4.1.3). Der Agent berücksichtigt für seine Entscheidungen neun Attribute, u. a. die eben geschilderten Überschusszeiten oder die mittlere Auslastung der beiden Ressourcen. Für jedes Auftragspaar berechnet der Agent auf diese Weise einen Indikator, an welcher Stelle der Sequenz das Auftragspaar eingelastet werden soll. Die durch den Agenten konstruierte Sequenz gilt fortan als Startlösung für die variable Nachbarschaftssuche. Es handelt sich um eine Metaheuristik, die iterativ eine gegebene Lösung verbessert. Im vorliegenden Problem optimiert der Algorithmus die Auftragssequenz hinsichtlich der Reduktion der Stillstandzeiten beider Ressourcen. Das gesamte Lösungskonzept wird anhand verschiedener Probleminstanzen mit 30 bis 120 Aufträgen evaluiert. Dabei weicht die Ergebnisqualität maximal weniger als neun Prozent von der jeweils bekannten optimalen Lösung ab. In manchen Fällen ist der Algorithmus in der Lage die beste Lösung zu finden. Jedoch brilliert der Ansatz insbesondere hinsichtlich der Recheneffizienz auf großen Probleminstanzen. In vielen Fällen ist die Laufzeit um das zehn- bis zwanzigfache geringer als die des exakten Lösungsverfahrens.

Ein weiterer themenbezogener Konferenzbeitrag sowie ein darauf aufbauender Journal-Artikel stammen von Du et al. (2019; 2020). In beiden Veröffentlichungen betrachten die Autoren ein Ablaufplanungsproblem aus der Raumfahrt, in welchem eine Menge von Anfragen durch eine Menge von Satelliten bewältigt werden muss. Das zugrundeliegende Optimierungsproblem ist somit ein Ressourcenzuordnungsproblem, in welchem Anfragen auf Satelliten alloziert werden müssen. Vor diesem Hintergrund ordnet sich die Arbeit der Einsatzart »Agentenbasierte Ressourcenbelegungsplanung« zu (siehe Abschnitt 4.1.2). Darüber hinaus berücksichtigen die Autoren diverse Nebenbedingungen, wie bspw. die maximale Energiemenge, die ein Satellit in einem Orbitzyklus verbrauchen darf oder die maximale Zeitspanne, in der Satellitenaufnahmen an eine Empfangsstation gesendet werden müssen. Für die Zuweisung von Anfragen zu Satelliten trainieren die Autoren einen KNN-basierten Agenten mithilfe des NEAT-Algorithmus. Eine Besonderheit der Implementierung der Autoren ist, dass nach jedem Evolutionszyklus zusätzlich jedes Genom durch die Metaheuristik »Tabu Search« lokal optimiert wird. Ungeachtet dessen verwenden die Autoren historische Daten anstatt einer Fitnessfunktion, um den Agenten zu trainieren. Die Fitness

jedes Genoms wird somit über den Fehler zwischen der KNN-Prognose und der erwarteten Ausgabe berechnet. Der zu trainierende Agent besitzt lediglich ein Ausgabeneuron, welches die Wahrscheinlichkeit ausgibt, dass eine gegebene Anfrage von einem bestimmten Satelliten fristgerecht bearbeitet werden kann. Für die Allokation einer Anfrage iteriert eine Entscheidungslogik über alle Satelliten und berechnet die Wahrscheinlichkeit, dass der jeweilige Satellit die Anfrage fristgerecht beantwortet. Die Entscheidungslogik wählt sodann den Satelliten mit der maximalen Wahrscheinlichkeit aus. Die Autoren vergleichen ihren Ansatz mit mehreren heuristischen und metaheuristischen Lösungsverfahren anhand der Anzahl erfüllter Anfragen, wobei jede Anfrage zusätzlich mit einer Priorität gewichtet ist. Bei einem Betrachtungshorizont von kleiner gleich 400 Anfragen leistet das Verfahren schlechtere Ergebnisse als ein adaptives Nachbarschaftssuchverfahren (Metaheuristik), jedoch bessere Ergebnisse als jedes verglichene heuristische Verfahren. Ab einem Betrachtungshorizont von 500 Anfragen leistet das Verfahren der Autoren die besten Ergebnisse unter allen verglichenen Lösungsansätzen.

4.3 Zusammenfassung und Diskussion der Forschungslücke

Für die Darstellung des Stands der Wissenschaft und Technik wurden insgesamt 71 Arbeiten analysiert. Die Mehrheit der geprüften Veröffentlichungen untersucht den Einsatz von gradientenabhängigen RL-Verfahren für die Produktionsablaufplanung. Zur Beantwortung der ersten Forschungsfrage (siehe Kapitel 1.2) wurden anhand dieser Veröffentlichungen fünf Kategorien abgeleitet, welche die verschiedenen Einsatzarten von RL-Verfahren für die Produktionsablaufplanung beschreiben, und zwar:

- die **agentenbasierte Auswahl von Prioritätsregeln**, bei welcher der Agent entscheidet, mit welcher Prioritätsregel der nächste Auftrag aus der vorgelagerten Warteschlange entnommen wird. Im weiteren Sinne handelt es sich um einen Spezialfall der agentenbasierten Sequenzierung von Aufträgen, wobei die Auswahl von Aufträgen lediglich indirekt durch Auswahl einer Prioritätsregel erfolgt;
- die **agentenbasierte Ressourcenbelegungsplanung**, bei welcher der Agent einen Auftrag einer stationären Ressource (z. B. Maschine) bzw. mobilen Ressource (z. B. Produktionspersonal, Transportroboter) oder eine mobile Ressource einer stationären Ressource zuweist;

- die **agentenbasierte Reihenfolgeplanung**, bei welcher der Agent unmittelbar oder mittelbar (Prioritätsregeln ausgenommen) entscheidet, welcher Auftrag aus der vorgelagerten Warteschlange entnommen wird;
- die **agentenbasierte Losbildung**, bei welcher der Agent über die Rüstung einer stationären Ressource entscheidet, durch die lediglich Aufträge derselben Familie auf der Ressource bearbeitet werden können. Das Ergebnis ist eine Sequenz von Auftragslosen, die auf der Ressource abgearbeitet wird;
- das **agentenbasierte Reparieren von ungültigen Ablaufplänen**, bei welcher der Agent einen gegebenen Ablaufplan, der eine oder mehrere Restriktionen des zugrundeliegenden Optimierungsproblems verletzt, iterativ zu einem neuen gültigen Ablaufplan umbaut.

Werden diejenigen Arbeiten miteinbezogen, welche die agentenbasierte Auswahl von Prioritätsregeln für die Reihenfolgebildung von Aufträgen untersuchen, so ergibt sich, dass die Sequenzierung von Aufträgen die am häufigsten untersuchte Einsatzart von RL-Verfahren für die Produktionsablaufplanung darstellt. Die zu diesem Zweck bisher veröffentlichten Konzepte unterliegen jedoch Limitationen. Die agentenbasierte Auswahl von Prioritätsregeln erlaubt lediglich eine indirekte Einflussnahme auf die Reihenfolgebildung von Aufträgen, da die Auftragssequenz konkret durch die ausgewählte Prioritätsregel bestimmt wird. Ferner können mithilfe von Prioritätsregeln nur unmittelbar diejenigen Aufträge ausgewählt werden, die das Prioritätskriterium der jeweiligen Regel zum Entscheidungszeitpunkt am meisten erfüllen. Hieraus resultiert eine starke Einschränkung des untersuchbaren Lösungsraums. Demgegenüber wurden in Abschnitt 4.1.3 einige Arbeiten vorgestellt, in welchen jede Aktion des jeweiligen Agenten mit der Auswahl eines spezifischen Auftrags assoziiert wird. Der Hauptnachteil dieses Konzepts ist dessen schlechte Skalierbarkeit auf andere Auftragsmengen. Die meisten der vorgestellten Arbeiten nehmen eine maximale Auftragsmenge an, die sich in der vorgelagerten Warteschlange befindet und dimensionieren dementsprechend den Aktionsraum des Agenten. Sofern sich weniger Aufträge als die maximale Auftragsmenge in der Warteschlange befinden, werden nicht benötigte Aktionen temporär als nicht auswählbar vermerkt. Ungeachtet dessen kann mithilfe dieses Konzepts ein angelernter Agent nicht in solchen Umgebungen eingesetzt werden, in denen ein Auftrag aus einer höheren Anzahl von Aufträgen selektiert werden muss als die maximal angenommene Auftragsmenge während des Trainings. Eine höhere Anzahl an Aufträgen erfordert einen größeren Aktionsraum, was mit einer strukturellen Erweiterung und somit einem erneuten Training des Agenten einhergeht.

Vor diesem Hintergrund zeichnet sich der erste Forschungsbedarf ab, nämlich die Entwicklung eines skalierbaren Einsatzkonzepts, wie gradientenabhängige RL-Verfahren für die unmittelbare Sequenzierung von Aufträgen eingesetzt werden können. Die Anforderung der Skalierbarkeit gilt dann als erfüllt, wenn ein angelernter Agent für jegliche Auftragsmengen und Warteschlangenlängen eingesetzt werden kann, unabhängig davon, welche Auftragsmengen und Warteschlangenlängen während des Trainings beobachtet wurden.

Hinsichtlich derjenigen Publikationen, die gradientenfreie RL-Verfahren für die Ablaufplanung präsentieren, weisen lediglich zwei der vierzehn geprüften Arbeiten einen produktionslogistischen Bezug auf. Beide Arbeiten untersuchen die Kreuzentropie-Methode, wobei die Art und Weise ihrer Verwendung vielmehr der Anwendung einer Metaheuristik gleicht. So dient in beiden Arbeiten die Kreuzentropie-Methode als Lösungsverfahren, das auf jeder Probleminstanz neu rechnet. In beiden Arbeiten werden keine Untersuchungen präsentiert, wie sich die Lösungsgüte der angelernten Modelle auf anderen Probleminstanzen verhält. Des Weiteren ist hervorzuheben, dass bisher noch keine Arbeiten existieren, die Neuro-Evolution-Verfahren für die Produktionsablaufplanung untersuchen. Unter den gradientenfreien RL-Verfahren sind insbesondere Neuro-Evolution-Verfahren mit den gradientenabhängigen DRL-Verfahren vergleichbar, da bei diesen ebenfalls der Agent stets durch ein KNN repräsentiert wird.

Zusammengefasst resultiert als Forschungslücke, dass gradientenfreie RL-Verfahren bisher noch nicht für die Produktionsablaufplanung mit der Zielstellung untersucht wurden, eine agentenbasierte Lösungsstrategie zu erlernen, die ebenfalls auf neuen Probleminstanzen gute Ergebnisse erzielt, hierbei jedoch analog zu Prioritätsregeln Ergebnisse in Echtzeit produzieren kann.

Ungeachtet der Unterteilung in gradientenabhängige und gradientenfreie RL-Verfahren ist festzustellen, dass die 71 analysierten Arbeiten kaum Details hinsichtlich des Entwurfs und der Implementierung von Agenten und Umgebungen präsentieren. Insbesondere die technische Integration eines Agenten mit dessen Umgebung wird oftmals nur unzureichend beschrieben. Demzufolge kann als weitere Forschungslücke herausgearbeitet werden, dass bisher noch kein Vorgehensmodell existiert, das detailliert und allgemeingültig zur Konzeption und Implementierung von gradientenabhängigen und gradientenfreien RL-Verfahren für die Produktionsablaufplanung anleitet.

5 Eine Methode zum Einsatz von bestärkenden Lernverfahren für die Produktionsablaufplanung

Wie in der Schlussbetrachtung zum Stand der Wissenschaft und Technik beschrieben, existiert bisher noch keine systematische Vorgehensweise, wie RL-Verfahren für die Produktionsablaufplanung einsetzbar gemacht werden können. Basierend auf der theoretischen Fundierung in den Kapiteln 2 und 3 soll im Folgenden eine Methode herausgearbeitet werden, die das Vorgehen zum Einsatz von RL-Verfahren für die Produktionsablaufplanung formalisiert. In Abschnitt 5.1 soll zunächst der aktuelle Prozess der Produktionsablaufplanung als Ausgangspunkt reflektiert werden, an welchen die zu entwickelnde Methode anknüpfen möchte. Hierauf basierend soll eine Anforderungsdefinition für die zu entwickelnde Methode abgeleitet werden. Die eigentliche Methode fußt im Wesentlichen auf zwei Bestandteilen. In Abschnitt 5.2 wird zunächst diskutiert, dass es einen Wandel von einer Produktionsablaufplanung zu einer agentenbasierten Produktionsablaufsteuerung bedarf. Daran anschließend präzisiert Abschnitt 5.2 den Prozess und das Funktionsprinzip der agentenbasierten Produktionsablaufsteuerung. Der zweite Bestandteil der Methode ist ein Vorgehensmodell zur Projektierung und Entwicklung von agentenbasierten Produktionsablaufsteuerungen. Diesem Vorgehensmodell widmet sich Abschnitt 5.3.

Ergänzende Information Die elektronische Version dieses Kapitels enthält Zusatzmaterial, auf das über folgenden Link zugegriffen werden kann https://doi.org/10.1007/978-3-658-41751-2_5.

S. Lang, *Methoden des bestärkenden Lernens für die Produktionsablaufplanung*,
https://doi.org/10.1007/978-3-658-41751-2_5

5.1 Ausgangssituation, Problemstellung und Anforderungsdefinition

Als Ausgangspunkt wird zunächst der aktuelle Prozess der Produktionsablaufplanung gemäß einem etablierten PPS-Modell betrachtet. Zu diesem Zweck wurde bereits in Abbildung 2.2 (Abschnitt 2.2) der Prozess der Eigenfertigungsplanung und -steuerung des Aachener PPS-Modells skizziert. Eine zentrale Schlussfolgerung war, dass die Ressourcenbelegungsplanung (respektive die sequenzielle Feinterminierung und Ressourcenfeinplanung) sowie die daran anschließende Reihenfolgeplanung die Produktionsablaufplanung im engeren Sinne repräsentieren, derweilen die vorgelagerte Losgrößenplanung der Produktionsablaufplanung im weiteren Sinne zugerechnet wird.

Im Folgenden soll der momentane Prozess der Produktionsablaufplanung gemäß dem Aachener PPS-Modell im Detail betrachtet werden. Zu diesem Zweck präzisiert Abbildung 5.1 die blaugefärbten Aktivitäten der Produktionsablaufplanung aus Abbildung 2.2 als UML-Aktivitätsdiagramm. Eine Erklärung der Notation für UML-Aktivitätsdiagramme ist als Anhang G dem elektronischen Zusatzmaterial beigefügt. Der Prozessablauf basiert im Wesentlichen auf den Ausführungen von Schuh et al. (2012a, S. 53 ff.) zur Losgrößenrechnung, Feinterminierung, Ressourcenfeinplanung und Reihenfolgeplanung. Ausgehend von einem vorbestimmten Eigenfertigungsprogramm wird zunächst überprüft, ob die einzulastenden Produktionsaufträge zu Produktionslosen konsolidiert werden können. Wie bereits in Abschnitt 2.2 beschrieben, wird mithilfe der Bildung von Losen ein kostenoptimaler Kompromiss zwischen niedrigen Umlaufbeständen und einer geringen Anzahl von Rüstvorgängen angestrebt, bei welchen es sich um gegensätzliche Zielstellungen handelt. Die Losgrößen werden häufig auf Basis von Erfahrungswissen einmalig und intuitiv festgelegt oder mithilfe mathematischer Formeln für bestimmte Produktionszeiträume berechnet. Darauffolgend werden im Zuge der Ressourcenbelegungsplanung die Start- und Endzeiten der einzelnen Produktionsaufträge bzw. der kumulierten Produktionslose durch Vorwärts-, Rückwärts- oder Mittelpunktterminierung bestimmt. Falls die resultierenden Start- und Endzeiten unzulässig sind (bspw. aufgrund verletzter Fertigstellungsfristen), kann durch eine Aufteilung bzw. Zusammenführung von Produktionslosen eine Durchlaufzeitverkürzung angestrebt werden. Falls im Voraus bereits festgestellt wurde, dass eine Losbildung nicht zielführend ist, können neue Start- und Endzeiten mit einem alternativen Terminierungsverfahren bestimmt werden. Im nächsten Schritt werden die Kapazitätsbedarfe für die erforderlichen Produktionsressourcen ermittelt. Die Kapazitätsbedarfe werden

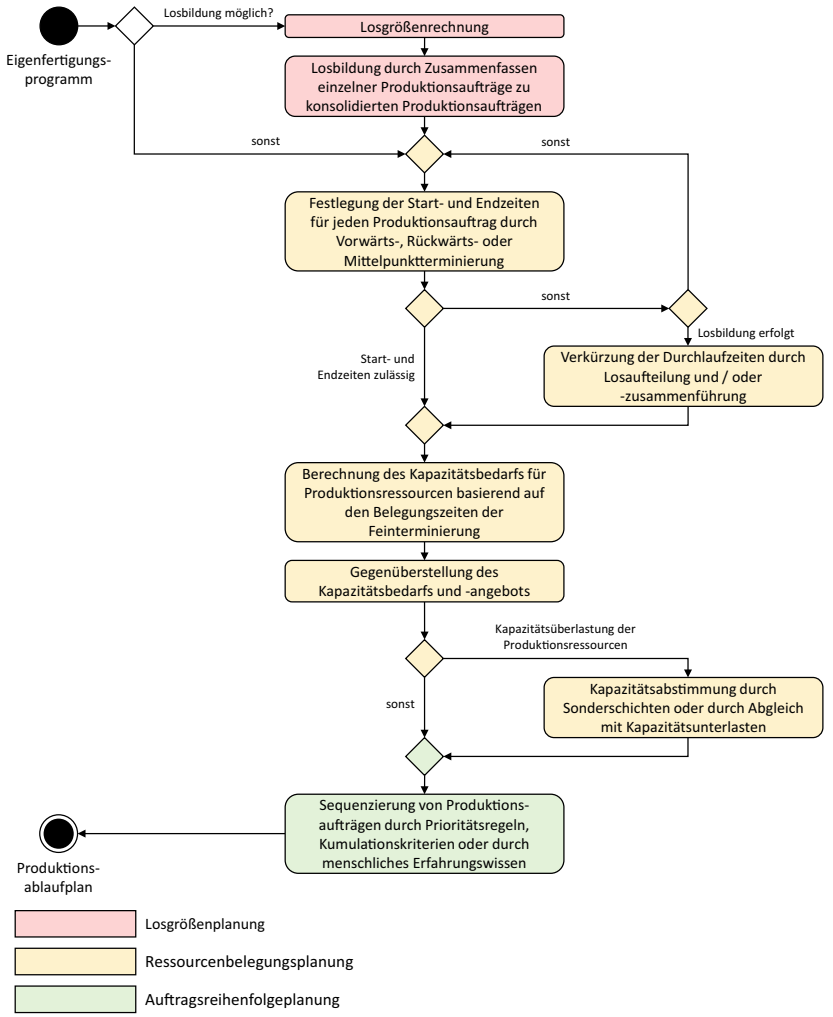


Abbildung 5.1 Detaillierter Prozess der Produktionsablaufplanung gemäß dem Aachener PPS-Modells

nicht nur auf einzelne Produktionsressourcen, sondern ebenfalls auf verschiedene Produktionsperioden aufgeschlüsselt, die aus den zuvor ermittelten Start- und Endzeiten resultieren. Durch eine Gegenüberstellung des Kapazitätsbedarfs mit dem verfügbaren Kapazitätsangebot wird analysiert, ob in bestimmten Produktionsperioden Kapazitätsüberlasten auftreten. Etwaige Kapazitätsüberlasten können mithilfe von Sonderschichten oder durch Verschiebung von Produktionsaufträgen in Perioden, in denen die verfügbaren Kapazitäten nicht voll ausgeschöpft sind, aufgelöst werden. Aufgrund der Ressourcenbelegungsplanung entstehen Warteschlangen aus einzelnen Aufträgen oder Auftragslosen vor den jeweiligen Produktionsressourcen. Im Zuge der Auftragsreihenfolgeplanung wird die Bearbeitungsreihenfolge der Aufträge bzw. Auftragslose auf jeder Produktionsressource festgelegt. Für die Reihenfolgebildung werden häufig Prioritätsregeln oder Kumulationskriterien (z. B. Auftragsauswahl gemäß der gerüsteten Produktfamilie zur Minimierung von Rüstzeiten) zu Rate gezogen. Darüber hinaus kann ebenfalls auf eine explizite Auftragsreihenfolgeplanung verzichtet werden. Stattdessen entscheiden die Fachkräfte im Fertigungsbereich erfahrungsbasiert über die Bearbeitung des nächsten Auftrags(loses), sobald eine Kapazitätseinheit auf der erforderlichen Ressource verfügbar wird.

Das Vorgehen für die Produktionsablaufplanung gemäß dem Aachener PPS-Modell zeichnet sich insbesondere durch einen klaren stringenten Ablauf sowie durch die Verwendung von einfach verständlichen und aufwandsarm zu realisierenden Methoden aus. Hierdurch ist es leicht zugänglich für Produktionsplaner*innen und begünstigt ebenfalls eine softwaretechnische Realisierung. Ungeachtet dessen lassen sich am geschilderten Vorgehen einige Charakteristiken beobachten, die insbesondere in hochdynamischen und volatilen Produktionsumgebungen nachteilig wirken:

- Die Produktionsablaufplanung des Aachener PPS-Modells geht von einem fix determinierten Eigenfertigungsprogramm aus. Hierbei handelt sich um eine Liste von Aufträgen, die in einer definierten Planungsperiode produziert werden sollen. Die Losgrößenrechnung, Ressourcenbelegungs- und Reihenfolgeplanung unterliegen somit der Annahme eines statischen Auftragshorizonts. Die Produktionsablaufplanung des Aachener PPS-Modells ist somit nur in geringem Maße für einen dynamischen stochastischen Auftragshorizont geeignet, bei welchen in zufälligen Zeitabständen Produktionsaufträge in das System gelangen, die aufgrund von eng gesetzten Lieferterminen in der aktuellen Planungsperiode produziert werden müssen.

- Die mangelnde Eignung des Vorgehens für hochdynamische und volatile Produktionsumgebungen lässt sich ferner anhand der Art und Weise begründen, wie Produktionsablaufpläne erstellt werden. Die Losgrößenrechnung, Ressourcenbelegungs- und Reihenfolgeplanung finden nicht integriert, sondern stufenweise nacheinander statt. Dieses Vorgehen birgt insbesondere einen Nachteil. Angenommen in einer vorgelagerten Planungsstufe wird eine anforderungsgerechte Lösung generiert, so ist nicht sichergestellt, dass hieran anschließend ein zulässiges Planungsergebnis in einer der darauffolgenden Stufen ermittelt werden kann. Im besten Fall kann mithilfe von einfachen Heuristiken und Entscheidungsregeln ein zulässiger Alternativplan generiert werden, der jedoch gewöhnlich eine geringere Lösungsqualität aufweist. Im schlechtesten Fall muss der Planungsprozess vollständig von Neuem, ausgehend von der Losgrößenrechnung, begonnen werden. Dieses Szenario wird in Abbildung 5.1 nicht dargestellt, findet jedoch im Prozessmodell der Eigenfertigungsplanung und -steuerung des Aachener PPS-Modells (Abbildung 2.2 in Abschnitt 2.2) Berücksichtigung. Ein ähnliches Problem tritt zu Tage, wenn in der laufenden Periode ein zusätzlicher Auftrag im Produktionsablaufplan berücksichtigt werden muss. Sofern durch Heuristiken und Entscheidungsregeln kein zulässiger Alternativplan ermittelt werden kann, muss ebenfalls eine grundlegende Neuplanung in Erwägung gezogen werden.
- Das geschilderte Problem wird durch den Umstand verschärft, dass das Aachener PPS-Modell für die Losgrößenrechnung, Ressourcenbelegungs- und Reihenfolgeplanung vornehmlich einfache heuristische Verfahren empfiehlt, die Lösungen nach einer fest definierten Vorschrift konstruieren. Diese Verfahren haben den Nachteil, dass sie den Lösungsraum eines Problems nicht systematisch durchsuchen können und für eine gegebene Eingabe lediglich eine Lösung generieren. Vor diesem Hintergrund bieten sie lediglich einen geringen Handlungsspielraum, wenn eine vorgeschlagene Lösung nicht den Planungsanforderungen entspricht.

Die zu entwickelnde Methode zum Einsatz von RL-Verfahren für die Produktionsablaufplanung soll eine Ergänzung zum etablierten Aachener PPS-Modell darstellen. Zu diesem Zweck soll die Methode Lösungen für die geschilderten Defizite anbieten. Konkret soll sie die Produktionsablaufplanung unter Berücksichtigung eines dynamischen stochastischen Auftragshorizonts sowie enger Zeitfenster für die Entscheidungsfindung begünstigen. Vor diesem Hintergrund

werden die folgenden Anforderungen (A) an die zu entwickelnde Methode postuliert:

A1: Die Methode soll die Verwendung von RL-trainierten Agenten für die Produktionsablaufplanung beschreiben.
A2: Die Methode soll die Konzeption und Implementierung von RL-Verfahren für die Produktionsablaufplanung beschreiben.
A3: Die Methode soll die Verwendung von RL-trainierten Agenten in einer Weise anleiten, dass eine echtzeitfähige, datenbasierte und adaptive Produktionsablaufplanung ermöglicht wird. Die Erfüllung dieser Anforderung dient dem Zweck einer Produktionsablaufplanung unter Berücksichtigung eines dynamischen stochastischen Auftragshorizonts sowie enger Zeitfenster für die Entscheidungsfindung.
A4: Beim Einsatz von RL für die Produktionsablaufplanung soll weiterhin der Mensch die letzte Entscheidungsinstanz darstellen. Vor diesem Hintergrund soll die zu entwickelnde Methode erlauben, dass eine RL-basierte Entscheidung durch den Menschen verworfen werden kann. Ferner soll die Methode ein Konzept präsentieren, wie RL-trainierte Agenten zusätzlich von der Expertise menschlicher Planer*innen lernen können.
A5: Die Methode soll so konzipiert sein, dass sie theoretisch für jedes Problem der Produktionsablaufplanung angewandt werden kann.
A6: Im Gegensatz zur Produktionsablaufplanung des Aachener PPS-Modells soll die zu entwickelnde Methode die Losbildung, Ressourcenbelegungs- und Reihenfolgeplanung nicht stufenweise, sondern integriert betrachten. Im Fall einer unzulässigen Lösung soll auf diese Weise die Berechnung von alternativen Lösungskandidaten vereinfacht werden.
A7: Für die Berechnung alternativer Lösungskandidaten ist es notwendig, dass RL-basierte Planungsverfahren, die gemäß der zu entwickelnden Methode konzipiert und implementiert werden, den Lösungsraum eines Problems durchsuchen können.
A8: Die Methode soll gewährleisten, dass der Einsatz von RL-trainierten Agenten zu einem gewissen Grad auf unterschiedlich große Probleminstanzen skalierbar ist. Konkret soll ein RL-trainierter Agent für eine flexible Anzahl von Aufträgen einsetzbar sein. Hingegen wird auf die Anforderung verzichtet, dass ein RL-trainierter Agent ebenfalls für eine beliebige Anzahl von Produktionsressourcen eingesetzt werden kann. Es wird angenommen, dass über einen mittel- bis langfristigen Planungshorizont die meisten Produktionssysteme über eine konstante Anzahl von Betriebsmitteln verfügen,

sodass bei einer Veränderung des Systems ausreichend Zeit bleibt, um einen strukturell angepassten Agenten zu trainieren.

5.2 Von der Produktionsablaufplanung zur agentenbasierten Produktionsablaufsteuerung – Prozessmodell und Funktionsprinzip

Zunächst sei festzustellen, dass die Anforderungen A1, A2 und A3 hohe Synergien zueinander aufweisen. Durch die Konzeption und Implementierung von RL-Verfahren (A1) für das Training und den Einsatz von Agenten (A2) kann eine echtzeitfähige, datenbasierte und adaptive Produktionsablaufplanung realisiert werden (A3). Die agentenbasierte Produktionsablaufplanung ist echtzeitfähig, da der Agent durch ein ML-Modell repräsentiert wird. Im Kern approximiert das ML-Modell eine mathematische Funktion, wodurch ungeachtet der Anzahl von Funktionsvariablen und -parametern eine Ausgabe gewöhnlich innerhalb von einer Sekunde berechnet wird. Diese Aussage wird in Kapitel 6 anhand von einigen Beispielen untermauert. Die agentenbasierte Produktionsablaufplanung ist zudem datenbasiert, weil Entscheidungen stets in Abhängigkeit vom aktuellen Umgebungszustand getroffen werden. Ferner ist die agentenbasierte Produktionsablaufplanung adaptiv, weil Entscheidungen individuell auf den aktuellen Umgebungszustand zugeschnitten sind. Die Eigenschaft der Anpassungsfähigkeit wird insbesondere dadurch untermauert, dass der Agent sich kontinuierlich anhand von beobachteten Zuständen und erhaltenen Belohnungen optimieren kann.

In Anbetracht der dargestellten Eigenschaften erscheint der Begriff der Produktionsablaufplanung nicht mehr geeignet. Vielmehr wird durch den Einsatz RL-trainierter Agenten die Planung von Produktionsabläufen als Steuerungsproblem behandelt. Durch die Anwendung von RL-Verfahren vollzieht die Produktionsablaufplanung einen Wandel zur agentenbasierten Produktionsablaufsteuerung. Die Änderung dieser Betrachtungsweise ist der erste Ansatzpunkt, um eines der Hauptprobleme der konventionellen Produktionsablaufplanung zu lösen. Wie im vorhergehenden Abschnitt geschildert geht diese von einem statischen Auftragshorizont aus. Die Aufträge innerhalb des Horizonts werden als Auftragsliste im Planungsprozess simultan berücksichtigt. Eine Einlastung zusätzlicher Produktionsaufträge, die während des Planungsprozesses nicht miteinbezogen wurden, führt gewöhnlich zu einer Verminderung der Lösungsgüte des ursprünglichen

Produktionsablaufplans. Der Verzicht auf eine in sich abgeschlossene Vorplanung auf Basis von Auftragslisten, zu Gunsten einer agentenbasierten Steuerung, die lediglich einzelne Aufträge betrachtet, dient als Grundpfeiler, um eine hochqualitative Produktionsablaufplanung für einen dynamischen stochastischen Auftragshorizont zu realisieren.

Im Folgenden soll zunächst diskutiert werden, wie sich der Prozess der agentenbasierten Produktionsablaufsteuerung im Vergleich zur konventionellen Produktionsablaufplanung gemäß Aachener PPS-Modell gestaltet. Zu diesem Zweck präsentiert Abbildung 5.2 ein entsprechendes Prozessmodell als UML-Aktivitätsdiagramm. Im Gegensatz zur konventionellen Produktionsablaufplanung ist der Ausgangspunkt der agentenbasierten Produktionsablaufsteuerung kein statisches Eigenfertigungsprogramm, sondern eine dynamische Liste mit freigegebenen Aufträgen, die zwischen null und unendlich vielen Aufträgen enthält (Startknoten in Abbildung 5.2). Der in Abbildung 5.2 dargestellte Prozess ist so gestaltet, dass dieser die Anforderungen A5 (Allgemeingültigkeit für die Produktionsablaufplanung) und A6 (Integration von Losbildung, Ressourcenbelegungs- und Reihenfolgeplanung) bestmöglich berücksichtigt.

Um Anforderung A5 Rechnung zu tragen, werden im Folgenden zwei Modelle für die Beschreibung von Produktionsbereichen formuliert, die ebenfalls im Prozess der agentenbasierten Produktionsablaufsteuerung unterschieden werden. Beide Modelle werden in Abbildung 5.3 veranschaulicht. Die erste Produktionsbereichsvariante (Abbildung 5.3 (a) bzw. die rechte Alternative im ersten Entscheidungsknoten in Abbildung 5.2) zeichnet sich dadurch aus, dass jede Produktionsressource durch eine eigene Warteschlange gespeist wird. Sie ist am universellsten einsetzbar, um verschiedene Produktionssysteme (z. B. Fließfertigung, Werkstattfertigung) als Produktionsablaufplanungsproblem zu modellieren (z. B. Flow-Shop und Job-Shop). Für Fließfertigungssysteme und ähnliche Produktionsstrukturen werden die Produktionsressourcen, deren Anzahl flexibel variiert werden kann, als gleichartig betrachtet. Ferner wird das Exklusiv-Oder-Gatter (XOR) vernachlässigt. Somit können beliebig stufige Fertigungssysteme durch Verkettung von Produktionsbereichen der Variante (a) modelliert werden. Hingegen werden für Werkstattfertigungen und verwandte Systeme die Produktionsressourcen als unterschiedlich betrachtet und das Exklusiv-Oder-Gatter berücksichtigt. Durch das Exklusiv-Oder-Gatter wird gewährleistet, dass ein Auftrag jede Produktionsressource beliebig oft besuchen kann. Auf diese Weise sind Fertigungspläne jeglicher Komplexität abbildbar. Hinsichtlich Anforderung A6 erlaubt Variante (a) zumindest eine integrierte Reihenfolgeplanung und Losbildung. Die Integration von Reihenfolgeplanung und Losbildung wird in Abschnitt 5.2.2 detailliert erläutert. Zusätzlich erfordert Variante (a) eine

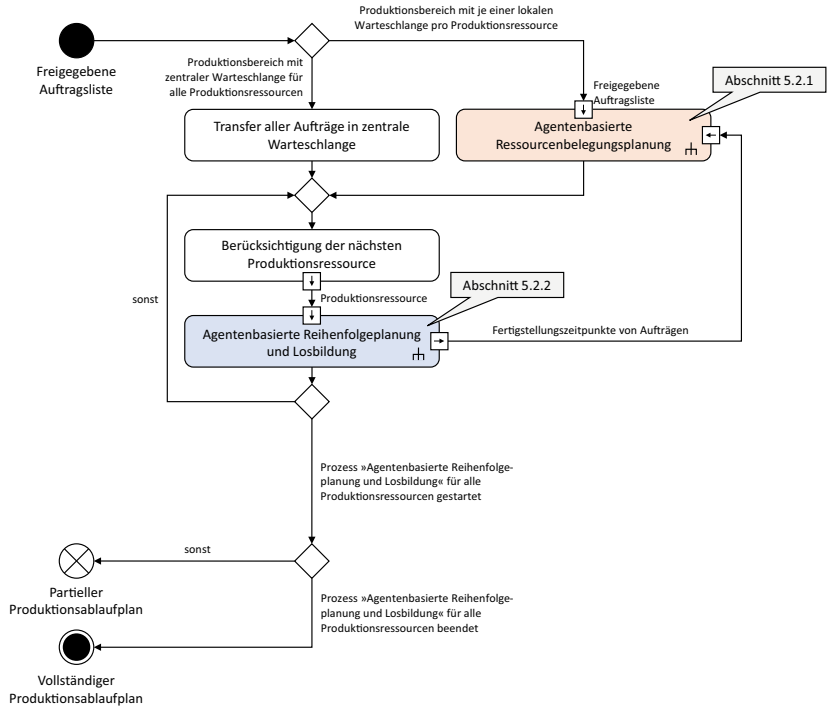


Abbildung 5.2 Prozess der agentenbasierten Produktionsablaufsteuerung

separate Ressourcenbelegungsplanung. Immer dann, wenn ein neuer Auftrag in den Produktionsbereich eintritt, entscheidet ein Agent über dessen Allokation zu einer Produktionsressource. Vor diesem Hintergrund erfordert eine rein agentenbasierte Produktionsablaufsteuerung in Variante (a) mindestens zwei Agenten, wobei ein Agent die Allokation und ein weiterer Agent die Sequenzierung von Aufträgen verantwortet. Ferner ist es möglich, jedoch methodisch nicht zwingend erforderlich, dass für jede Produktionsressource ein eigener Agent angelernt wird, der für die Reihenfolgebildung der jeweils vorgelagerten Warteschlange verantwortlich ist.

In der zweiten Produktionsbereichsvariante (Abbildung 5.3 (b) bzw. die untere Alternative im ersten Entscheidungsknoten in Abbildung 5.2) werden alle Produktionsressourcen durch eine zentrale Warteschlange gespeist. Variante (b) eignet

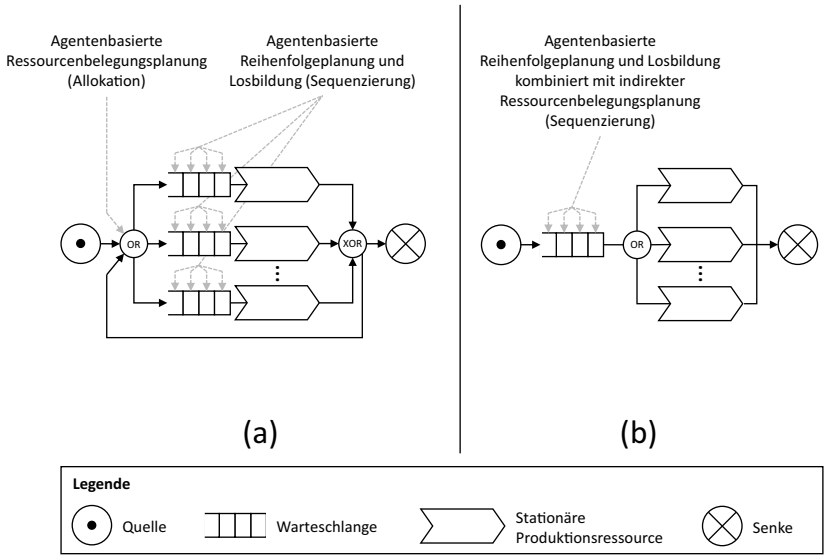


Abbildung 5.3 Die agentenbasierten Produktionsablaufplanung unterscheidet zwei Modelle zur Abbildung von Produktionsbereichen, und zwar Produktionsbereiche mit (a) lokalen Warteschlangen für jede Produktionsressource und (b) zentraler Warteschlange für alle Produktionsressourcen

sich, um Parallel-Maschinen-Umgebungen und einzelne Stufen von Fließfertigungssystemen zu modellieren, in welchen alle stationären Ressourcen durch eine gemeinsame Warteschlange bedient werden. Ferner eignet sich die Variante für die Modellierung einzelner Fertigungszentren mit mehreren Bearbeitungskapazitäten. Variante (b) erlaubt eine ganzheitliche Produktionsablaufsteuerung mit lediglich einem Agenten. Konkret kann auf eine explizite Ressourcenbelegungsplanung verzichtet werden, sodass lediglich ein Agent notwendig ist, der die Reihenfolge- und Losbildung steuert. Zu jedem Zeitpunkt, wenn eine Bearbeitungskapazität verfügbar ist, entscheidet der Agent, welcher Auftrag als nächstes auf der Produktionsressource bearbeitet wird. Hieraus resultiert eine implizite ereignisgesteuerte Ressourcenbelegungsplanung.

Die Modelle der Produktionsbereiche (a) und (b) können flexibel miteinander kombiniert und verkettet werden, um Fabriksysteme jeglicher Art abzubilden. In Abhängigkeit von der Art des Produktionsbereichs, der jeweiligen Anzahl enthaltener Produktionsressourcen, der Eigenschaften der Produktionsressourcen

und der geltenden lokalen Optimierungskriterien ist für jeden Produktionsbereich das Training und die Integration eigener Agenten notwendig. Der Prozess der Produktionsablaufsteuerung endet, sobald der Unterprozess der agentenbasierten Reihenfolgeplanung und Losbildung für alle Produktionsressourcen abgeschlossen ist. Im Folgenden sollen die Unterprozesse »Agentenbasierte Ressourcenbelegungsplanung« und »Agentenbasierte Reihenfolgeplanung und Losbildung« näher beleuchtet werden.

5.2.1 Agentenbasierte Ressourcenbelegungsplanung

Im Rahmen der agentenbasierten Ressourcenbelegungsplanung werden freigegebene Aufträge Produktionsressourcen zugewiesen. Abbildung 5.4 präsentiert das Prozessmodell der agentenbasierten Ressourcenbelegungsplanung als UML-Aktivitätsdiagramm.

Der Prozess wird nur dann aufgerufen, sofern der Agent die Produktionsabläufe für einen Bereich steuert, in dem jede Produktionsressource durch eine eigene lokale Warteschlange bedient wird. In diesem Fall wird der Prozess immer dann initiiert, sobald ein neuer Auftrag freigegeben wird und dann für jeden nichtallozierten Auftrag in der freigegebenen Auftragsliste wiederholt. Die Wiederholung des Prozessablaufs für nichtallozierte Aufträge ist insbesondere dann relevant, wenn sich das Produktionssystem zu Beginn in einem nichteingeschwungenen Zustand befindet, sodass eine initiale Belegung erforderlich ist. Im ersten Schritt beobachtet der Agent den nächsten nichtallozierten Auftrag und schlägt darauffolgend eine Entscheidung vor, welcher Produktionsressource der Auftrag zugewiesen werden soll. Die Art und Weise, wie der Agent eine Ressourcenbelegungsentscheidung ermittelt, wird im Rahmen von Abschnitt 5.3.2.1 detailliert beschrieben.

Im nächsten Schritt erfolgt eine Evaluation, ob die vorgeschlagene Agentenentscheidung den Anforderungen der Produktion entspricht. Die Evaluation wird im besten Fall auf Basis menschlicher Expertise durchgeführt. In Anbetracht dessen, dass die agentenbasierte Produktionsablaufsteuerung echtzeitfähig sein soll (Anforderung A3), muss der Evaluationsprozess durch den Menschen so einfach und effizient wie möglich gestaltet sein. Für die Implementierung einer agentenbasierten Produktionsablaufsteuerung in einem realen System empfiehlt sich eine lediglich implizite Evaluation durch den Menschen. Ein mögliches Realisierungskonzept umfasst die Installation eines Sensorsystems, das erfasst, ob der Mensch den aktuell betrachteten Auftrag derjenigen Produktionsressource zuweist, die

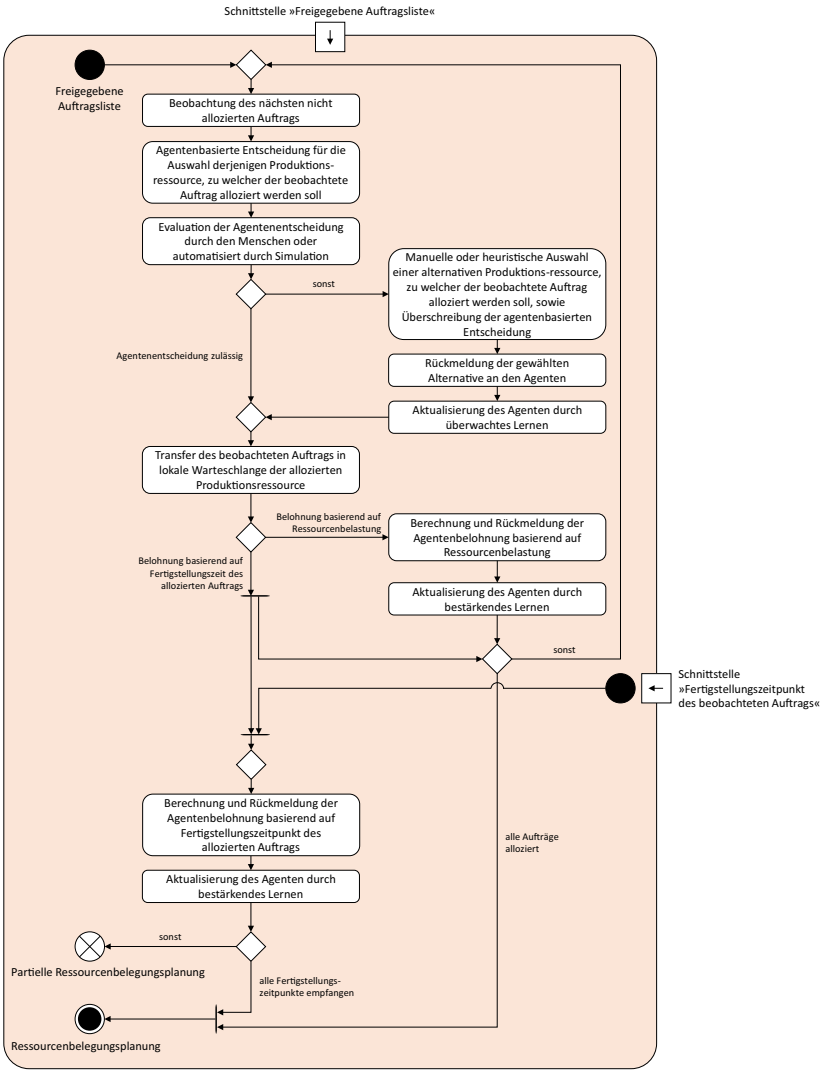


Abbildung 5.4 Prozess der agentenbasierten Ressourcenbelegungsplanung

ebenfalls durch den Agenten vorgeschlagen wurde, oder ob stattdessen eine alternative Ressource ausgewählt wird. Das skizzierte Konzept funktioniert sowohl in Situationen, in denen der Agent qualitativ schlechte Entscheidungen trifft, als auch in Situationen, in welchen der Agent gänzlich unzulässige Entscheidungen fällt. Es dient insbesondere der Erfüllung von Anforderung A4 (Mensch als letzte Planungsinstanz).

Hingegen ist in hochautomatisierten Produktionssystemen, in welchen Ablaufentscheidungen in sekündlichen Intervallen getroffen werden, eine Evaluation durch den Menschen nicht zweckmäßig. Stattdessen kann eine automatisierte Evaluation von Agentenentscheidungen mithilfe von Simulation realisiert werden. Zu diesem Zweck bietet es sich an, das reale Produktionssystem mit einem digitalen Zwilling zu verknüpfen. Ausgehend von der aktuellen Liste freigegebener Aufträge sowie der aktuellen Belegung von Warteschlangen und Produktionsressourcen kann die Lösungsgüte des Agenten anhand von Leistungskennzahlen prognostiziert werden. Die Leistungskennzahlen werden ab dem gegenwärtigen Zeitpunkt über eine bestimmte Produktionsperiode berechnet. Als Leistungskennzahlen eignen sich u. a. die in Abschnitt 2.3.1.3 gelisteten Zielfunktionen. Sofern anhand der Leistungskennzahlen die Agentenentscheidung als unzulässig bewertet wird, kann mithilfe heuristischer Verfahren der Auftrag einer alternativen Produktionsressource in Echtzeit zugeordnet werden.

Durch die Auswahl einer alternativen Produktionsressource entsteht ein Delta zwischen der durch den Agenten empfohlenen und der tatsächlichen Ressourcenbelegungsentscheidung. Dieses Delta wird als Lernsignal verwendet, um mittels überwachten Lernens die Entscheidungspolitik des Agenten zu optimieren. Auf diese Weise wird gewährleistet, dass der zweite Teil von Anforderung A4 (Agent kann von menschlicher Expertise oder aus heuristischen Entscheidungen lernen) durch die Methode berücksichtigt wird. Der beobachtete Auftrag wird nun in die Warteschlange derjenigen Produktionsressource transferiert, die entweder durch den Agenten oder – im Fall einer unzulässigen Agentenentscheidung – manuell bzw. heuristisch zugewiesen wurde.

Ungeachtet der (Un-)Zulässigkeit der getroffenen Agentenentscheidung wird die tatsächliche Allokation des Auftrags zu einer Produktionsressource durch eine Belohnungsfunktion bewertet. Die Belohnungsfunktion dient der Generierung von Lernsignalen für die Durchführung von RL-Trainingsschritten. Falls der Auftrag einer anderen als der durch den Agenten empfohlenen Produktionsressource zugewiesen wurde, dient die Belohnungsfunktion zusätzlich als Metrik, um die Lösungsgüte der durch den Menschen oder durch die Heuristik getroffene Entscheidungsalternative zu bewerten. Der negative Einfluss fehlerhafter Alternativentscheidungen auf das Agententraining kann auf diese Weise

reduziert werden. Das Prozessmodell nimmt an, dass für die Berechnung von Belohnungen entweder Statistiken zur Ressourcenbelastung (z. B. verbleibende Arbeitslast in den Warteschlangen vor den Produktionsressourcen, Gleichmäßigkeit der Verteilung der Arbeitslast, Anzahl wartender Aufträge, u. v. m.) oder fertigstellungszeitbasierte Kennzahlen allozierter Aufträge (z. B. Durchlaufzeit oder Verspätung des Auftrags) zugrunde gelegt werden. Im ersten Fall kann die Belohnung unmittelbar nach der Ressourcenbelegungsentscheidung berechnet und der Agent RL-basiert aktualisiert werden. Im zweiten Fall muss der Agent für unbestimmte Zeit warten, bis der Auftrag im Prozess der agentenbasierten Reihenfolgeplanung und Losbildung ausgewählt und auf der entsprechenden Produktionsressource bearbeitet wurde. Erst dann kann der wahre Fertigstellungszeitpunkt zurückgemeldet und die entsprechende Belohnung berechnet werden. In der Zwischenzeit werden verbleibende freigegebene Aufträge auf die Produktionsressourcen verteilt. Sowohl die Berechnung von Belohnungen als auch die RL-basierte Aktualisierung des Agenten finden somit zeitverzögert und asynchron zur eigentlichen Ressourcenbelegungsplanung statt.

Der Prozess terminiert, sobald alle Aufträge alloziert wurden. Sofern Belohnungen auf Fertigstellungszeitpunkten basieren, gilt die Ermittlung aller Fertigstellungszeitpunkte als zusätzliches Terminationskriterium.

5.2.2 Agentenbasierte Reihenfolgeplanung und Losbildung

Die agentenbasierte Reihenfolgeplanung und Losbildung entscheidet, in welcher Reihenfolge Aufträge in einer Warteschlange auf einer Produktionsressource bearbeitet werden. Sofern die Nebenbedingung existiert, dass Produktionsressourcen für unterschiedliche Auftragsfamilien gerüstet werden müssen, folgt aus der expliziten Auswahl von Aufträgen eine implizite Steuerung der Losbildung. Immer dann, wenn ein Auftrag zur Bearbeitung ausgewählt wird, dessen Familie nicht der aktuellen Rüstung der Produktionsressource entspricht, muss die Ressource umgerüstet werden. Dieses Ereignis führt gleichzeitig dazu, dass das alte Produktionslos abgeschlossen und ein neues Produktionslos gebildet wird. Wird hingegen ein Auftrag gewählt, dessen Familie mit der aktuellen Ressourcenrüstung übereinstimmt, so wird das aktuelle Produktionslos fortgeführt und um einen weiteren Auftrag ergänzt. Abbildung 5.5 illustriert den Prozess der agentenbasierten Reihenfolgeplanung und Losbildung als UML-Aktivitätsdiagramm. Der Prozess wird für jede stationäre Ressource des betrachteten Produktionsbereichs instanziiert. Eine Prozessinstanz wird solange ausgeführt, solange sich Aufträge

in der zugehörigen Warteschlange befinden. Die Warteschlange der Produktionsressource bildet zugleich den Startknoten im Prozess. Der Agent entscheidet zunächst, welcher Auftrag als nächstes auf der Produktionsressource bearbeitet wird. Um zu gewährleisten, dass der angelernte Agent für die Sequenzierung einer beliebigen Anzahl von Aufträgen eingesetzt werden kann (Anforderung A8), werden im Rahmen dieser Arbeit drei unterschiedliche Ansätze zur Auswahl von Aufträgen präsentiert. Zum einen kann die Auswahl direkt erfolgen, indem die Aufträge in der vorgelagerten Warteschlange durch den Agenten immer dann priorisiert werden, sobald eine Produktionsressource verfügbar wird. Darauffolgend wird der Auftrag mit der maximalen Priorität gewählt. In der zweiten Variante werden Aufträge ebenfalls priorisiert, jedoch nur einmalig und zu dem Zeitpunkt, zu welchem sie in den jeweiligen Produktionsbereich eintreten. Der Auftrag wird anschließend gemäß seiner Priorität an der entsprechenden Stelle in der Sequenz eingefügt. In beiden Varianten wird die Sequenzierung von Aufträgen durch einen Agenten mit kontinuierlichem Aktionsraum als Regressionsproblem gelöst. Darüber hinaus kann die Auswahl ebenfalls indirekt erfolgen, indem das eigentliche Steuerungsproblem auf die Auswahl einer reihenfolgebildenden Prioritätsregel reduziert wird. Die agentenbasierte Sequenzierung von Aufträgen wird somit als Klassifikationsproblem mit diskretem Aktionsraum modelliert. Abschnitt 5.3.2.2 widmet sich der detaillierten Erklärung und Umsetzung der drei genannten Ansätze.

Analog zur agentenbasierten Ressourcenbelegungsplanung erfolgt im nächsten Schritt eine Evaluation, ob die vorgeschlagene Agentenentscheidung den Anforderungen der Produktion entspricht. Eine implizite Evaluation durch den Menschen kann realisiert werden, indem sensorisch erfasst wird, ob der durch den Agenten vorgeschlagene Auftrag oder ein alternativer Auftrag ausgewählt wurde. Sofern ein alternativer Auftrag ausgewählt wurde, kann der Unterschied zwischen der agentenbasierten und der manuellen bzw. heuristischen Entscheidungsalternative quantifiziert und als Lernsignal rückwärts durch das Agentenmodell propagiert werden. Die Art und Weise der Quantifizierung hängt davon ab, ob die Auftragsauswahl durch Berechnung von Prioritäten oder durch Bestimmung von Prioritätsregeln gesteuert wird. Im ersten Fall entspricht das Lernsignal der Differenz zwischen der maximal möglichen Priorität und der Priorität des alternativ gewählten Auftrags. Wenn die agentenbasierte Reihenfolgebildung mithilfe von Prioritätsregeln gesteuert wird, muss zunächst ermittelt werden, welche Prioritätsregel am besten den alternativ gewählten Auftrag widerspiegelt. Für die Berechnung des Lernsignals wird darauffolgend ein alternativer Ausgabevektor konstruiert, der auf alle auswählbaren Prioritätsregeln abbildet. Jedes Element des alternativen Ausgabevektors besitzt einen Wert gleich null, ausgenommen

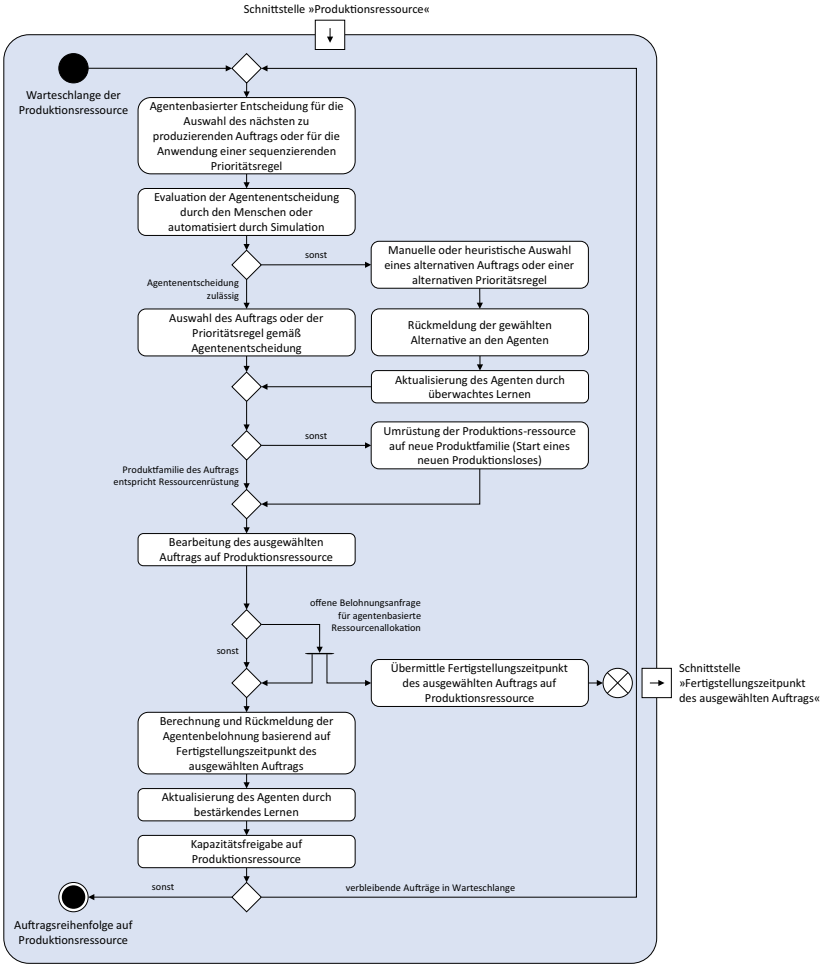


Abbildung 5.5 Prozess der agentenbasierten Reihenfolgeplanung und Losbildung

des Elements, welches die Prioritätsregel des alternativ gewählten Auftrags widerspiegelt und demzufolge einen Wert von eins besitzt. Um auf die Prioritätsregel des alternativen Auftrags zu schließen, können diverse Auftragseigenschaften herangezogen werden, z. B. dessen Fertigstellungsfrist, Produktfamilie oder Bearbeitungszeiten. Von der Betrachtung ausgenommen ist diejenige Eigenschaft, die

von der durch den Agenten ausgewählten Prioritätsregel herangezogen wurde. Hierdurch wird gewährleistet, dass eine andere Prioritätsregel identifiziert wird als diejenige, die durch den Agenten ausgewählt wurde. Ferner wird die am stärksten ausgeprägte Eigenschaft für die Bestimmung einer alternativen Prioritätsregel herangezogen. Der Grad der Ausprägung einer Auftragseigenschaft kann bspw. durch einen Vergleich mit den anderen Aufträgen in der Warteschlange beurteilt werden.

Zum besseren Verständnis wird der Ansatz am folgenden Beispiel erläutert. Das betrachtete System ist ein Ein-Maschinen-Problem zum Zeitpunkt $t = 0$, in welchem die Bearbeitungsreihenfolge durch die agentenbasierte Auswahl von Prioritätsregeln gesteuert wird. Der Agent kann zwischen den Regeln [SPT, LPT, EDD, LST] entscheiden (siehe Abschnitt 2.3.3.2 zu deren Bedeutung und Funktionsweise), wobei die Elemente des Ausgabevektors den vier Prioritätsregeln in derselben Reihenfolge zugeordnet sind. In $t = 0$ gibt der Agent den Vektor [0,2 , 0,3 , 0,4 , 0,1] aus. Da das Vektorelement mit dem Wert $0,4$ das Maximum darstellt, entscheidet sich der Agent für die Anwendung der EDD-Prioritätsregel. Hieraus folgt, dass die Bearbeitung des Auftrags mit der geringsten Fertigstellungsfrist empfohlen wird. Dieser Auftrag wird folgend als $j = 1$ bezeichnet. Der Auftrag $j = 1$ besitzt eine Fertigstellungsfrist von $d_1 = 20$ und eine Bearbeitungszeit von $o_1 = 3$. Entgegen der Agentenentscheidung wird jedoch Auftrag $j = 2$ als nächstes für die Bearbeitung ausgewählt, der eine Fertigstellungsfrist von $d_2 = 21$ und eine Bearbeitungszeit von $o_2 = 17$ besitzt. Für die Identifikation einer alternativen Prioritätsregel scheidet alleinig die Fertigstellungsfrist d als Eigenschaft aus, da der ursprüngliche Auftrag auf Basis der EDD-Prioritätsregel gewählt wurde. Die zu vergleichenden Eigenschaften sind die Bearbeitungszeiten o_1 und o_2, die von den Prioritätsregeln SPT und LPT für die Sequenzierung von Aufträgen als Vergleichskriterium herangezogen werden, sowie die Schlupfzeiten $(d_1 - o_1)$ und $(d_2 - o_2)$, die von der LST-Prioritätsregel für die Reihenfolgebildung zugrunde gelegt werden. Es ist auszuschließen, dass sich für den alternativ ausgewählten Auftrag $j = 2$ aufgrund der SPT-Prioritätsregel entschieden wurde, da $j = 2$ eine geringere Bearbeitungszeit als der alternativ ausgewählte Auftrag $j = 1$ besitzt. Als alternative Prioritätsregeln verbleiben somit nur noch LPT und LST. Im nächsten Schritt werden die Bearbeitungszeit und die Schlupfzeit des alternativ gewählten Auftrags mit allen Aufträgen in der vorgelagerten Warteschlange verglichen. Hierbei stellt sich heraus, dass der alternativ ausgewählte Auftrag die drittlängste Bearbeitungszeit und die zweitkürzeste Schlupfzeit unter allen wartenden Aufträgen besitzt. Da somit die Schlupfzeit des alternativ ausgewählten Auftrags stärker

ausgeprägt ist als dessen Bearbeitungszeit, wird die Prioritätsregel LST verwendet, um den Unterschied zwischen der agentenbasierten und der manuell bzw. heuristisch ausgewählten Entscheidungsalternative zu quantifizieren. Der alternative Ausgabevektor lautet somit [0, 0, 0, 1]. Das Lernsignal, welches rückwärts durch das Agentenmodell propagiert wird, resultiert aus der Differenz zwischen dem Ausgabevektor des Agenten [0,2 , 0,3 , 0,4 , 0,1] und dem alternativen Ausgabevektor [0, 0, 0, 1].

Das Konzept zum Einsatz von überwachtem Lernen für die Berücksichtigung von menschlicher Expertise im Agententraining verhält sich darüber hinaus analog zur agentenbasierten Ressourcenbelegungsplanung. Der agentenbasierte bzw. manuell oder heuristisch ausgewählte Auftrag wird nun auf die Produktionsressource umgelagert. Vor dem Start der Bearbeitung des Auftrags wird überprüft, ob die Produktionsressource für die Bearbeitung die richtige Rüstung besitzt. Zu diesem Zweck wird die Familie des Auftrags mit dem Rüsttypen der Produktionsressource verglichen. Falls Produktfamilie und Rüstung voneinander abweichen, wird die Produktionsressource umgerüstet. Damit einhergehend gilt das aktuelle Produktionslos als abgeschlossen, während der soeben gewählte Auftrag den Start eines neuen Produktionsloses markiert.

Nach der Bearbeitung des Auftrags wird eine Belohnung für den Agenten berechnet. Diese basiert stets auf dem Fertigstellungszeitpunkt des Auftrags. Die Ermittlung und Rückmeldung der Belohnung erfolgt unabhängig davon, ob der Auftrag durch den Agenten oder manuell bzw. heuristisch ausgewählt wurde. Analog zur agentenbasierten Ressourcenbelegungsplanung sollen auf diese Weise sowohl agentenbasierte als auch menschliche Entscheidungen belohnt und bestraft werden können. Darüber hinaus wird der Fertigstellungszeitpunkt des zuletzt gewählten Auftrags an den Prozess »Agentenbasierte Ressourcenbelegungsplanung« zurückgemeldet, sofern dieser zuvor initiiert wurde und dessen Belohnungssystem auf Fertigstellungszeitpunkten von Aufträgen basiert.

In den letzten beiden Prozessschritten wird der Agent für die Reihenfolgeplanung und Losbildung auf Basis der berechneten Belohnung aktualisiert und die Bearbeitungskapazität der betrachteten Produktionsressource freigegeben. Der Prozess terminiert, sobald sich keine weiteren Aufträge in der vorgelagerten Warteschlange befinden. Andernfalls durchläuft der Prozess eine weitere Iteration, sodass der Agent den nächsten Auftrag zur Bearbeitung auf der betrachteten Produktionsressource vorschlägt.

5.3 Projektierung und Entwicklung von agentenbasierten Produktionsablaufsteuerungen

Nachdem im vorhergehenden Abschnitt der Prozess und das Funktionsprinzip der agentenbasierten Produktionsablaufsteuerung hergeleitet wurde, soll im Folgenden ein Vorgehensmodell beschrieben werden, welches die Projektierung und Entwicklung von agentenbasierten Produktionsablaufsteuerungen anleitet. Wie Abbildung 5.6 zeigt, beinhaltet das Vorgehensmodell sieben Schritte. Ausgangspunkt ist ein beliebiges Problem der Produktionsablaufplanung. Das Vorgehensmodell mündet in der Realisierung einer agentenbasierten Produktionsablaufsteuerung. Im weiteren Verlauf sollen die einzelnen Schritten des Vorgehensmodells im Detail dargelegt werden.

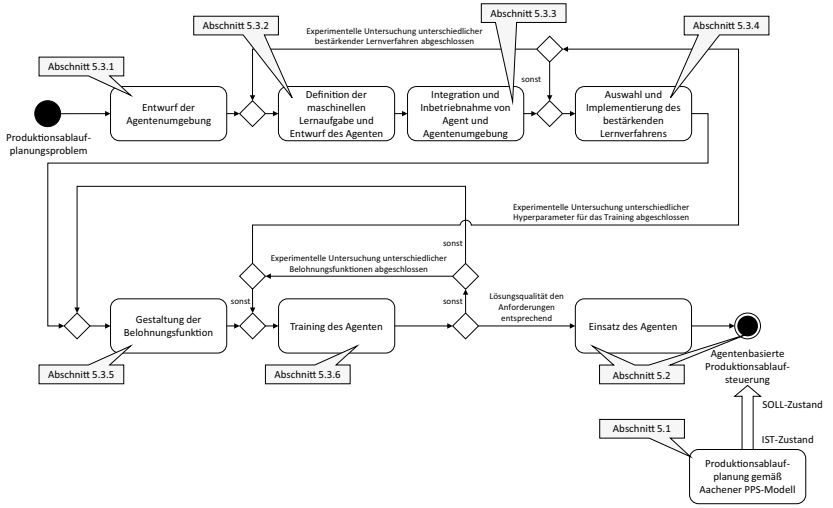


Abbildung 5.6 Vorgehensmodell zur Projektierung und Entwicklung von agentenbasierten Produktionsablaufsteuerungen

5.3.1 Entwurf von Agentenumgebungen

Für den Entwurf von Agentenumgebungen ist eine Methode zu wählen, welche die Modellierung von Problemen der Produktionsablaufplanung als MEP erlaubt (siehe Abschnitt 3.3.1). Um eine datengetriebene agentenbasierte Produktionsablaufplanung zu realisieren, muss die Methode ferner beliebig viele und komplexe Informationen in einem Modell berücksichtigen können. Vor diesem Hintergrund eignen sich Modellierungsmethoden, die Produktionsprozesse gemäß ihres Ablaufs, ihrer Struktur, ihrer Informationen und ihrer Dynamik detailgetreu abbilden. Die DES-Methode, deren Grundlagen bereits in Abschnitt 2.3.2.2 dargelegt wurden, erfüllt die genannten Anforderungen und wird in dieser Arbeit für den Entwurf von Agentenumgebungen empfohlen.

Das Ziel dieses Abschnitts ist eine umfassende und generische Darstellung, wie Probleme der Produktionsablaufplanung mithilfe der DES-Methode als Agentenumgebung modelliert werden können. Wie bereits in den theoretischen Grundlagen diskutiert wurde, existieren sowohl grafische als auch sprachliche Beschreibungsformen für DES-Modelle. Für die Erklärung der Modellierung von Agentenumgebungen werden im Folgenden sprachliche Beschreibungsformen zugrunde gelegt, da diese eine algorithmische Darstellung über wiederum grafische Programmablaufpläne begünstigen. Darüber hinaus entsprechen sprachliche Beschreibungsformen einer rein Python-basierten Implementierung von DES-Modellen. Die Implementierung von DES-Modellen in Python ist vorteilhaft, weil die beiden weitverbreitetsten Bibliotheken für Deep Learning »TensorFlow« (Abadi et al. 2016) und »PyTorch« (Brockman et al. 2016) ebenfalls auf Python als Modellierungssprache setzen. Eine rein Python-basierte Implementierung vereinfacht sowohl die Integration von ML- und DES-Modellen als auch den Datenaustausch zwischen den Modellen. Ferner und im Unterschied zu kommerziellen DES-Softwarepaketen sind die beiden bekanntesten Python-Bibliotheken zur Implementierung von DES-Modellen »SimPy« (Matloff 2008) und »Salabim« (van der Ham 2018) kostenfrei verfügbar, quelloffen und somit niedrigschwellig zugänglich.

Konkret sind die folgenden Programmablaufpläne an die Methodik zur Beschreibung von Prozessabläufen in der Modellierungssprache »SIMULA« (Dahl und Nygaard 1966) angelehnt, an der sich ebenfalls der Modellierungsprozess von Salabim orientiert. In SIMULA wird das Verhalten eines jeden Flussobjekts, Quellobjekts (für die Erzeugung von Flussobjekten) und Ressourcenobjekts (für die Bearbeitung von Flussobjekten) durch eine jeweils eigene Prozesslogik beschrieben. Hierdurch wird eine dezentrale agentenbasierte

Modellierung begünstigt. Ferner unterstützt SIMULA das zeitbasierte und ereignisbasierte Pausieren von Prozessen. Beim zeitbasierten Pausieren wird eine Ablauflogik für eine definierte Zeitspanne unterbrochen. Hingegen wird beim ereignisbasierten Pausieren eine Ablauflogik für eine unbestimmte Zeitspanne unterbrochen. Der Prozess wird erst dann fortgesetzt, wenn ein bestimmtes Ereignis eintritt, welches durch eine andere Prozesslogik ausgelöst wird. Abbildung 5.7 präsentiert ein umfassendes, generisches Prozessmodell für die Implementierung von Problemen der Produktionsablaufplanung als Agentenumgebung. Das Modell setzt sich aus der Prozesslogik zur Erzeugung von Aufträgen, der Prozesslogik von Aufträgen und der Prozesslogik von Produktionsressourcen zusammen. In den folgenden Abschnitten werden die drei Unterprozesse im Detail beschrieben.

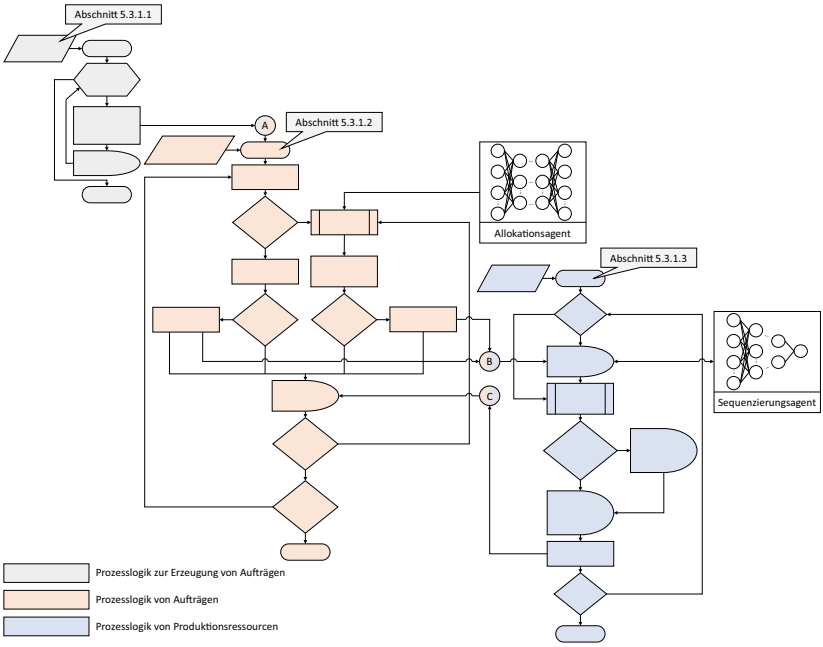


Abbildung 5.7 Aufbau, Struktur und Komponenten des Prozessmodells für die Implementierung von Problemen der Produktionsablaufplanung als Agentenumgebung

5.3.1.1 Modellierung der Erzeugung von Aufträgen

Zunächst wird betrachtet, wie die Erzeugung von Aufträgen als Programmablaufplan formuliert werden kann. In Abhängigkeit von der Charakteristik des betrachteten Produktionssystems existieren zwei verschiedene Varianten zur Modellierung von Auftragserzeugungsprozessen. Beide Varianten werden in Abbildung 5.8 dargestellt. Die Ablauflogik in Abbildung 5.8 (a) geht von einer im Voraus bekannten Liste mit freigegebenen Aufträgen aus. Da die agentenbasierte Produktionsablaufsteuerung (Abbildung 5.2 in Abschnitt 5.2) ebenfalls von einer Liste mit freigegebenen Aufträgen ausgeht, repräsentiert Variante (a) den Standardablauf zur Modellierung von Auftragserzeugungsprozessen. Vor diesem Hintergrund ist Variante (a) ebenfalls stellvertretend in der Gesamtübersicht des Prozessmodells in Abbildung 5.7 dargestellt. Es wird angenommen, dass jedem Auftrag in der Liste freigegebener Aufträge eine Menge von Attributen zugeordnet ist, die zum einen für den Simulationsablauf, zum anderen für die Zustandsbildung für agentenbasierte Produktionsablaufentscheidungen von Bedeutung sind. Typische Auftragsattribute werden in Abschnitt 5.3.1.2 gelistet. Über eine for-Schleife werden alle Aufträge der freigegebenen Liste mit den entsprechenden Attributen initialisiert. Darauffolgend wird die Prozesslogik für null Zeiteinheiten pausiert. Das Pausieren hat keine Bedeutung für die Simulation des Produktionsablaufs, sondern ist als technischer Kniff zu verstehen, der einen korrekten Ablauf des DES-Modells erlaubt. Durch das Pausieren wird ein zusätzliches Ereignis im Ereignisverwalter gesetzt, wodurch zunächst einige Schritte in der Prozesslogik des erstellten Auftrags abgearbeitet werden, bevor die Quelle den nächsten Auftrag produzieren kann. Hierdurch wird gewährleistet, dass der Auftrag die Quelle verlässt und sich in der Warteschlange einer Produktionsressource einreiht, bevor der nächste Auftrag erstellt wird und die Quelle temporär blockiert. Die Auftragserzeugung in Abbildung 5.8 (a) eignet sich insbesondere, um eine initiale Ressourcen- und Warteschlangenbelegung für ein leeres Produktionssystem zu erzeugen.

Abbildung 5.8 (b) zeigt ein alternatives Prozessmodell zur Erzeugung von Aufträgen. Der Prozess eignet für die Modellierung eines dynamischen stochastischen Auftragshorizonts, bei welchem weder die genauen Aufträge, noch ihre genauen Ankunftszeitpunkte bekannt sind. Es wird jedoch angenommen, dass alle Aufträge die gleiche Art und Anzahl von Attributen besitzen und nur die Werte der jeweiligen Attribute unterschiedlich sind. Ferner wird angenommen, dass für die zufälligen Werte von Attributen und Zwischenankunftszeiten Wahrscheinlichkeitsverteilungen bekannt sind. Auf Basis dieser Wahrscheinlichkeitsverteilungen generiert das Prozessmodell in Abbildung 5.8 (b) Aufträge in zufälligen Abständen mit zufällig initialisierten Attributen. Die Generierung von Aufträgen wird

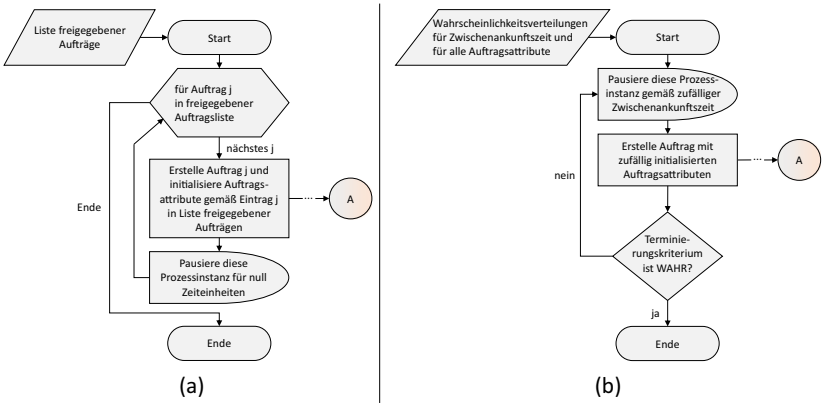


Abbildung 5.8 Prozess der Erzeugung von Aufträgen als Programmablaufplan unter der Annahme (a) eines (temporär) finiten Auftragshorizonts bzw. (b) eines dynamischen stochastischen Auftragshorizonts

über eine while-Schleife gesteuert, die nach jedem Auftrag ein bestimmtes Terminationskriterium prüft, bspw. das Überschreiten einer maximalen Anzahl von generierten Aufträgen, eines bestimmten Zeitpunkts, einer bestimmten Systemlast u. v. m. Sobald das Terminationskriterium erfüllt ist, werden keine weiteren Aufträge generiert.

Ungeachtet dessen, welches Prozessmodell für die Erzeugung von Aufträgen Verwendung findet, wird durch die Initialisierung eines Auftrags eine Instanz der Auftragsprozesslogik erstellt und ausgeführt (Schnittstelle »A«). Der Aufbau und Ablauf der Auftragsprozesslogik soll im Folgenden behandelt werden.

5.3.1.2 Modellierung von Aufträgen

Jeder Auftrag, der durch ein Quellobjekt generiert wird, durchläuft eine eigene Instanz einer vordefinierten Prozesslogik. Abbildung 5.9 demonstriert die Prozesslogik für den Fluss von Aufträgen. Der Prozesslogik beschreibt alle Möglichkeiten, wie sich ein Auftrag durch das System bewegen kann, um vollständig bearbeitet zu werden. Der genaue Weg eines Auftrags durch das System, respektive welche Operationen des Auftrags durch welche Produktionsressourcen bearbeitet werden, ist von den jeweiligen Ressourcenbelegungsentscheidungen abhängig. Vor diesem Hintergrund wird an das zu formulierende Auftragsprozessmodell der Anspruch gestellt, dass dieses ausreichend generisch ist, um für

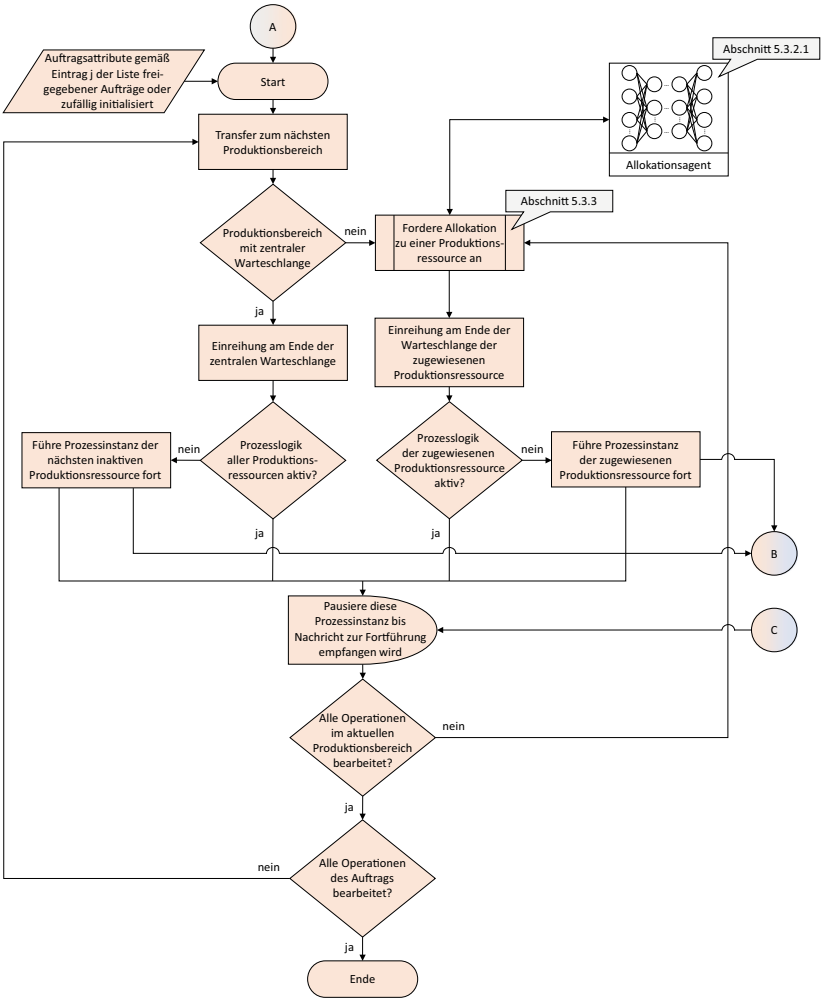


Abbildung 5.9 Prozess des Auftragsflusses durch Produktionssysteme als Programmablaufplan

eine Vielzahl von Produktionssystemen gültig zu sein. Die Prozesslogik in Abbildung 5.9 erfüllt diese Anforderung, indem eine notwendige Annahme getroffen wird. Diese fordert, dass das betrachtete Produktionssystem in einzelne Produktionsbereiche gemäß Abbildung 5.3 (siehe Abschnitt 5.2) untergliedert werden kann. Darüber hinaus werden zwei vereinfachende Annahmen getroffen, mit der Zielstellung den Untersuchungsbereich der vorliegenden Arbeit zu verfeinern und die Komplexität des Prozessmodells in Abbildung 5.9 zu reduzieren. Die erste vereinfachende Annahme lautet, dass jeder Auftrag alle Produktionsbereiche in einer festen Reihenfolge durchläuft. Die zweite vereinfachende Annahme besagt, dass alle Operationen eines Auftrags in fester Reihenfolge bearbeitet werden. Die Prozesslogik in Abbildung 5.9 eignet sich somit zur Modellierung von allen Problemen der Produktionsablaufplanung, die in Abschnitt 2.3.1.1 gelistet sind. Eine Ausnahme bilden Open-Shop-Probleme, in welchen die Operationen eines Auftrags in beliebiger Reihenfolge durchlaufen werden können. Aus konzeptioneller Sicht kann die vorgestellte Prozesslogik jedoch auf einfache Weise erweitert werden, um ebenfalls Open-Shop-Probleme als Agentenumgebung zu implementieren. Die notwendigen Erweiterungen werden am Ende dieses Abschnitts kurz skizziert. Zunächst soll jedoch auf das Prozessmodell in Abbildung 5.9 genauer eingegangen werden.

Nachdem ein Auftrag mit vordefinierten oder zufälligen Attributwerten initialisiert wurde (Startknoten), gelangt dieser in den nächsten Produktionsbereich. Tabelle 5.1 listet die wichtigsten Attribute von Aufträgen, die für die Modellierung, Simulation und Optimierung von Problemen der Produktionsablaufplanung relevant sind. Ferner sind die gelisteten Attribute maßgeblich für die Beschreibung von Umgebungszuständen, die ein Agent für die Berechnung von Aktionen analysiert. Die Verarbeitung von Umgebungszuständen zu Produktionsablaufentscheidungen ist Gegenstand von Abschnitt 5.3.2. Sofern der Produktionsbereich eine zentrale Warteschlange besitzt (unterer Ausgang der ersten Verzweigung), reiht sich der Auftrag an deren hinterem Ende ein. Zum Eintrittszeitpunkt des Auftrags kann die Situation auftreten, dass nicht alle Produktionsressourcen belegt sind. Diejenigen Produktionsressourcen, welche keine Aufträge bearbeiten, pausieren ihre Prozesslogik auf unbestimmte Zeit und werden nur durch die Ankunft von neuen Aufträgen reaktiviert. Aktive und inaktive Produktionsressourcen werden in unterschiedlichen Listen gespeichert. Falls Produktionsressourcen bei Warteschlangeneintritt inaktiv sind (linker Ausgang der mittleren linken Verzweigung), sendet die Prozesslogik des Auftrags ein Signal an die erstgelistete inaktive Produktionsressource. Hierdurch wird die Prozesslogik der Produktionsressource fortgeführt und der eingetroffene Auftrag aus der Warteschlange

bearbeitet (Schnittstelle »B«). Sofern jedoch die Ressourcen des Produktionsbereichs durch lokale Warteschlangen gespeist werden (rechter Ausgang der ersten Verzweigung), entscheidet zunächst ein sogenannter Allokationsagent, zu welcher Produktionsressource der Auftrag zugewiesen werden soll. Die Art und Weise der Entscheidungsfindung von Allokationsagenten wird in Abschnitt 5.3.2.1 erläutert. Die Berücksichtigung von Agentenentscheidungen im Simulationsmodell wird in Abschnitt 5.3.3.1 für gradientenabhängige bzw. in Abschnitt 5.3.3.2 für gradientenfreie RL-Verfahren gesondert behandelt. Der Auftrag reiht sich daraufhin am hinteren Ende der Warteschlange der zugewiesenen Produktionsressource ein. Wenn die Produktionsressource zum Eintrittszeitpunkt keinen Auftrag bearbeitet und sich keine weiteren Aufträge in der Warteschlange befinden, kann geschlussfolgert werden, dass die Produktionsressource ihre Prozesslogik gegenwärtig pausiert (rechter Ausgang der mittleren rechten Verzweigung). In diesem Fall sendet der Auftrag ein Signal zur Fortführung der Prozesslogik, wodurch dieser von der Produktionsressource zur Bearbeitung aufgenommen wird (Schnittstelle »B«).

Ungeachtet der Art des Produktionsbereichs pausiert der Auftrag seine Prozesslogik auf unbestimmte Zeit, nachdem dieser eine Warteschlange betritt und u. U. die Prozesslogik einer Produktionsressource reaktiviert. Die Prozesslogik des Auftrags wird wiederum durch eine Produktionsressource reaktiviert, sobald diese die aktuelle Operation am Auftrag bearbeitet hat (Schnittstelle »C«). Wenn weitere Operationen im aktuellen Produktionsbereich bearbeitet werden müssen (rechter Ausgang der vorletzten Verzweigung), wird erneut der Allokationsagent für eine Ressourcenbelegungsentscheidung konsultiert. Sofern im aktuellen Produktionsbereich alle möglichen Operationen durchgeführt wurden (unterer Ausgang der vorletzten Verzweigung), wird geprüft, ob der Auftrag noch weitere Operationen durchlaufen muss. Falls dem so ist (linker Ausgang der letzten Verzweigung), wird der Auftrag in den nächsten Produktionsbereich transferiert. Falls alle Operationen komplettiert wurden (unterer Ausgang der letzten Verzweigung), verlässt der Auftrag die Umgebung und wird in der fortlaufenden Simulation nicht mehr berücksichtigt.

Wie bereits erwähnt kann die geschilderte Logik um einige wenige Elemente erweitert werden, sodass ebenfalls die Modellierung von Open-Shop-Problemen möglich ist. Unter der Annahme, dass eine rein agentenbasierte Produktionsablaufsteuerung realisiert werden soll, sind darüber hinaus zwei weitere Arten von Agenten erforderlich. Der erste zusätzliche Agent entscheidet zunächst, welche Operation des betrachteten Auftrags als nächstes bearbeitet werden soll. Der zweite zusätzliche Agent entscheidet danach, in welchem Produktionsbereich die

Tabelle 5.1 Geläufige Attribute für die Modellierung von Aufträgen

Attribut	Datentyp	Datenstruktur	Beschreibung
Freigabezeit	Ganzzahlig oder Gleitkommazahl	Variable	Zeitpunkt der Erzeugung des Auftrags durch Quellobjekt. Relevant für Produktionsablaufplanung mit dynamischen Auftragshorizont.
Operationen	Ganzzahlig	Liste oder Tabelle	Anzahl und Art von Operationen, die an einem Auftrag auszuführen sind. Nur relevant für Job-Shop- und Open-Shop-Probleme. Bei Job-Shop-Problemen entspricht die Reihenfolge der Listenelemente der Reihenfolge der auszuführenden Operationen. Werden die Operationen als Tabelle dargestellt, existieren verschiedene zulässige technologische Reihenfolgen (Zeilen).
Bearbeitungszeit(en)	Ganzzahlig oder Gleitkommazahl	Variable, Liste oder Tabelle	Variable entspricht einzelner Bearbeitungszeit, Liste entspricht Bearbeitungszeiten auf unterschiedlichen Ressourcen auf einer Stufe oder von (gleichen) Ressourcen auf unterschiedlichen Stufen, Tabelle zur Darstellung von Bearbeitungszeiten von unterschiedlichen Operationen (Zeilen) auf unterschiedlichen Ressourcen (Spalten).

(Fortsetzung)

Tabelle 5.1 (Fortsetzung)

Attribut	Datentyp	Datenstruktur	Beschreibung
Fertigstellungsfrist	Ganzzahlig oder Gleitkommazahl	Variable	Relevant, sofern Verspätungskriterien (T, U, L_{max}) in der zu optimierenden Zielfunktion berücksichtigt werden.
Fertigstellungsgrad	Gleitkommazahl	Variable	Anzahl bearbeiteter Operationen im Verhältnis zur Gesamtzahl von Operationen. Nur relevant für Job-Shop- und Open-Shop-Probleme. Wird stets mit null initialisiert.
Familie	Ganzzahlig	Variable	Legt die erforderliche Rüstung von Ressourcen fest und initiiert Umrüstprozesse.

gewählte Operation bearbeitet werden soll. Im Folgenden würde die Prozesslogik gemäß Abbildung 5.9 durchlaufen werden. Sofern nach der Fertigstellung der gewählten Operation noch weitere Operationen offen sind, erfolgt die Auswahl der nächsten zu bearbeitenden Operation. Bei der daran anschließenden Auswahl des nächsten Produktionsbereichs besteht die Möglichkeit den aktuellen Produktionsbereich beizubehalten.

5.3.1.3 Modellierung von Produktionsressourcen

Ein weiterer entscheidender Aspekt hinsichtlich des Entwurfs von Agentenumgebungen ist die Modellierung von Produktionsressourcen. Abbildung 5.10 präsentiert eine generische Programmlogik für den Prozessablauf von Produktionsressourcen. Im Folgenden soll das Prozessmodell im Detail erläutert werden. Zudem wird anhand von Beispielen aufgezeigt, wie das Prozessmodell erweitert werden kann, um spezifische Restriktionen von unterschiedlichen Problemen der Produktionsablaufplanung zu berücksichtigen.

Die Produktionsressourcen und ihre Prozessmodelle werden im Zuge der Umgebungsinitialisierung instanziiert. Im Rahmen der Instanziierung werden ebenfalls die Attribute der Produktionsressourcen initialisiert. Tabelle 5.2 enthält einige der geläufigsten Attribute für die Modellierung von Produktionsressourcen.

Analog zu Tabelle 5.1 dienen viele der gelisteten Attribute ebenfalls der Beschreibung von Umgebungszuständen, welche ein Agent für die Ausgabe von Aktionen vorwärtspropagiert. Einhergehend mit ihrer Instanziierung wird die Prozesslogik einer Produktionsressource automatisch gestartet. Zunächst wird abgefragt, ob die vorgelagerte Warteschlange der Produktionsressource leer ist. Zu Beginn einer Episode ist dies stets zutreffend (unterer Ausgang der ersten Verzweigung), da die Prozesslogik von jeder Produktionsressource ausgeführt wird, bevor der erste Auftrag durch eine Quelle produziert wird. Demzufolge wird die Prozesslogik pausiert, bis sich ein Auftrag in die vorgelagerte Warteschlange einordnet (Schnittstelle »B«). In fortschreitenden Zyklen der Prozesslogik befinden sich gewöhnlich mehrere Aufträge in der vorgelagerten Warteschlange, wodurch das Pausieren übersprungen und mit dem nachfolgenden Schritt fortgefahren wird (linker Ausgang der ersten Verzweigung).

Im Folgenden wird ein sogenannter Sequenzierungsagent für die Auswahl des nächsten zu bearbeitenden Auftrags konsultiert. Die Auswahl des Auftrags erfolgt entweder direkt durch die Priorisierung von Aufträgen oder indirekt durch die Auswahl einer Prioritätsregel. Beide Formulierungsvarianten werden in Abschnitt 5.3.2.2 detailliert beschrieben. Analog zur Allokation von Aufträgen wird für gradientenabhängige bzw. -freie RL-Verfahren die Berücksichtigung von agentenbasierten Sequenzierungsentscheidungen im Simulationsmodell in Abschnitt 5.3.3.1 bzw. 5.3.3.2 gesondert behandelt. Die darauffolgende Verzweigung dient als Beispiel für die Modellierung von problemspezifischen Nebenbedingungen. Es wird geprüft, ob die Familie des ausgewählten Auftrags mit der aktuellen Rüstung der Produktionsressource übereinstimmt. Ist dies nicht der Fall (linker Ausgang der zweiten Verzweigung), muss die Prozesslogik gemäß einer bestimmten Rüstzeit pausiert werden. Die Rüstzeit kann entweder konstant oder reihenfolgeabhängig sein. Im zweiten Fall wird die Rüstzeit aus einer Tabelle ausgelesen, die alle Rüstzeiten zwischen gerüsteten (Zeilen) und zu rüstenden Familien (Spalten) enthält. Auf ähnliche Art und Weise können weitere Restriktionen der Produktionsablaufplanung modelliert werden.

Die Modellierung von zusätzlichen mobilen Ressourcen (z. B. Rüst-Kits, Servicekräfte u. v. m.) wäre in der Prozesslogik von Produktionsressourcen noch einfacher zu berücksichtigen. Für jede angeforderte mobile Ressource muss lediglich die Prozesslogik der Produktionsressource für unbestimmte Zeit pausiert werden. Im Gegensatz zur Modellierung von Rüstprozessen ist eine zusätzlich vorgelagerte Verzweigung nicht notwendig. Allerdings erfordert jede mobile Ressourcenart die Definition einer eigenen Prozesslogik, die ihr Verhalten beschreibt. Die Modellierung von mobilen Ressourcen wird in dieser Arbeit nicht explizit betrachtet, folgt aber sehr ähnlichen Prinzipien wie die Prozessmodelle von

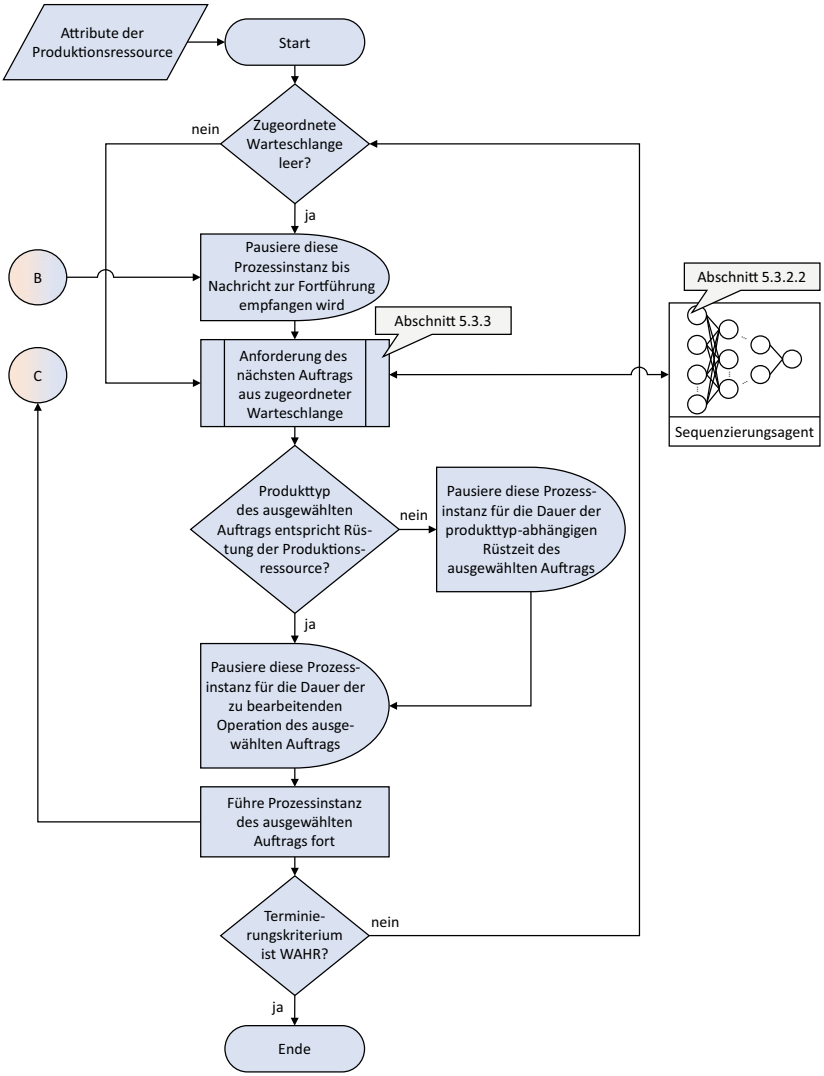


Abbildung 5.10 Prozess von Produktionsressourcen als Programmablaufplan

Tabelle 5.2 Geläufige Attribute für die Modellierung von Produktionsressourcen

Attribut	Datentyp	Datenstruktur	Beschreibung
Status	Ganzzahlig	Variable	Relevant für die Modellierung von Umgebungszuständen. Verschiedene Ressourcenzustände (z. B. »Leerlauf«, »bearbeitend«, »rüstend«, »auf mobile Ressource wartend«, »ausgefallen« u. v. m.) können binär oder ganzzahlig kodiert werden.
Warteschlange	Auftrag	Liste	Warteschlange, welche die Produktionsressource mit Aufträgen speist. Warteschlangen sind einfache Listen, welche Objekte des Typs »Auftrag« enthalten.
Arbeitslast	Ganzzahlig oder Gleitkommazahl	Variable	Bearbeitungszeiten und (falls ermittelbar) Nebenzeiten (z. B. geplante Rüstzeiten, geplante Wartezeiten für mobile Ressourcen, geplante Ausfall- und Wartungszeiten) kumuliert über den aktuell in Bearbeitung befindlichen Auftrag und alle Aufträge der vorgelagerten Warteschlange.
Rüstung	Ganzzahlig	Variable	Aktuelle Rüstung der Produktionsressource in Abhängigkeit von der Familie des derzeit in Bearbeitung befindlichen Auftrags.

(Fortsetzung)

Tabelle 5.2 (Fortsetzung)

Attribut	Datentyp	Datenstruktur	Beschreibung
Rüstmatrix	Ganzzahlig oder Gleitkommazahl	Tabelle	Für die Modellierung von reihenfolgeabhängigen Rüstzeiten. Die Tabelleneinträge geben die Rüstzeit zwischen zwei aufeinanderfolgenden Produkten oder Produktfamilien an.
Verfügbarkeit	Wahrscheinlichkeitsverteilung, Ganzzahlig oder Gleitkommazahl	Objekt oder Liste	Gibt Rückschluss auf zufällige (Wahrscheinlichkeitsverteilung als Objekt) und geplante Ausfallzeiten (Liste mit Ausfallzeiten) der Produktionsressourcen.
Bearbeitungs-geschwindigkeitsfaktor	Gleitkommazahl	Variable	Gibt Rückschluss auf die Bearbeitungsgeschwindigkeit von Operationen. Wird mit der Bearbeitungszeit von Operationen multipliziert (Faktor $= 1$ $\rightarrow$ entspricht der Bearbeitungszeit, Faktor $< 1 \rightarrow$ kürzere Bearbeitungszeit, Faktor $> 1 \rightarrow$ höhere Bearbeitungszeit).

Aufträgen und Produktionsressourcen. Im Folgenden soll lediglich ein grobes verallgemeinerndes Prozessmodell für mobile Ressourcen umrissen werden, um das Prinzip zu verdeutlichen: Anfragen an mobile Ressourcen werden in einer separaten Warteschlange gepuffert. Die Anfragen werden nach einer bestimmten Sequenzierungslogik (z. B. Prioritätsregeln) oder agentenbasiert (gemäß den Entscheidungen eines zusätzlichen Sequenzierungsagenten) abgearbeitet. Für die Bearbeitung der Anfragen können zusätzliche Zeiten anfallen, z. B. für den Transport der mobilen Ressourcen und für weitere Hilfsoperationen. Sobald eine mobile Ressource ihren Bestimmungsort erreicht hat, sendet sie ein Signal an die Produktionsressource, sodass diese ihre Prozesslogik fortsetzt. Währenddessen wird die Prozesslogik der mobilen Ressource pausiert, bis die bediente Produktionsressource zurückmeldet, dass alle Operationen, für welche die mobile Ressource angefordert wurde, abgeschlossen sind und diese nicht mehr benötigt wird. Die Rückmeldung reaktiviert die Prozesslogik der mobilen Ressource. Sofern weitere Anfragen offen sind, werden diese abgearbeitet. Andernfalls wird die Prozesslogik der mobilen Ressource ein weiteres Mal auf unbestimmte Zeit pausiert, bis sich neue Anfragen in die entsprechende Warteschlange einreihen. Nach diesem kurzen Exkurs zur Modellierung von zusätzlichen Planungsrestriktionen sowie des Verhaltens von mobilen Ressourcen, soll nun mit der Beschreibung der Prozesslogik in Abbildung 5.10 fortgefahren werden.

Die Bearbeitung von Auftragsoperationen wird durch zeitbasiertes Pausieren der Prozesslogik realisiert. Die Unterbrechungsdauer ist abhängig von der Bearbeitungszeit der aktuellen Operation und u. U. von weiteren Größen (z. B. Leistung der Produktionsressource, siehe Tabelle 5.2). Nach Fertigstellung der Operation wird die Prozesslogik des ausgewählten Auftrags fortgesetzt (Schnittstelle »C«), wodurch dieser die Produktionsressource freigibt. Bevor die Prozesslogik der Produktionsressource in den nächsten Zyklus übergeht, wird überprüft, ob ein Terminationskriterium eingetreten ist, das den Prozess vollständig beendet. Als Terminationskriterium kann bspw. geprüft werden, ob der zuletzt bearbeitete Auftrag der letzte Auftrag im System war und keine weiteren Aufträge von der Quelle produziert werden. Ferner kann der Abbruch der Prozesslogik von der Überschreitung einer bestimmten Zeitschranke abhängig gemacht werden.

5.3.2 Definition von maschinellen Lernaufgaben und Gestaltung von Agenten

Die Prozessmodelle für Aufträge (Abbildung 5.9) und Produktionsressourcen (Abbildung 5.10) in Abschnitt 5.3.1 beinhalten lediglich symbolische Darstellungen für agentenbasierte Allokations- bzw. Sequenzierungsentscheidungen. Die Zielstellung des folgenden Abschnitts ist zunächst eine präzise Darstellung, wie Allokations- und Sequenzierungsentscheidungen als maschinelle Lernaufgaben (ML-Aufgaben) zu formulieren und die entsprechenden Agenten zu gestalten sind. Zu diesem Zweck widmet sich Abschnitt 5.3.2.1 der Formulierung von Allokationsproblemen für die Ressourcenbelegungsplanung als ML-Aufgabe. Im Anschluss werden in Abschnitt 5.3.2.2 drei Varianten zur Formulierung von Sequenzierungsproblemen, zum Zweck der Reihenfolgeplanung und indirekten Losbildung, als ML-Aufgaben diskutiert.

Die in den Teilabschnitten 5.3.2.1 und 5.3.2.2 vorgestellten Konzepte verwenden teils gleiche, teils verschiedenartige Informationen von Aufträgen und Produktionsressourcen, um Zustände zum Zweck der Aktionsermittlung durch Vorwärtspropagierung zu bilden. Ohne Anspruch auf Vollständigkeit listet Tabelle 5.3 relevante Informationen von Aufträgen und Produktionsressourcen auf, die für die Bildung von Zuständen in den verschiedenen Formulierungsvarianten verwendet werden. Bei den gelisteten Zustandsdaten handelt es sich zu großen Teilen um Untermengen der Attribute von Aufträgen und Produktionsressourcen aus Tabelle 5.1 bzw. Tabelle 5.2. Als Zustandsdaten nicht berücksichtigt werden solche Attribute, die entweder keinen Bezug zu Allokations- und Sequenzierungsproblemen haben (z. B. die Freigabezeit, welche die Ankunft eines Auftrags im betrachteten System bestimmt) oder für die alternative Attribute mit einer höheren Informationsdichte existieren (z. B. kann die Liste bereits fertiggestellter bzw. noch auszuführender Operationen eines Auftrags über den zeitabhängigen Fertigstellungsgrad aggregiert berücksichtigt werden). Ferner enthält Tabelle 5.3 einige zusätzliche Zustandsdaten, die zur Laufzeit der Umgebung und auf Basis weiterer Attribute berechnet werden (z. B. die Ausfallwahrscheinlichkeit einer Produktionsressource basierend auf dem Attribut »Verfügbarkeit« oder diverse Warteschlangenstatistiken). Bei den Zustandsdaten handelt es sich entweder um normierte oder standardisierte Gleitkommazahlen oder um binär kodierte Werte. Die Standardisierung bzw. Normierung dient insbesondere dem Zweck, dass verschieden ausgeprägte Zustandsdaten auf einen Wertebereich skaliert und somit vergleichbar gemacht werden. Im Fall einer binären Kodierung (bspw. für Auftrags- oder Rüstfamilien) wird das 1-aus-n-Kodierungsverfahren verwendet,

wobei n der Anzahl von Auftrags- bzw. Rüstfamilien entspricht. Das 1-aus-n-Kodierungsverfahren bildet einen diskreten Wertebereich über einen binären Vektor ab. Die Länge des Vektors entspricht der Anzahl von Werten. Das Vektorelement, das den zu kodierenden Wert repräsentiert, nimmt den Wert eins an. Die restlichen Vektorelemente besitzen den Wert null.

5.3.2.1 Formulierung von Allokationsproblemen für die Ressourcenbelegungsplanung als maschinelle Lernaufgabe

Für die Allokation von Aufträgen zu Produktionsressourcen wird jede Aktion des Agenten mit einer Ressource des betrachteten Produktionsbereichs assoziiert. Sofern der Agent durch ein KNN repräsentiert wird, entspricht die Anzahl der Ausgabeneuronen der Anzahl von Produktionsressourcen des betrachteten Produktionsbereichs. Aus allgemeiner ML-Perspektive resultiert hieraus ein Klassifikationsproblem und aus spezifischer RL-Perspektive ein Entscheidungsproblem mit diskreten Aktionsraum. Abbildung 5.11 veranschaulicht den Einsatz von Agenten für die Allokation von Aufträgen anhand eines Beispiels.

Jedes Mal, wenn ein Auftrag einer Produktionsressource zuzuordnen ist, werden verschiedene Informationen aus der Umgebung als Zustand zusammengefasst. Der Zustand wird vorwärts durch das Agentenmodell propagiert, um die Auswahlwahrscheinlichkeiten für alle Produktionsressourcen des Bereichs zu ermitteln. Im Beispiel von Abbildung 5.11 weist das zweite Ausgabeneuron die größte Anregung mit einem Wert von 0,58 auf. Auftrag A würde somit der zweiten Produktionsressource zugewiesen werden, sofern der Entscheidungspolitik des Agenten Folge geleistet wird.

Für die Zustandsbildung sind sowohl Informationen des Auftrags als auch Attribute der Produktionsressourcen und ihrer vorgelagerten Warteschlangen relevant. In Abbildung 5.11 wird die Netzeingabe des Agenten stark aggregiert dargestellt. Jedes Eingabeneuron ist lediglich repräsentativ für eine beliebige Anzahl von Attributen des Auftrags bzw. der jeweiligen Produktionsressource und ihrer Warteschlange. Einige relevante Informationen für die Bildung von Zuständen können Tabelle 5.3 entnommen werden. Die momentanen Rüstungen der Produktionsressourcen werden nicht als Zustandsinformationen für Allokationsentscheidungen berücksichtigt, weil diese erst zu dem Zeitpunkt relevant sind, zu welchem der Auftrag für die Bearbeitung auf der Produktionsressource ausgewählt wird. Ferner ist es möglich, dass sich die Rüstung einer Produktionsressource nach dem Zeitpunkt der Allokation noch mehrfach ändert, bevor der

Tabelle 5.3 Relevante Informationen von Aufträgen und Produktionsressourcen zur Beschreibung von Zuständen für verschiedene Formulierungen von Allokations- und Sequenzierungsproblemen als maschinelle Lernaufgaben

Zustandsdaten von Aufträgen					Abbildung 5.11	Abbildung 5.12	Abbildung 5.13	Abbildung 5.14
Bezeichnung		Datentyp						
Bearbeitungszeit(en) der nächsten Operation (für alle Produktionsressourcen)		Gleitkommazahl			✔	✔	✔	
Fertigstellungsfrist		Gleitkommazahl			✔	✔	✔	
Fertigstellungsgrad		Gleitkommazahl			✔	✔	✔	
Familie		Gleitkommazahl oder binär (1-aus-n-Kodierung)			✔	✔	✔	
Zustandsdaten von Produktionsressourcen								
Bezeichnung		Datentyp						
Arbeitslast		Gleitkommazahl			✔	✔		✔
Rüstung		Gleitkommazahl oder binär (1-aus-n-Kodierung)				✔		✔
Ausfallwahrscheinlichkeit		Gleitkommazahl			✔	✔		✔
Leistung		Gleitkommazahl			✔	✔		✔
Warteschlangenstatistiken					✔			✔
· Summe · Minimum · Maximum · 0,25-, 0,5-, 0,75-Quantil	…der…	· Bearbeitungszeiten · Fertigstellungsfristen · Fertigstellungsgrade	…über…	· alle Aufträge · alle Aufträge einer Familie, ermittelt für alle Familien · Gleitkommazahlen				

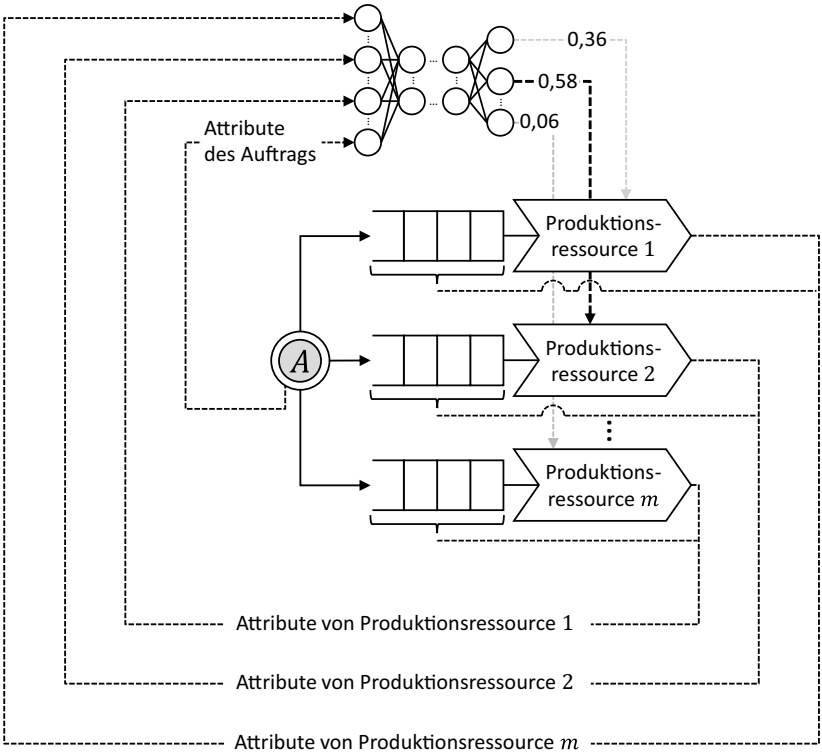


Abbildung 5.11 Formulierung von Allokationsproblemen als maschinelle Lernaufgabe. (In Anlehnung an Lang et al. 2021a)

allozierte Auftrag bearbeitet wird, weil zuvor andere Aufträge mit unterschiedlichen Familien durch den nachgelagerten Sequenzierungsagenten zur Bearbeitung ausgewählt werden.

Aus der in Abbildung 5.11 illustrierten Formulierung von Allokationsproblemen folgt, dass immer dann ein neuer Agent mit angepasster Ausgabeschicht von Grund auf neu angelernt werden muss, sobald sich die Anzahl der Produktionsressourcen verändert. Eine bessere Skalierbarkeit könnte dadurch realisiert werden, indem die Ausgabeneuronen des Agenten nicht mit den Produktionsressourcen, sondern mit Allokationsregeln assoziiert werden. Eine Formulierung für eine ähnliche Lernaufgabe, welche die Sequenzierung von Aufträgen mithilfe von

Prioritätsregeln adressiert, wird in Abschnitt 5.3.2.2 diskutiert. Im Rahmen dieser Arbeit wird jedoch argumentiert, dass eine derartige Skalierbarkeit für die Ressourcenbelegungsplanung nicht notwendig ist, da sich die Dimensionierung von Produktionssystemen der taktischen bis strategischen Planungsebene zuordnet. Somit ist die Annahme legitim, dass die Anzahl von Produktionsressourcen über eine Monate bis Jahre andauernde Planungsperiode konstant bleibt, wodurch das Risiko und der Aufwand für ein erneutes Training vernachlässigbar ist.

5.3.2.2 Formulierung von Sequenzierungsproblemen für die Reihenfolgeplanung und Losbildung als maschinelle Lernaufgabe

Im Gegensatz zu Allokationsproblemen existieren mindestens drei skalierbare (Anforderung A8) Varianten für die Formulierung von Sequenzierungsproblemen als ML-Aufgabe. Alle drei Varianten können für eine flexible Anzahl von Aufträgen eingesetzt werden, weisen jedoch unterschiedliche Vor- und Nachteile auf. Sowohl in der ersten als auch in der zweiten Variante werden Sequenzierungsprobleme aus allgemeiner ML-Sicht als Regressionsproblem bzw. aus spezifischer RL-Sicht als Entscheidungsproblem mit kontinuierlichem Aktionsraum betrachtet. Das bedeutet, dass der Agent lediglich ein Ausgabeneuron besitzt, sofern es sich bei dem Agenten um ein KNN handelt. Die Ausgabe des Agenten wird als Priorität des jeweiligen Auftrags interpretiert.

In der ersten Variante ermittelt der Agent jedes Mal, wenn die betrachtete Produktionsressource verfügbar wird, eine neue Priorität für jeden Auftrag in der vorgelagerten Warteschlange. Sofern die Entscheidungspolitik des Agenten Anwendung findet, wird derjenige Auftrag gewählt, der die maximale Priorität aufweist. Abbildung 5.12 illustriert das Konzept anhand eines Beispiels. Für die Berechnung von Aktionen kann der Agent entweder über die Warteschlange iterieren und die enthaltenen Aufträge einzeln und nacheinander priorisieren oder alternativ (und wie ebenfalls in Abbildung 5.12 dargestellt) eine vektorisierte Priorisierung vornehmen. Eine vektorisierte Priorisierung führt zu derselben Ausgabe wie eine iterative Priorisierung, ist jedoch laufzeiteffizienter, da weniger Berechnungsschritte notwendig sind. Bei der vektorisierten Priorisierung wird zunächst ein Los von Zuständen gebildet. Die Losgröße entspricht der Anzahl von Aufträgen in der betrachteten Warteschlange. Jeder Zustand setzt sich aus den individuellen Zustandsdaten des jeweiligen Auftrags zusammen sowie aus den Zustandsdaten der betrachteten Produktionsressource, wobei Letztere zum Entscheidungszeitpunkt für alle Aufträge identisch sind. Zu diesem Zweck wird der Zustandsvektor der Produktionsressource entsprechend der Anzahl von Aufträgen in der Warteschlange kopiert und an die jeweiligen Zustandsvektoren der

Aufträge angehängt. Wird das Zustandslos vorwärts durch den Agenten propagiert, ist dessen Ausgabe wiederum ein Los von Aktionen, die als Prioritäten der Aufträge interpretiert werden. Durch Ermittlung des Indizes mit der maximalen Priorität kann nun auf den nächsten zu bearbeitenden Auftrag in der betrachteten Warteschlange geschlossen werden.

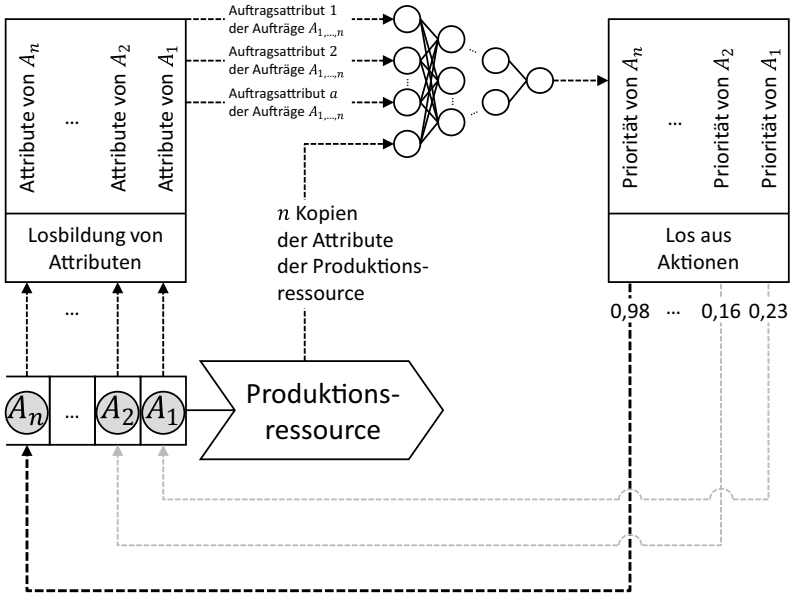


Abbildung 5.12 Erste Variante zur Formulierung von Sequenzierungsproblemen als maschinelle Lernaufgabe – Auswahl des nächsten zu produzierenden Auftrags mittels Priorisierung. (In Anlehnung an Lang et al. 2021a)

Wie bereits eingangs erwähnt, werden in der zweiten Variante ebenfalls Aufträge mithilfe eines Agenten priorisiert. Im Unterschied zur ersten Variante findet die Priorisierung nur einmalig und nicht jedes Mal statt, wenn die betrachtete Produktionsressource verfügbar und ein neuer Auftrag gewählt wird. Konkret wird ein Auftrag immer genau einmal nach dessen Allokation zu einer Produktionsressource priorisiert. Der Auftrag wird an der Stelle in der Sequenz eingefügt, an der die Priorität des vorherigen Auftrags größer oder gleich und die des nachfolgenden Auftrags niedriger als die eigene Priorität ist. Auf diese Weise

konstruiert der Agent iterativ eine Sequenz von Aufträgen, die nach dem FIFO-Prinzip abgearbeitet wird. Die Sequenz kann lediglich durch neu eingehende Aufträge verändert werden, die aufgrund ihrer Priorisierung zwischen bestimmten Aufträgen in der bestehenden Sequenz eingefügt werden. Abbildung 5.13 veranschaulicht die Formulierungsvariante anhand eines Beispiels. Für Auftrag A wird eine Priorität von 0,97 ermittelt. Der Auftrag wird somit zwischen Auftrag A_1 (vorher zu produzierender Auftrag) und A_2 (nachfolgend zu produzierender Auftrag) eingefügt.

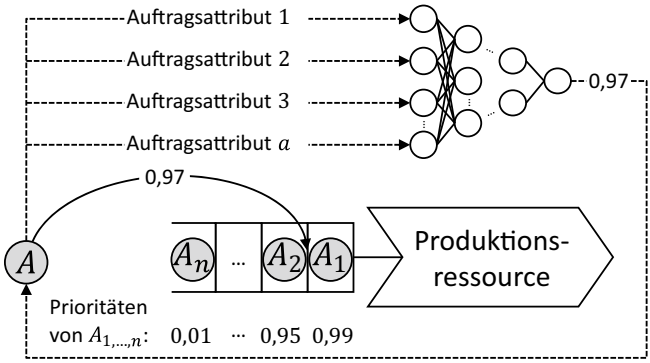


Abbildung 5.13 Zweite Variante zur Formulierung von Sequenzierungsproblemen als maschinelle Lernaufgabe – iterative Konstruktion einer Auftragssequenz durch Priorisierung

Bei der zweiten Variante sind i. d. R. lediglich die Attribute des zu priorisierenden Auftrags für die Zustandsbildung relevant. Aufgrund der iterativen Konstruktion der Sequenz werden fortlaufend neue Aufträge vor und hinter den bereits priorisierten Aufträgen eingefügt, wodurch sich deren Positionen und Bearbeitungsstartzeiten ebenfalls fortlaufend verändern. Demzufolge sind die Attribute von Produktionsressourcen für die Priorisierung von Aufträgen von geringer Bedeutung, da diese sich vom Priorisierungszeitpunkt bis zum Bearbeitungsstartzeitpunkt eines Auftrags noch mehrmals verändern können. Gegebenenfalls kann zumindest die aktuelle Rüstung der Produktionsressource als Zustandsattribut berücksichtigt werden. Für Aufträge, welchen der Agent eine hohe Priorität zuweisen möchte, kann die aktuelle Rüstung der Produktionsressource das entscheidende Kriterium sein, ob der Agent den betrachteten Auftrag auf die erste oder auf eine nachgelagerte Position setzt.

Die dritte Variante zur Formulierung von Sequenzierungsproblemen als ML-Aufgabe unterscheidet sich wesentlich von den ersten beiden vorgestellten Varianten. Die Sequenzierung von Aufträgen erfolgt indirekt, indem der Agent immer dann, wenn eine Produktionsressource verfügbar wird, die nächste anzuwendende Prioritätsregel aus einer fixen Menge von Prioritätsregeln auswählt. Analog zur agentenbasierten Allokation von Aufträgen wird somit die Reihenfolge- und Losbildung als Klassifikationsproblem bzw. als Entscheidungsproblem mit diskreten Aktionsraum behandelt. Abbildung 5.14 veranschaulicht das Prinzip anhand eines Beispiels. Jedes Ausgabeneuron ist einer bestimmten Prioritätsregel zugeordnet. Die Anzahl der Ausgabeneuronen entspricht der Anzahl der auswählbaren Prioritätsregeln. Im Beispiel erfährt das erste Ausgabeneuron mit einem Wert von 0,42 die stärkste Anregung, sodass die erste Prioritätsregel Anwendung findet, sofern der Entscheidungspolitik des Agenten gefolgt wird. Als auswählbare Prioritätsregeln kommen grundsätzlich alle reihenfolgebildenden Prioritätsregeln in Frage, so z. B. die in Abschnitt 2.3.3.2 gelisteten Beispiele. Beim Zusammenstellen der Menge auswählbarer Prioritätsregeln ist das verfolgte Optimierungskriterium zu berücksichtigen. Sofern bspw. die Gesamtdauer des Ablaufplans zu minimieren ist, sollte die Auswahlmenge aus Prioritätsregeln bestehen, welche die Produktionsressourcen bestmöglich auslasten und Nichtproduktivzeiten vermeiden, z. B. SPT und LPT.

Falls die zu optimierende Zielfunktion auf Verspätungskriterien basiert, z. B. auf der Anzahl verspäteter Aufträge oder der Gesamtverspätung über alle Aufträge, sollte die Auswahlmenge aus Prioritätsregeln bestehen, durch welche die Verspätung von Aufträgen reduziert wird, z. B. EDD und LST. Um zu gewährleisten, dass der Ansatz für eine beliebige Anzahl von Aufträgen angewandt werden kann, werden für die Definition von Zuständen Aufträge nicht separat betrachtet. Im Gegensatz zu den ersten beiden Varianten wird ferner kein Zustandslos, sondern lediglich ein einzelner Zustand vorwärts durch den Agenten propagiert, um auf die Auswahl einer Prioritätsregel zu schließen. Um dennoch die Attribute von Aufträgen zu berücksichtigen, empfiehlt es sich, einige Statistiken zu verschiedenen Attributen über alle Aufträge der betrachteten Warteschlange zu bilden. Beispiele für relevante Statistiken sind in Tabelle 5.3 gelistet. Ferner kommen für die Definition von Zuständen Attribute der betrachteten Produktionsressource in Frage, bspw. deren aktuelle Rüstung, Ausfallwahrscheinlichkeit oder Bearbeitungsgeschwindigkeit.

Die ersten beiden Varianten zur Formulierung von Sequenzierungsproblemen besitzen den Vorteil, dass alle in der Warteschlange befindlichen Aufträge, aufgrund ihrer lückenlosen Priorisierung, im Entscheidungsprozess gleichermaßen

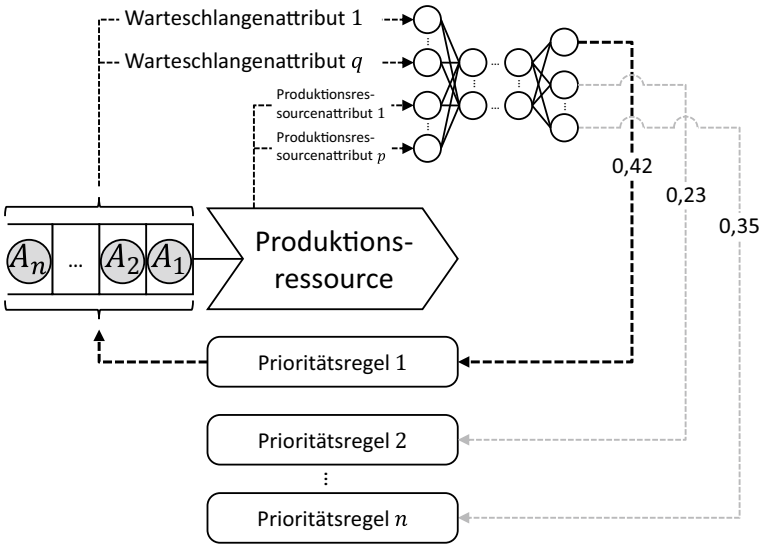


Abbildung 5.14 Dritte Variante zur Formulierung von Sequenzierungsproblemen als maschinelle Lernaufgabe – indirekte Auswahl des nächsten zu produzierenden Auftrags mittels agentenbasierter Bestimmung von Prioritätsregeln

berücksichtigt werden. Hingegen können in der dritten Variante nur diejenigen Aufträge zur Bearbeitung auf der Produktionsressource ausgewählt werden, die zum Entscheidungszeitpunkt das Auswahlkriterium von mindestens einer möglichen Prioritätsregel am meisten erfüllen.

Die erste Variante hat gegenüber der zweiten Variante den Vorteil, dass die Prioritäten immer augenblicklich zum Entscheidungszeitpunkt und somit gemäß den aktuellen Zustandsinformationen berechnet werden. Ein Nachteil der ersten Formulierungsvariante ist, dass sie zu einem gewissen Grad mit den Eigenschaften eines MEP (siehe Abschnitt 3.3.1) bricht. In einem klassischen MEP wird in einem Zustand S_t eine Aktion A_t gewählt, aus welcher der Folgezustand S_{t+1} resultiert. In der ersten Formulierungsvariante wird jedoch zu einem Zeitpunkt t ein Los von Zuständen $S_{t,1\ldots n}$ für n Aufträge gebildet, für die n Aktionen $A_{t,1,\ldots,n}$ berechnet werden. Es wird jedoch nur eine Aktion gewählt, in deren Abhängigkeit die Umgebung in der Zeit voranschreitet. Ungeachtet dessen ist es möglich RL-Algorithmen gemäß der ersten Formulierungsvariante

anzuwenden. Gradientenfreie RL-Algorithmen können ohne weitere Komplikationen angewandt werden, da sie die Parameter eines Agenten stochastisch und nicht mit dem TD-Lernverfahren anpassen. Ebenfalls sind gradientenabhängige RL-Algorithmen für die erste Formulierungsvariante anwendbar, müssen jedoch zu diesem Zweck angepasst werden. Transitionen müssen nicht nur einzelne Zustände, sondern vollständige Zustandslose enthalten. Das TD-Lernverfahren muss ebenfalls vektorisiert für Zustands- und Aktionslose implementiert werden. Eine weitere Herausforderung ist die Modellierung der Belohnungsfunktion. Jede Aktion des Agenten, respektive jede vergebene Priorität, muss durch die Belohnungsfunktion bewertet werden, ungeachtet dessen, ob der Auftrag zum Zeitpunkt der Priorisierung zur Bearbeitung ausgewählt wurde oder nicht (hierzu später mehr in Abschnitt 5.3.5.1). Zusammengefasst erfordert die erste Variante zur Formulierung von Sequenzierungsproblemen einen hohen konzeptionellen und implementierungstechnischen Aufwand, sofern sie in Verbindung mit einem gradientenabhängigen RL-Verfahren Anwendung findet.

Die zweite Formulierungsvariante ist von dem geschilderten Problem nicht betroffen, obgleich sie ebenfalls einen Agenten mit kontinuierlichen Aktionsraum zur Priorisierung von Aufträgen verwendet. Der Grund ist, dass Aufträge nur einmal priorisiert werden. Die Belohnung für die Priorisierung wird nicht unmittelbar nach der Priorisierung ermittelt, sondern erst dann, wenn der Auftrag auf der Produktionsressource bearbeitet wurde. Die Berechnung der Aktion und die Vergabe der Belohnung sind somit zeitlich entkoppelt, was mit einem zusätzlichen Implementierungsaufwand für die Verwaltung von Daten im Erfahrungsspeicher einhergeht. Ein Nachteil der zweiten Variante ist, dass lediglich Zustandsdaten von Aufträgen, jedoch keine Zustandsdaten der Produktionsressourcen, auf sinnvolle Weise für die Berechnung von Aktionen berücksichtigt werden können. Wie bereits erläutert, sind die Zustandsdaten von Produktionsressourcen in der zweiten Formulierungsvariante gegenstandslos, da die Priorisierung von Aufträgen und deren Bearbeitung auf der jeweiligen Produktionsressource ebenfalls zeitlich entkoppelt sind.

Die dritte Formulierungsvariante, die auf der Auswahl von Prioritätsregeln basiert, besitzt keinen der geschilderten Nachteile der ersten beiden Varianten. Zum einen erfüllt die dritte Variante alle Eigenschaften eines MEP, zum anderen fließen sowohl Zustandsdaten von Aufträgen als auch von Produktionsressourcen in die Berechnung von Aktionen mit ein. Der offensichtliche Nachteil ist, dass der durchsuchbare Lösungsraum durch die auswählbaren Prioritätsregeln eingeschränkt wird, weil nur diejenigen Aufträge zur Bearbeitung in Frage kommen, die zu einem Zeitpunkt t das Auswahlkriterium von mindestens einer Prioritätsregel am meisten erfüllen.

Es sei angemerkt, dass ferner eine vierte potenzielle Variante existiert, in der die Sequenzierung von Aufträgen ebenfalls als Klassifikationsproblem bzw. Entscheidungsproblem mit diskreten Aktionsraum formuliert wird. Gemäß dieser Formulierung bilden die Aktionen des Agenten direkt auf die Aufträge in der betrachteten Warteschlange ab. Obgleich Abschnitt 4.1.3 aufzeigt, dass diese Variante in der wissenschaftlichen Literatur mehrfach Beachtung findet, wird sie im vorliegenden Vorgehensmodell nicht berücksichtigt. Der Grund ist, dass diese Variante nicht in der Lage ist, auf eine flexible Anzahl von Aufträgen zu skalieren (Anforderung A8). Ein Agent, der auf eine bestimmte Anzahl von Aufträgen angelernt wurde, kann lediglich auf Auftragsvolumen angewandt werden, die kleiner oder gleich der Anzahl der während des Trainings beobachteten Aufträge sind. Ein höheres Auftragsvolumen in der Warteschlange würde das Training eines neuen Agenten mit angepasster Ausgabeschicht erfordern.

5.3.3 Integration und Inbetriebnahme von Agenten und Agentenumgebungen

In den vorhergehenden Abschnitten 5.3.1 und 5.3.2 wurde erläutert, wie Agentenumgebungen gestaltet bzw. wie Allokations- und Sequenzierungsprobleme modelliert werden müssen, sodass sie durch einen Agenten stufenweise entscheidbar sind. Der Zweck des vorliegenden Abschnitts ist, die technische Integration von Agenten und Simulationsumgebungen darzulegen. Die technische Integration dient zum einen als Grundlage für die Implementierung von RL-Trainingszyklen, zum anderen für den Einsatz von trainierten Agenten für PPS-Entscheidungen im Realbetrieb. Hierbei gestaltet sich die technische Integration von Agenten und Umgebungen für gradientenabhängige und gradientenfreie RL-Verfahren unterschiedlich. In Abschnitt 5.3.3.1 wird zunächst ein Integrationskonzept für gradientenabhängige RL-Verfahren beschrieben, gefolgt von einem Integrationskonzept für gradientenfreie RL-Verfahren in Abschnitt 5.3.3.2. Die Inbetriebnahme von Agenten und Agentenumgebungen wird nicht explizit erläutert, weil bei einer erfolgreichen Integration von Agenten und Umgebung die Inbetriebnahme lediglich die Ausführung des entsprechenden Programms erfordert. Die technische Realisierung der Inbetriebnahme kann somit als trivial betrachtet werden, sofern Agent und Umgebung fehlerfrei implementiert und miteinander integriert wurden. Ungeachtet dessen ist die Inbetriebnahme ein wichtiger Schritt im Vorgehensmodell zur Projektierung und Entwicklung von agentenbasierten Produktionsablaufsteuerungen (Abbildung 5.6 in Abschnitt 5.3), da mit dieser die technische Umsetzung der Integration validiert werden kann.

Somit findet die Inbetriebnahme im Titel des vorliegenden Abschnitts Erwähnung, um konsistent mit den Prozessschritten des Vorgehensmodells zu bleiben.

5.3.3.1 Integrationskonzept für gradientenabhängiges bestärkendes Lernen

Für die Integration von Agenten, die durch gradientenabhängige RL-Verfahren trainiert werden, muss die Simulationsumgebung gemäß eines MEP modelliert sein (siehe Abschnitt 3.3.1). Insbesondere bedeutet dies, dass der Ablauf der Simulationsumgebung schrittweise – von Zustand zu Folgezustand – sowie in Abhängigkeit von gewählten Aktionen gesteuert werden muss. Hierfür ist es notwendig das in Abschnitt 5.3.1 eingeführte Prozessmodell für den Entwurf von Agentenumgebungen um wenige Aspekte zu erweitern.

Vor diesem Hintergrund wird für die Anwendung von gradientenabhängigen RL-Verfahren empfohlen, eigens entwickelte DES-Modelle mit der OpenAI-Gym-Schnittstelle auszustatten. Die OpenAI-Gym-Schnittstelle (Brockman et al. 2016) ist ein softwaretechnisches Objektmodell und zugleich eine Modellierungskonvention, mit der Zielstellung die Entwicklung von RL-Agentenumgebungen zu vereinheitlichen. Die Schnittstelle enthält alle benötigten Funktionen, um Agenten mithilfe von gradientenabhängigen RL-Verfahren zu trainieren. Aufbauend auf Abbildung 3.5 in Abschnitt 3.3.1 veranschaulicht Abbildung 5.15 das Zusammenspiel zwischen OpenAI-Gym-Umgebung und Agent für gradientenabhängige RL-Verfahren. Ferner zeigt Abbildung 5.15 die beiden wichtigsten Methoden, mit welchen jede OpenAI-Gym-Umgebung ausgestattet ist. Konkret handelt es sich um die Reset- und Step-Methode. Beide Methoden sollen im Folgenden kurz erläutert werden.

Abbildung 5.16 zeigt den Aufbau und Ablauf der Reset-Methode als Programmablaufplan. Die Reset-Methode ist für die Initialisierung und das Zurücksetzen der Agentenumgebung verantwortlich. Die Agentenumgebung gilt dann als initialisiert, sobald sie das erste Ereignis erreicht, zu welchem eine Produktionsablaufentscheidung erforderlich ist. Je nachdem, welches Modell zur Abbildung des betrachteten Produktionsbereichs Anwendung findet (siehe Abbildung 5.3 in Abschnitt 5.2), handelt es sich hierbei um eine Allokations- oder Sequenzierungsentscheidung.

Aus konzeptioneller Sicht stellen sich sowohl für die Reset-Methode als auch für die Step-Methode zwei Fragen:

1. Wie können diejenigen Ereignisse identifiziert werden, zu welchen Produktionsablaufentscheidungen getätigt werden müssen?

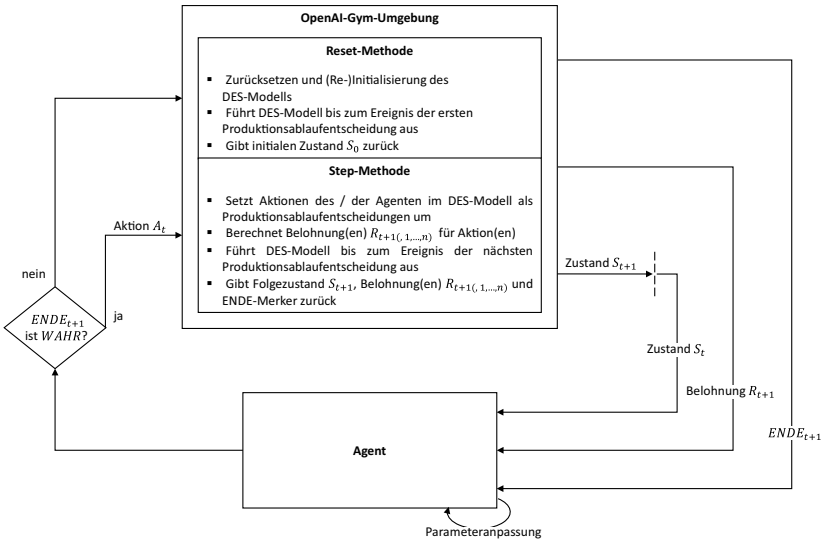


Abbildung 5.15 Wichtige Eigenschaften von OpenAI-Gym-Umgebungen und deren Integration in gradientenabhängige bestärkende Lernprozesse

2. Wie kann gewährleistet werden, dass die Simulationsumgebung genau zu diesen Ereignissen ihren Ablauf unterbricht?

Gewissermaßen alle DES-Entwicklungs- und Ablaufumgebungen bieten bereits eine interne Step-Funktion, die nicht mit der Step-Methode der OpenAI-Gym-Schnittstelle zu verwechseln ist. Die interne Step-Funktion ermöglicht es, ein DES-Modell schrittweise von Ereignis zu Ereignis zu durchlaufen. Hierbei hält das DES-Modell zu jedem Ereignis, bis die Step-Funktion erneut aufgerufen wird. Die Step-Funktion von DES-Modellen dient als Grundlage, um eine spezifizierte Reset- und Step-Methode im Sinne der OpenAI-Gym-Konvention zu entwickeln. In Abbildung 5.16 wird nach der Initialisierung des DES-Modells eine globale Merker-Variable mit der Bezeichnung »AUFRUFENDES_OBJEKT« initialisiert, ohne dass dieser ein Wert zugewiesen wird (*None*). Zum einen soll über den Merker AUFRUFENDES_OBJEKT festgestellt werden, ob eine Produktionsablaufentscheidung gefordert ist. In diesem Fall ist dem Merker ein Wert zugewiesen ($\neq None$). Zum anderen kann anhand der Art des Werts festgestellt werden, ob im DES-Modell eine Allokations- oder Sequenzierungsentscheidung

getroffen werden muss. Gehört das aufrufende Objekt der Klasse »Auftrag« an, so ist eine Allokationsentscheidung gefordert. Eine Sequenzierungsentscheidung ist notwendig, sofern der Merker AUFRUFENDES_OBJEKT ein Objekt der Klasse »Produktionsressource« referenziert. Das DES-Modell wird mithilfe der internen Step-Funktion und einer while-Schleife solange durchschritten, bis der Merker AUFRUFENDES_OBJEKT erstmals auf ein Objekt referenziert. Je nach zugewiesenem Objekt wird danach der initiale Zustand für den Allokations- oder Sequenzierungsagenten gebildet und als Ergebnis zurückgegeben.

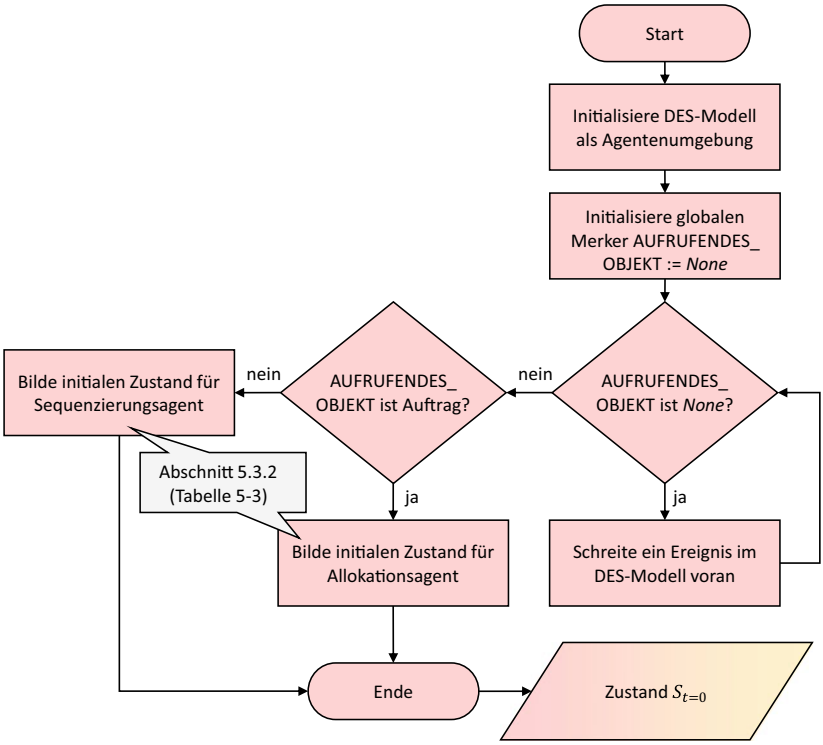


Abbildung 5.16 Algorithmischer Ablauf der Reset-Methode als Programmablaufplan

Der Einfachheit halber wird in Abbildung 5.16 der zurückgegebene initiale Zustand als $S_{t=0}$ verallgemeinert. Genau genommen muss bei der initialen Zustandsbildung hinsichtlich des initialen Zustands des Allokationsagenten

$S_{t=0,\theta_{Allok}}$ und des Sequenzierungsagenten $S_{t=0,\theta_{Seq}}$ unterschieden werden. Diese Unterscheidung gilt ferner für alle weiteren Zustände $S_{t=1,\ldots T}$, die jedoch gleichermaßen in allen folgenden Programmablaufplänen vernachlässigt wird. Sofern die Sequenzierung von Aufträgen durch Priorisierung und anschließender Auswahl des maximal priorisierten Auftrags gesteuert wird, handelt es sich bei einem Zustand S_t um ein Los von Zuständen ($S_{t,1,\ldots,n}$ bzw. $S_{t,1,\ldots,n,\theta_{Seq}}$). Diese Unterscheidung in der Notation wird in den folgenden Programmablaufplänen ebenfalls vernachlässigt. Für die Bildung von Zuständen werden diverse Attribute von Aufträgen und Produktionsressourcen aus der DES-Agentenumgebung ausgelesen. Einige relevante Zustandsdaten wurden bereits in Tabelle 5.3 (Abschnitt 5.3.2) vorgestellt. Analog der Zustände werden in den folgenden Programmablaufplänen Aktionen und Belohnungen als A_t bzw. R_{t+1} verallgemeinert. Konkreter Weise müssen diese ebenfalls für die Allokation ($A_{t,\theta_{Allok}}$ bzw. $A_{t,\theta_{Seq}}$) und Sequenzierung ($R_{t+1,\theta_{Allok}}$ bzw. $R_{t+1,\theta_{Seq}}$) spezifiziert werden. Ebenfalls existieren Aktions- und Belohnungslose ($A_{t,1,\ldots,n}$ bzw. $A_{t,1,\ldots,n,\theta_{Seq}}$ und $R_{t+1,1,\ldots,n}$ bzw. $R_{t,1,\ldots,n,\theta_{Seq}}$), sofern die Sequenzierung mittels Auftragsauswahl durch Priorisierung erfolgt. Auch dieser Sachverhalt wird in den folgenden Programmablaufplänen vernachlässigt.

Bisher wurde noch nicht erklärt, wie das aufrufende Objekt innerhalb der Simulationslogik der entsprechenden Merker-Variable zugewiesen wird. Zu diesem Zweck sei nochmal Abbildung 5.9 (Prozesslogik des Auftragsflusses) und Abbildung 5.10 (Prozesslogik von Produktionsressourcen) in Abschnitt 5.3.1.2 bzw. 5.3.1.3 in Erinnerung gerufen. Beide Prozesse enthalten jeweils eine Aktivität, welche auf dasselbe Unterprogramm verweisen. Abbildung 5.17 dient als Erweiterung der Prozessmodelle aus Abbildung 5.9 und Abbildung 5.10, indem sie den Ablauf der Unterprogramme präzisiert.

Genau genommen wird immer dann, wenn in der Prozesslogikinstanz eines Auftrags oder einer Produktionsressource eine Allokations- bzw. eine Sequenzierungsentscheidung getroffen werden muss, dem globalen Merker AUFRUFENDES_OBJEKT der jeweilige Auftrag bzw. die jeweilige Produktionsressource zugewiesen, die mit der entsprechenden Prozesslogikinstanz assoziiert wird. Darauffolgend wird die Prozesslogik für null Zeiteinheiten pausiert. Analog der ersten Prozessvariante für die Auftragserzeugung (Abbildung 5.8 (a) in Abschnitt 5.3.1.1) hat das Pausieren keine Bedeutung für die Simulation des Produktionsablaufs, sondern dient lediglich der Terminierung eines zusätzlichen Ereignisses im Ereignisverwalter. Das Ereignis gewährleistet, dass die interne Step-Funktion des DES-Modells genau bei dem Ereignis hält, zu welchem eine Produktionsablaufentscheidung erforderlich ist.

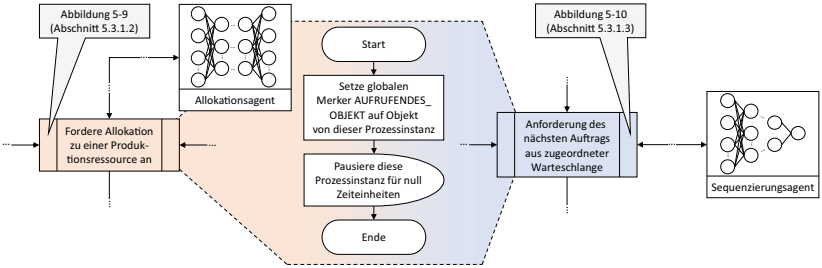


Abbildung 5.17 Präzisierung der Prozessmodelle von Aufträgen und Produktionsressourcen an der Schnittstelle zum Allokations- bzw. Sequenzierungsagenten für gradientenabhängige bestärkende Lernverfahren

Der initiale Zustand wird nun an den jeweiligen Agenten übergeben, um eine Aktion zu bestimmen. Abbildung 5.18 präsentiert einen generischen Programmablaufplan, der die agentenbasierte Berechnung von Aktionen in Abhängigkeit von eingehenden Zuständen beschreibt. Der Programmablaufplan unterscheidet, ob der Agent eine Aktionsnutzenfunktion $q_\pi(s, a; \theta)$, eine stochastische Entscheidungspolitik $\pi(a|s; \theta)$ oder eine deterministische Entscheidungspolitik $\pi(s; \theta)$ approximiert. Je nachdem finden unterschiedliche Techniken zur Exploration des Lösungsraums Anwendung. Die Approximation der Aktionsnutzenfunktion ist insbesondere für den DQN-Algorithmus charakteristisch, der ausschließlich für diskrete Aktionsräume Anwendung findet. Die Exploration des Lösungsraum wird dadurch realisiert, dass in einigen Zuständen eine zufällige Aktion anstelle der Aktion des maximal angeregten Ausgabeneurons gewählt wird. Die Explorationsstrategie wird über einen zusätzlichen Hyperparameter ε gesteuert, der ungefähr gleich, jedoch kleiner als eins ist. In jedem Zustand wird eine Zufallszahl rnd ausgewürfelt. Sofern diese kleiner als ε ist, wird eine zufällige Aktion ausgewählt. Andernfalls wird die Aktion mit der maximalen neuronalen Anregung gewählt. Darauffolgend wird ε diskontiert.

Der beschriebene Ablauf hat zur Folge, dass zu Beginn des Trainings vermehrt zufällige Aktionen gewählt werden und mit fortschreitender Zeit zunehmend der Entscheidungspolitik des Agenten gefolgt wird. Die Intention hinter diesem Vorgehen ist, dass der zufällig parametrisierte Agent zu Beginn des Trainings noch keine Entscheidungspolitik gelernt hat und durch eine verstärkte Exploration zunächst den Lösungsraum analysieren kann. Mit fortschreitender Trainingszeit ist es wiederum sinnvoll, dass die Exploration abnimmt, sodass der Agent zu einer bestimmten Entscheidungspolitik konvergiert.

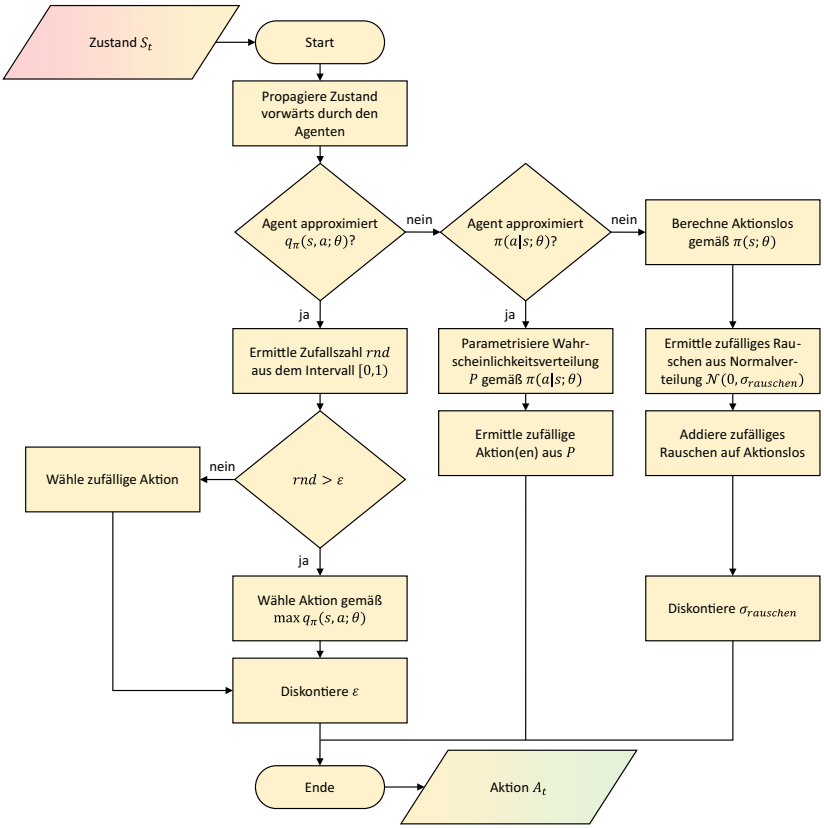


Abbildung 5.18 Aktionsermittlung durch Vorwärtspropagierung als Programmablaufplan

Hingegen ist die Approximation einer stochastischen Entscheidungspolitik bspw. für die Algorithmen A2C, A3C und PPO charakteristisch. Für die Exploration des Lösungsraums sind keine zusätzlichen Techniken notwendig. Wie einleitend in Abschnitt 3.3.4 beschrieben, werden die eigentlichen Aktionen als Zufallsstichproben von den resultierenden Wahrscheinlichkeitsverteilungen entnommen. Zu Beginn des Trainings ist es üblich, dass die ermittelten Parameter der gesuchten Wahrscheinlichkeitsverteilung noch starken Schwankungen unterliegen, sodass die resultierenden Aktionen einen breiten Lösungsraum abdecken. Mit fortschreitendem Training nehmen diese Schwankung und damit einhergehend die Streubreite der Aktionen ab.

Der Agent approximiert immer dann eine deterministische Entscheidungspolitik, wenn es sich bei dem verwendeten DRL-Verfahren bspw. um den DDPG- oder TD3-Algorithmus handelt. Beide Verfahren werden ausschließlich für kontinuierliche Aktionsräume angewandt, d. h. dass der vollständige Aktionsraum über den Aktivierungsbereich eines Ausgabeneurons abgebildet wird. Für die Exploration des Lösungsraums wird jeweils ein zufälliger Wert auf alle Aktionen addiert, der im Folgenden auch »Rauschen« oder »Rausch-Term« genannt wird. Der Rausch-Term wird stichprobenartig von einer um den Wert null zentrierten Normalverteilung gezogen. Die Standardabweichung $\sigma_{rauschen}$ ist ein benutzerdefinierter Hyperparameter, der ungefähr gleich, jedoch kleiner als eins ist und nach jeder Aktion diskontiert wird. Die Diskontierung hat zur Folge, dass zu Beginn des Trainings die Aktionen stark verrauscht sind und eine Exploration des Lösungsraums begünstigt wird. Mit zunehmender Trainingszeit nimmt das Rauschen ab, bis dieses irgendwann vernachlässigbar ist und ausschließlich der Entscheidungspolitik des Agenten gefolgt wird. Hierdurch wird die Konvergenz der Lernkurve gewährleistet.

Wie in Abbildung 5.18 dargestellt, ist der Rückgabewert des Agenten eine Aktion. Die Aktion dient insbesondere als Eingangsparameter der Step-Methode der OpenAI-Gym-Schnittstelle. Abbildung 5.19 veranschaulicht die Step-Methode als Programmablaufplan. Eine Besonderheit der im Rahmen dieser Arbeit konzipierten Step-Methode ist, dass sie nicht ausschließlich für das Training eines einzelnen Agenten ausgelegt ist, sondern ebenfalls das parallele Training von bis zu zwei Agenten (konkret einen Allokations- und einen Sequenzierungsagenten) unterstützt. Auf diese Weise eignet sich die Step-Methode zur Anwendung für beide in Abbildung 5.3 (Abschnitt 5.2) eingeführten Produktionsbereichsmodelle.

Im Folgenden soll der Ablauf der Step-Methode kurz skizziert werden. Zu Beginn wird überprüft, ob der globale Merker AUFRUFENDES_OBJEKT einen Auftrag oder eine Produktionsressource referenziert. Zu diesem Zeitpunkt ist dem globalen Merker in jedem Fall ein Wert ungleich *None* zugewiesen, entweder aufgrund der zuvor ausgeführten Reset-Methode oder aufgrund der vorherigen Iteration der Step-Methode. Je nachdem wird die Belohnungsfunktion des Allokations- oder Sequenzierungsagenten aufgerufen. Für Allokationsentscheidungen erhält der Agent stets eine skalare Belohnung. Für Sequenzierungsentscheidungen erhält der Agent ebenfalls eine skalare Belohnung, sofern die Sequenzierung indirekt durch die agentenbasierte Auswahl von Prioritätsregeln gesteuert wird. Erfolgt die Sequenzierung hingegen durch die Priorisierung von Aufträgen, wird ein Belohnungsvektor zurückgegeben, der auf die Aufträge in der

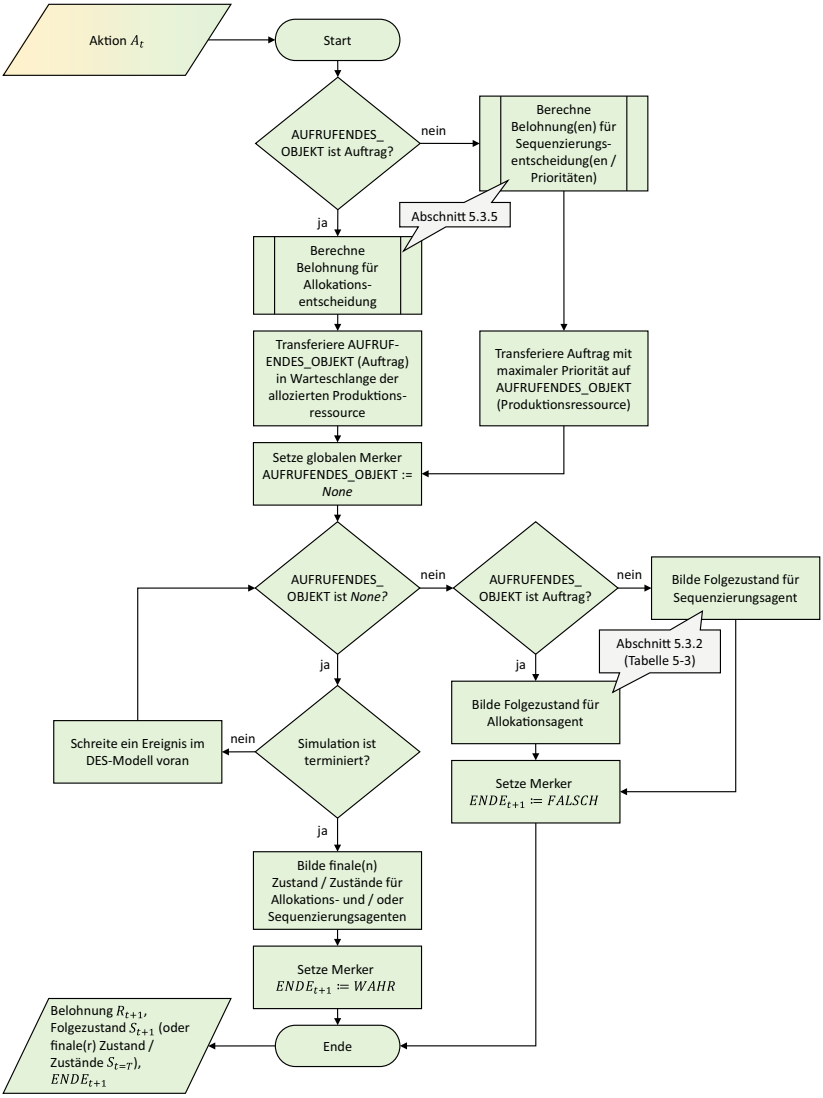


Abbildung 5.19 Algorithmischer Ablauf der Step-Methode als Programmablaufplan

vorgelagerten Warteschlange abbildet. Die Gestaltung von Belohnungsfunktionen wird gesondert in Abschnitt 5.3.5 behandelt.

Sofern es sich beim aufrufenden Objekt um einen Auftrag handelt, wird dieser darauffolgend zu einer Produktionsressource entsprechend der übergebenen Aktion alloziert. Andernfalls handelt es sich bei dem aufrufenden Objekt um eine Produktionsressource. In diesem Fall wird der Auftrag mit der höchsten Priorität ausgewählt und auf die aufrufende Produktionsressource transferiert. Falls die Sequenzierung durch die agentenbasierte Auswahl von Prioritätsregeln gesteuert wird, ist die Priorität als dasjenige Attribut zu verstehen, das von der ausgewählten Prioritätsregel für die Sortierung von Aufträgen zu Rate gezogen wird.

Der Wert der globalen Merker-Variable AUFRUFENDES_OBJEKT wird nun zurückgesetzt (:= $None$). Analog zu der oben beschriebenen Reset-Methode wird im Folgenden die interne Step-Funktion der DES-Agentenumgebung wiederholt aufgerufen, bis die Umgebung beim nächsten Ereignis hält, in dem eine Produktionsablaufentscheidung gefordert wird. Abhängig von der Art des aufrufenden Objektes wird entweder der Folgezustand des Allokations- oder Sequenzierungsagenten gebildet. Darüber hinaus wird der lokale Merker $ENDE_{t+1}$ auf $FALSCH$ gesetzt, was impliziert, dass die Simulation der Agentenumgebung noch nicht beendet ist. Bei der Ausführung der Step-Methode kann jedoch ebenfalls die gegenteilige Situation auftreten, nämlich dass die Agentenumgebung in einem finalen Zustand terminiert, in dem weder eine Allokations- noch eine Sequenzierungsentscheidung erforderlich ist. In diesem Fall werden die finalen Zustände $S_{t=T}$ von allen involvierten Agenten gebildet und der lokale Merker $ENDE_{t+1}$ auf $WAHR$ gesetzt. Der finale Zustand eines Allokations- bzw. Sequenzierungsagenten ($S_{t=T,\theta_{Allok}}$ bzw. $S_{t=T,\theta_{Seq}}$) ist ein Tensor (d. h. ein mehrdimensionaler Vektor) mit derselben Dimensionalität aller zuvor beobachteten Allokations- bzw. Sequenzierungszustände, wobei jedes Element einen Wert von null besitzt.

Zusammengefasst gibt die Step-Methode die Belohnung(en) R_{t+1}, den Folgezustand S_{t+1} und den lokalen Merker $ENDE_{t+1}$ zurück. In Abhängigkeit von $ENDE_{t+1}$ wird entschieden, ob die DES-Agentenumgebung zurückgesetzt und (u. U.) ein weiteres Mal initialisiert wird ($ENDE_{t+1} = WAHR$) oder ob eine weitere Iteration der Step-Methode durchgeführt wird ($ENDE_{t+1} = FALSCH$). Im zweiten Fall wird zuvor der Folgezustand S_{t+1} vorwärts durch den Agenten propagiert, um eine entsprechende Aktion A_{t+1} als Übergabeparameter für den erneuten Aufruf der Step-Methode zu ermitteln. Die Belohnung(en) R_{t+1} finden außerhalb der OpenAI-Gym-Umgebung für das Agententraining mithilfe eines ausgewählten RL-Verfahrens Verwendung. Abbildung 5.20 fasst das beschriebene Prozessmodell der OpenAI-Gym-Schnittstelle zusammen.

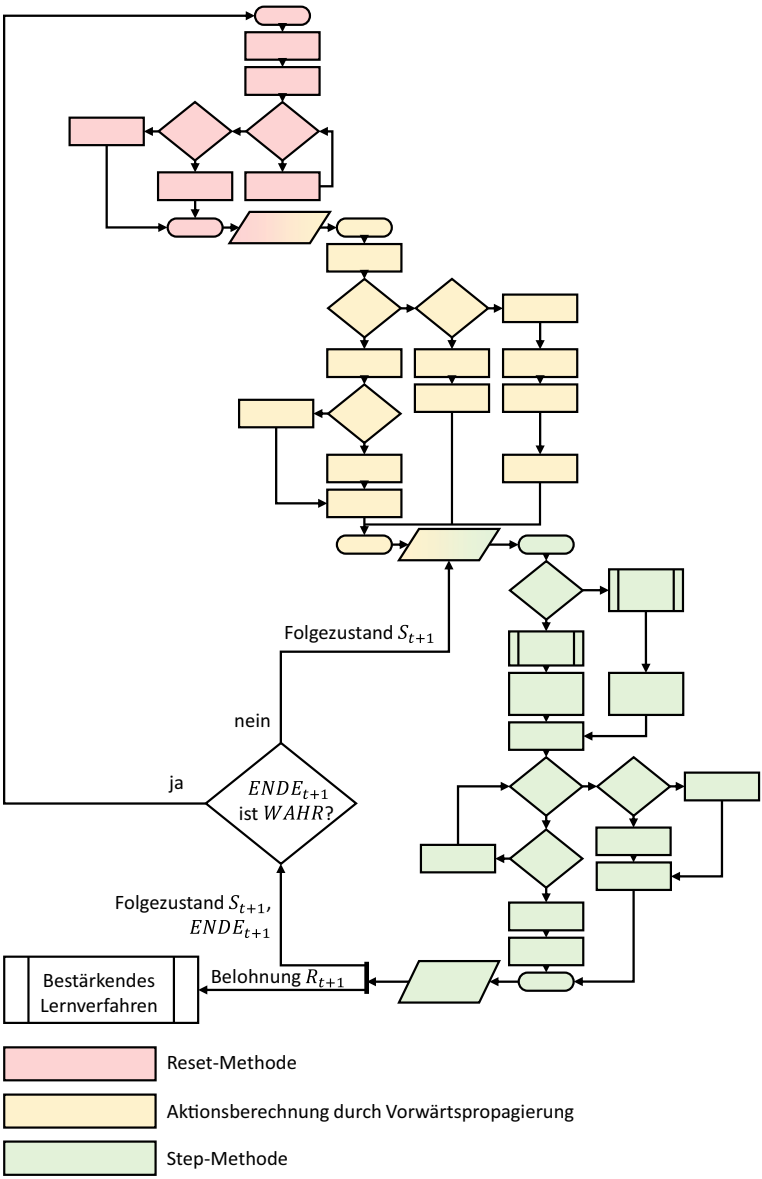


Abbildung 5.20 Vollständiges Prozessmodell der OpenAI-Gym-Schnittstelle

5.3.3.2 Integrationskonzept für gradientenfreies bestärkendes Lernen

Im Vergleich zu ihren gradientenabhängigen Verwandten ist für gradientenfreie RL-Verfahren die Einbindung von Agentenentscheidungen in DES-modellierte Umgebungen einfach zu realisieren. Der Grund ist, dass die Anwendung gradientenfreier RL-Verfahren keine Agentenumgebung mit den Eigenschaften eines MEP (siehe Abschnitt 3.3.1) erfordert. Konkret entfällt die Notwendigkeit, jede Aktion des Agenten mit einer Belohnung zu quittieren, weil der Lernprozess nicht schrittweise und einem Gradienten folgend, sondern stochastisch abläuft. Die Belohnung dient somit lediglich der Bewertung einer Lösung, jedoch nicht als Lernsignal, auf dessen Basis die Änderungsrichtung und -größenordnung der Agentenparameter gesteuert werden. Aus diesem Grund muss lediglich eine Belohnung am Ende jeder Episode ausgeschüttet werden. Zusammengefasst kann auf die aufwändige Implementierung der im vorhergehenden Abschnitt vorgestellten OpenAI-Gym-Schnittstelle verzichtet werden. Stattdessen erfolgt für gradientenfreie RL-Verfahren die Integration von Agenten und Agentenumgebungen nach einem grundlegend anderen, einfacheren Prinzip als bei gradientenabhängigen RL-Verfahren. Bei gradientenabhängigen RL-Verfahren werden die Agenten und die Agentenumgebung als dezentrale Software-Komponenten verstanden, welche insbesondere über die Step-Methode der OpenAI-Gym-Schnittstelle verknüpft sind. Hingegen werden bei gradientenfreien RL-Verfahren die Agenten durch geeignete Übergabeparameter in das DES-Modell als Objektattribute eingebettet und somit unmittelbar in die Simulationsablauflogik integriert.

Bereits im vorhergehenden Abschnitt wurde die Prozesslogik des Auftragsflusses (Abbildung 5.9 in Abschnitt 5.3.1.2) und die Prozesslogik von Produktionsressourcen (Abbildung 5.10 in Abschnitt 5.3.1.3) für gradientenabhängige RL-Verfahren präzisiert (Abbildung 5.17 in Abschnitt 5.3.3.1). Nach demselben Schema konkretisiert Abbildung 5.21 für gradientenfreie RL-Verfahren das entsprechende Unterprogramm für die gleichen beiden Aktivitäten.

Da der Agent direkt in das Simulationsmodell eingebettet wird, erfolgt die Bildung von Allokations- und Sequenzierungszuständen ebenfalls innerhalb der Ablauflogik des Simulationsmodells. Der generierte Zustand wird nun vorwärts durch den jeweiligen Agenten propagiert und darauffolgend in eine Produktionsablaufentscheidung übersetzt. Abbildung 5.22 konkretisiert beide Aktivitäten als Programmablaufplan. Analog zu Abschnitt 5.3.3.1 werden die unterschiedlichen Zustände und Aktionen von Allokations- und Sequenzierungsagenten zu Gunsten der Übersichtlichkeit als S_t bzw. A_t zusammengefasst. Ferner handelt es sich bei S_t und A_t um Zustands- bzw. Aktionslose, sofern der Sequenzierungsagent Aufträge auf Basis einer vorausgehenden Priorisierung auswählt. Im

Unterschied zu ihren gradientenabhängigen Verwandten müssen für gradientenfreie RL-Verfahren keine zusätzlichen Explorationsstrategien für die Berechnung von Aktionen implementiert werden. Die Explorationsstrategien der gradientenabhängigen RL-Verfahren dienen insbesondere der Vermeidung von lokalen Optima. Dadurch dass der Gradient immer dem am nächsten gelegenen lokalen Extrempunkt folgt, kann bei einer zu geringen Schrittweite die Entscheidungspolitik in einem lokalen Optimum gefangen sein. Aufgrund der Tatsache, dass gradientenfreie RL-Verfahren die Parameter von Agenten zufällig verändern, sind diese von dem geschilderten Problem nicht betroffen. Obgleich die Bildung von Zuständen, die Berechnung von Aktionen sowie deren Umsetzung zu Produktionsablaufentscheidungen innerhalb der Simulationslogik stattfinden, ist es empfehlenswert die entsprechenden Ablaufroutinen als separate Funktionen oder Klassen zu implementieren. Dies fördert die Wiederverwendbarkeit und Wartbarkeit der DES-Agentenumgebung. Durch die Definition von zusätzlichen Übergabeparametern innerhalb der DES-Agentenumgebungsklasse können einfach und aufwandsarm unterschiedliche Agentenmodelle sowie Arten von Zustandsräumen experimentell untersucht werden.

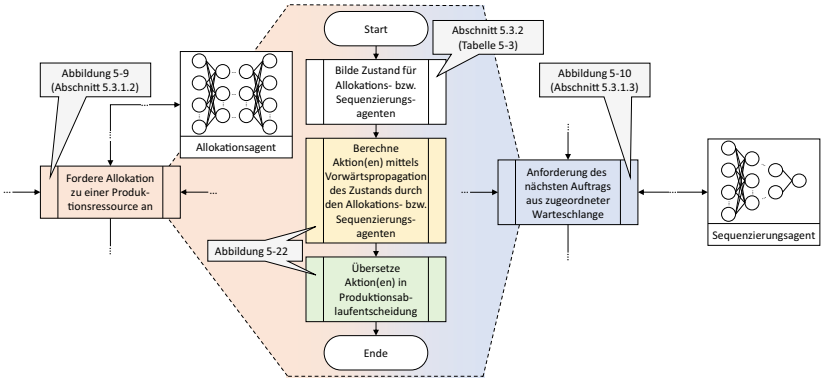


Abbildung 5.21 Präzisierung der Prozessmodelle von Aufträgen und Produktionsressourcen an der Schnittstelle zum Allokations- bzw. Sequenzierungsagenten für gradientenfreie bestärkende Lernverfahren

5.3.4 Auswahl und Implementierung von bestärkenden Lernverfahren

Bisher adressierte das Vorgehensmodell in den Abschnitten 5.3.1 bis 5.3.3 die Konzeption und Implementierung von DES-basierten Umgebungen, welche Probleme der Produktionsablaufplanung abbilden, sowie von Agenten für unterschiedliche Arten von Produktionsablaufentscheidungen. Zu diesem Zeitpunkt können die Funktionalität und Integration von Agenten und deren Umgebung bereits validiert werden, indem die Agenten mit zufälligen Parametern initialisiert und darauffolgend für Produktionsablaufentscheidungen in der Umgebung eingesetzt werden. Der vorliegende Abschnitt beschäftigt sich nun mit der Auswahl und Implementierung von RL-Verfahren, um die Parameter eines Agenten in Richtung einer optimalen Entscheidungspolitik schrittweise anzupassen.

Hinsichtlich der Auswahl von geeigneten RL-Verfahren sei zunächst festzustellen, dass keine wissenschaftlichen Arbeiten bekannt sind, welche die Dominanz eines bestimmten RL-Verfahrens gegenüber allen existierenden RL-Verfahren attestieren. Wie im Vorgehensmodell zur Umsetzung von agentenbasierten Produktionsablaufsteuerungen in Abbildung 5.6 (Abschnitt 5.3) dargestellt, ist die Auswahl eines geeigneten RL-Verfahrens vielmehr als experimentierbare Größe zu verstehen, die für die Optimierung des Agententrainings untersucht werden kann.

Abhängig von der betrachteten Teilproblemstellung der Produktionsablaufplanung (Allokation oder Sequenzierung) und deren Formulierung als ML-Aufgabe können jedoch einige RL-Verfahren im Voraus von der Untersuchung ausgeschlossen werden. Abbildung 5.23 präsentiert zu diesem Zweck ein Klassifikationsschema für den Einsatz von RL-Verfahren für die Produktionsablaufplanung. Dem Schema liegt die Überlegung zugrunde, dass einige RL-Verfahren lediglich für diskrete Aktionsräume ausgelegt sind, während andere RL-Verfahren allen voran für den Einsatz für Probleme mit kontinuierlichen Aktionsräumen gestaltet wurden. Hieraus kann geschlussfolgert werden, dass insbesondere für Allokationsprobleme die Algorithmen DDPG, TD3 und SAC von Untersuchungen ausgeschlossen werden können. Sofern ein Sequenzierungsproblem gemäß Abbildung 5.12 oder Abbildung 5.13 (Abschnitt 5.3.3.2) formuliert wird, kann hingegen auf die Untersuchung von all denjenigen Verfahren verzichtet werden, die vornehmlich für diskrete Aktionsräume ausgelegt sind (Q-Learning, DQN, SARSA, REINFORCE). Stochastische EPA-Verfahren (A2C, A3C, TRPO, PPO) können grundsätzlich für beide Teilproblemstellungen angewandt werden, da für diskrete Aktionsräume die Entscheidungspolitik als multinominale Verteilung bzw. für kontinuierliche Aktionsräume als Normalverteilung repräsentiert wird.

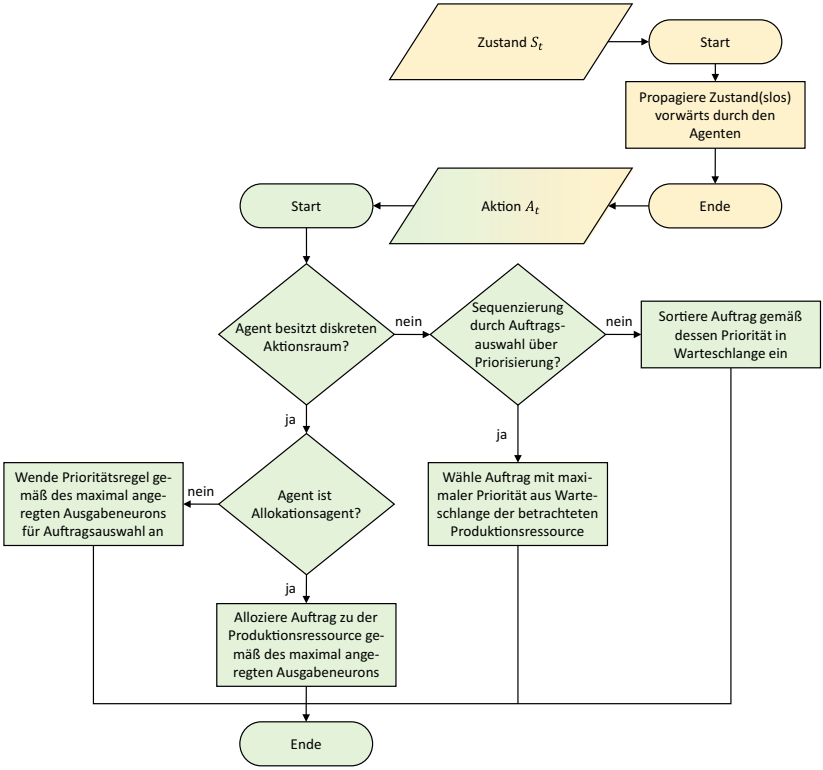


Abbildung 5.22 Programmablaufplan der Aktionsermittlung durch Vorwärtspropagierung und Übersetzung von Aktionen zu Produktionsablaufentscheidungen für gradientenfreies bestärkendes Lernen

Ferner sind ebenfalls gradientenfreie RL-Verfahren im Allgemeinen sowohl für Allokations- als auch für Sequenzierungsprobleme anwendbar.

Eine Diskussion von Implementierungsstrategien für alle gelisteten RL-Verfahren geht über den Rahmen dieser Arbeit hinaus. Zudem würde eine solche Darstellung keinen neuen wissenschaftlichen Beitrag leisten, da jedes der gelisteten RL-Verfahren bereits in wissenschaftlichen Publikationen, Onlinebeiträgen sowie in Form von öffentlich zugänglichen Software-Projekte beschrieben wurde. Aus Implementierungssicht soll stattdessen der Fokus auf den Entwurf und die

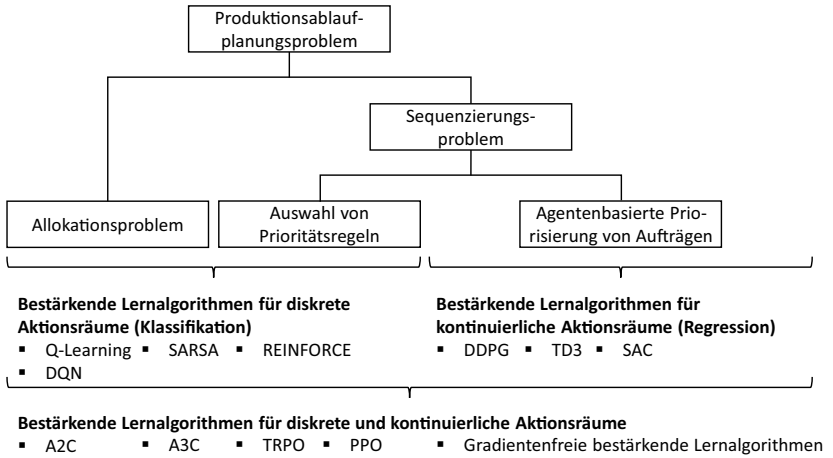


Abbildung 5.23 Klassifikation von bestärkenden Lernverfahren hinsichtlich ihrer Einsatzmöglichkeiten für die Produktionsablaufplanung

Umsetzung von Trainingsprozeduren gelegt werden. Je nachdem ob das verwendete RL-Verfahren eigens implementiert oder vorgefertigt aus einer etablierten Programmbibliothek verwendet wird, unterscheiden sich die resultierenden Trainingsprozeduren wesentlich hinsichtlich ihrer Komplexität und ihres Umfangs voneinander.

Zunächst soll eine Trainingsprozedur für den Fall skizziert werden, dass ein gradientenabhängiges RL-Verfahren für eine Agentenumgebung angewandt wird, welche die Eigenschaften eines MDP (siehe Abschnitt 3.3.1) erfüllt und den Konventionen von OpenAI-Gym entspricht. Beispielsweise erfüllen die Formulierungsvarianten für die Allokation von Aufträgen (siehe Abbildung 5.11 in Abschnitt 5.3.2.1) bzw. für die Auswahl von Prioritätsregeln (siehe Abbildung 5.14 in Abschnitt 5.3.2.2) gleichermaßen die Eigenschaften eines MDP. Folglich bietet es sich an, einen vorgefertigten RL-Algorithmus aus einer etablierten Programmbibliothek wie »OpenAI Baselines« (Dhariwal et al. 2017), »Ray RLlib« (Liang et al. 2017) oder »Stable Baselines« (Hill et al. 2018) für das Training zu verwenden. Die Trainingsprozedur ist dementsprechend einfach zu implementieren.

Abbildung 5.24 spezifiziert zwei Trainingsprozeduren am Beispiel der Bibliotheken Stable Baselines (a) und Ray RLlib (b). Beide Trainingsprozeduren überzeugen durch ihre Einfachheit, da alle komplexeren Routinen, die für das

Training erforderlich sind, innerhalb der jeweiligen Bibliothek implementiert sind. Aus konzeptioneller Sicht unterscheiden sich beide Trainingsprozeduren nur marginal. In Stable Baselines wird zunächst die Agentenumgebung initialisiert. Das resultierende Objekt wird als Parameter an das RL-Verfahren übergeben. Das Training wird über einen Methodenaufruf des gewählten RL-Verfahrens gestartet und dann in Abhängigkeit von der spezifizierten Anzahl von Episoden oder Zeitschritten (bei nicht terminierenden Umgebungen) durchgeführt. In Ray RLlib wird kein instanziiertes Objekt der Umgebungsklasse, sondern die Umgebungsklasse selbst als Parameter an das RL-Verfahren übergeben. Im Unterschied zu Stable Baselines ist es somit nicht möglich die DES-Umgebung und die OpenAI-Schnittstelle als separate Klassen zu implementieren. Dies würde erfordern, dass ein in Ray RLlib implementiertes RL-Verfahren die Agentenumgebung nicht als Klasse, sondern in Form eines instanziierten Objekts berücksichtigen kann. Nur auf diese Weise kann zunächst eine DES-Umgebung erstellt und darauffolgend mit der OpenAI-Schnittstelle umhüllt werden. Jegliche Parameter der Umgebung müssen ferner in einem separaten Konfigurationsobjekt definiert werden. Ein weiterer Unterschied zu Stable Baselines ist, dass in Ray RLlib die Anzahl der Episoden bzw. Zeitschritte nicht mittels eines Übergabeparameters der Trainingsmethode zur Verfügung gestellt wird. Gewöhnlich wird stattdessen das Training über ein Schleifenkonstrukt außerhalb der Trainingsmethode gesteuert.

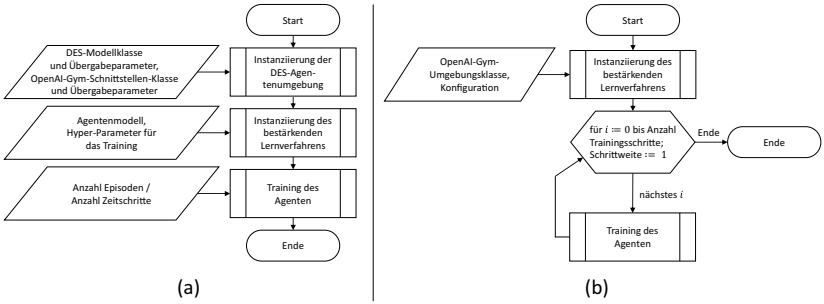


Abbildung 5.24 Trainingsprozedur für gradientenabhängige bestärkende Lernverfahren unter Verwendung der Bibliotheken (a) Stable Baselines und (b) Ray RLlib

Für Sequenzierungsstrategien, in welchen Aktionen als Auftragsprioritäten betrachtet werden, erfordert die Anwendung von vorgefertigten RL-Algorithmen diverse Anpassungen innerhalb der Programmbibliotheken. Der Grund ist, dass die jeweiligen Formulierungsvarianten gewisse Eigenschaften eines MDP und

damit einhergehend bestimmte Modellierungskonventionen von OpenAI-Gym nicht erfüllen. Konkret erfordert die erste Formulierungsvariante (siehe Abbildung 5.12 in Abschnitt 5.3.2.2) die Berechnung von mehreren Aktionen und Belohnungen, bevor die Umgebung in Abhängigkeit eines gewählten Auftrags voranschreiten kann. Bei der zweiten Formulierungsvariante (siehe Abbildung 5.13 in Abschnitt 5.3.2.2) können die Belohnungen erst verspätet vergeben werden, da die Priorisierung und die Auswahl von Aufträgen zeitlich entkoppelt sind. Aufgrund ihrer Komplexität sind Anpassungen in den Programmbibliotheken sehr aufwändig. Demgegenüber kann der Aufwand geringer sein, wenn das jeweilige RL-Verfahren zzgl. der Trainingsprozedur eigens implementiert werden. Abbildung 5.25 präsentiert eine beispielhafte Trainingsprozedur für genau diesen Zweck.

Die skizzierte Trainingsprozedur fußt auf der Annahme, dass die Agentenumgebung mit der OpenAI-Gym-Schnittstelle ausgestattet ist und die Umgebung in einem finalen Zustand S_T terminiert. Über eine for-Schleife durchläuft die Prozedur eine benutzerdefinierte Anzahl von Episoden in der Agentenumgebung. Zu diesem Zweck werden die in Abschnitt 5.3.3.1 eingeführte Reset- (Abbildung 5.16) und Step-Methode (Abbildung 5.19) der OpenAI-Gym-Schnittstelle sowie die Methode zur Aktionsermittlung durch Vorwärtspropagierung (Abbildung 5.18) verwendet. Die Reset-Methode wird lediglich zu Beginn einer Episode aufgerufen. Das Durchschreiten der Umgebung mithilfe der Step- und Aktionsermittlungsmethode wird über eine while-Schleife gesteuert, die terminiert, sobald in einem Umgebungszustand $ENDE_{t+1}$ als $WAHR$ beobachtet wird. Immer dann, wenn eine Aktion bestimmt wurde, in deren Abhängigkeit die Umgebung vorangeschritten ist, wird eine Belohnung R_{t+1}, ein Folgezustand S_{t+1} sowie der Merker $ENDE_{t+1}$ ausgegeben. Die drei Informationen dienen zunächst der Vervollständigung der aktuellen Transition $(S_t, A_t, R_{t+1}, S_{t+1}, ENDE_{t+1})$. In Abhängigkeit davon, ob der untersuchte RL-Algorithmus auf das N-Step-Bootstrapping- (siehe Abschnitt 3.3.5) oder das gewöhnliche TD-Lernverfahren zur Generierung von Lernsignalen zurückgreift, wird die Transition entweder dem Erfahrungsspeicher oder einer bestehenden bzw. einer neuen Trajektorie hinzugefügt.

Das N-Step-Bootstrapping-Verfahren wird insbesondere von stochastischen EPA-Verfahren verwendet, bei welchen es sich i. d. R. um On-Policy-Methoden handelt. On-Policy-Methoden trainieren mit Beobachtungen, die mit den aktuellen Agentenparametern θ gesammelt wurden. In einem Trainingsschritt werden die Ziel-Zustandsnutzenwerte durch Rückwärtsrechnung ermittelt, indem, ausgehend von der Zustandsnutzenprognose des Critic-Modells, die Belohnungen von dem am weitesten in der Zukunft liegenden bis zu dem am weitesten in der

Vergangenheit liegenden Zustand kumuliert werden. Das genaue Prinzip wurde bereits in Abschnitt 3.3.5 dargelegt.

RL-Algorithmen, die das TD-Lernverfahren verwenden, sind bspw. der DQN-, der DDPG- und der TD3-Algorithmus. Jedes der drei Verfahren besitzt einen sogenannten Erfahrungsspeicher. Während eines Trainingsschritts wird eine benutzerdefinierte Anzahl zufälliger Transitionen aus dem Erfahrungsspeicher zu einem Los gebildet. Die Transitionen sind zeitlich nicht zusammenhängend und stammen größtenteils aus Beobachtungen, die mit vergangenen Agentenparametern θ^{-} verarbeitet wurden. Aus diesem Grund werden jene Verfahren auch als Off-Policy-Methoden klassifiziert.

Nachdem die vollständige Transition entweder dem Erfahrungsspeicher oder der Trajektorie hinzugefügt wurde, prüft die Prozedur in Abbildung 5.25, ob in der aktuellen Iteration ein Trainingsschritt durchzuführen ist. Das Prüfkriterium ist abhängig vom verwendeten RL-Verfahren. Trajektorien bildende Verfahren führen gewöhnlich wiederholend und stets nach einer konstanten Anzahl von observierten Zuständen einen Trainingsschritt aus. RL-Verfahren, die über einen Erfahrungsspeicher verfügen, führen gewöhnlich nach jedem observierten Zustand einen Trainingsschritt aus, sofern der Erfahrungsspeicher über eine ausreichende Anzahl von Transitionen verfügt. Manche RL-Verfahren, z. B. der TD3-Algorithmus, kombinieren die beiden zuvor beschriebenen Prüfkriterien.

Abschließend prüft die Trainingsprozedur, ob $ENDE_{t+1}$ den Wert $WAHR$ besitzt. Ist dies der Fall, so ist die Agentenumgebung terminiert, wodurch entweder eine neue Episode begonnen oder das Training beendet wird. Sofern $ENDE_{t+1}$ den Wert $FALSCH$ besitzt, wird der Folgezustand S_{t+1} fortan als aktueller Zustand S_t betrachtet, für die Eröffnung einer neuen Transition verwendet und schließlich dem Agenten zur Aktionsermittlung mittels Vorwärtspropagierung übergeben.

Abbildung 5.25 kann gewissermaßen als standardmäßige Trainingsprozedur betrachtet werden. Sie eignet sich sowohl für die Allokation von Aufträgen als auch für die Auswahl von Prioritätsregeln, deren Formulierung als ML-Aufgaben gleichermaßen die Eigenschaften eines MDP und die Modellierungskonventionen von OpenAI-Gym erfüllen. Ferner eignet sich die Prozedur für das Training von Agenten, welche Auftragsreihenfolgen durch Priorisierung aller wartenden Aufträge und anschließender Auswahl des nächsten zu produzierenden Auftrags bilden (siehe Abbildung 5.12 in Abschnitt 5.3.2.2). In diesem Fall ändert sich lediglich das Datenformat der Belohnungen und Aktionen, die durch die Step-Methode zurückgegeben und innerhalb von Transitionen gespeichert werden. Anstelle von Skalaren werden beide Größen nun als Vektoren ausgedrückt, deren Größe der Anzahl wartender Aufträge in der betrachteten Warteschlange

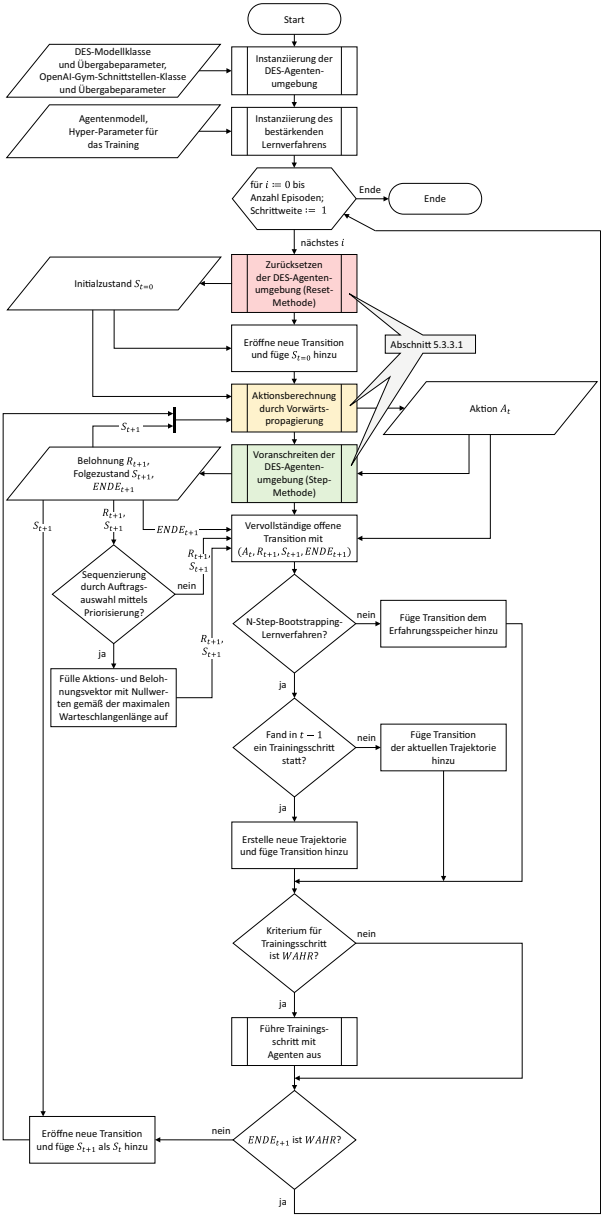


Abbildung 5.25 Beispielhafte Trainingsprozedur für selbstimplementierte gradientenabhängige bestärkende Lernverfahren

entsprechen. Um zu gewährleisten, dass das N-Step-Bootstrapping- bzw. das TD-Lernverfahren funktionieren, ist es notwendig, dass die Aktions- und Belohnungsvektoren während des Trainings von einheitlicher Länge sind. Ein Grund hierfür ist u. a., dass bei der Rückwärtsrechnung zur Ermittlung der Ziel-Nutzenwerte Belohnungen von unterschiedlichen Zeitschritten kumuliert werden. Aus mathematischer Sicht handelt es sich hierbei nur dann um eine zulässige Operation, wenn die zu addierenden Vektoren gleich groß sind. Zu diesem Zweck muss während des Trainings eine maximale Anzahl von Aufträgen in der betrachteten Warteschlange angenommen werden. Wie in Abbildung 5.25 dargestellt, werden hierfür in den Zeitschritten, in welchen die gegenwärtige Anzahl kleiner als die maximale Anzahl von Aufträgen ist, die Aktions- und Belohnungsvektoren mit einer entsprechenden Menge von Nullwerten aufgefüllt, bevor sie dem Erfahrungsspeicher hinzugefügt werden. Es sei angemerkt, dass die Annahme einer maximalen Auftragsanzahl nur für das Training, jedoch nicht für den anschließenden Einsatz notwendig ist.

Hingegen kann die Trainingsprozedur in Abbildung 5.25 nicht für die zweite Formulierungsvariante von Sequenzierungsproblemen eingesetzt werden, in welcher mittels Priorisierung eine Auftragssequenz iterativ konstruiert wird. Hierzu müsste die Trainingsprozedur so angepasst werden, dass Transitionen zunächst ohne eine Belohnung dem Erfahrungsspeicher oder der Trajektorie hinzugefügt werden. Eine Belohnung kann erst ermittelt werden, sobald der jeweilige Auftrag bearbeitet wurde. Im Grenzfall, unter der Annahme eines temporär finiten Auftragshorizonts, würde der erste Auftrag auf einer Produktionsressource erst dann bearbeitet werden, wenn jeder Auftrag der initialen Auftragsliste einer Position in der Sequenz zugeordnet wurde. Um zu gewährleisten, dass die richtige Belohnung der richtigen Transition zugeordnet wird, kann bspw. eine zusätzliche Transitions-ID eingeführt werden. Diese verknüpft die jeweilige Transition mit dem jeweiligen Auftrag, der im entsprechenden Zeitschritt für die Zustandsbildung analysiert wurde. Über die Transitions-ID kann nach der Bearbeitung eines Auftrags die resultierende Belohnung der entsprechenden Transition zugeordnet werden. Ferner muss gewährleistet sein, dass unvollständige Transitionen nicht für das Training berücksichtigt werden. Für RL-Verfahren, welche einen Erfahrungsspeicher verwenden, kann bspw. eine zusätzliche Liste gepflegt werden, die unvollständige Transitionen temporär puffert. Bei RL-Verfahren, die auf N-Step-Bootstrapping zur Erzeugung von Lernsignalen zurückgreifen, kann bspw. gewartet werden, bis die Umgebung in einem finalen Zustand terminiert, sodass alle Aufträge einer Episode für die Bildung einer Trajektorie verwendet werden.

Aufgrund ihrer Vielfalt und Individualität ist es nicht zweckmäßig eine generische Trainingsprozedur für gradientenfreie RL-Verfahren herzuleiten. Analog zu

den theoretischen Grundlagen in Abschnitt 3.4 soll eine Trainingsprozedur für den NEAT-Algorithmus als Stellvertreter der gradientenfreien RL-Verfahren skizziert werden. Abbildung 5.26 veranschaulicht eine mögliche Trainingsprozedur von NEAT, unter der Annahme, dass eine vorimplementierte Version des Algorithmus aus der Programmbibliothek NEAT-Python verwendet wird. Ergänzend sei an dieser Stelle ein weiteres Mal auf Abschnitt 3.4.2 verwiesen, in welchem das NEAT-Verfahren ausführlich beschrieben wird.

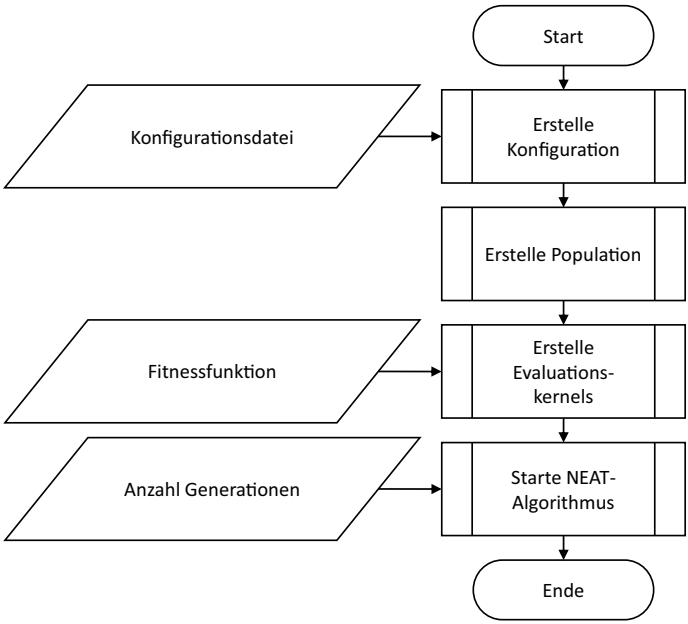


Abbildung 5.26 Trainingsprozedur für den NEAT-Algorithmus der Programmbibliothek NEAT-Python, stellvertretend für gradientenfreie bestärkende Lernverfahren

Der Programmablauf ist ähnlich simpel strukturiert wie die Trainingsprozeduren von gradientenabhängigen RL-Verfahren unter Verwendung einer etablierten Programmbibliothek (siehe Abbildung 5.24). Analog zu diesen sind komplexere Routinen, die für das Training erforderlich sind, innerhalb der jeweiligen Bibliotheken implementiert und werden in Abbildung 5.26 in Form von Unterprogrammbausteinen zusammengefasst. NEAT erfordert für die Initialisierung einige

Hyperparameter, die innerhalb einer Konfigurationsdatei definiert und gespeichert werden. Die Hyperparameter sind insbesondere für die Steuerung des Trainingsprozesses verantwortlich und werden in Abschnitt 5.3.6 näher beleuchtet. Nachdem die Konfiguration eingelesen wurde, wird eine Population von Genomen erstellt. Die Größe der Population geht aus der benutzerdefinierten Konfiguration hervor. Im nächsten Schritt wird eine spezifische Anzahl von Evaluationskernels instanziiert. Ein Evaluationskernel ist für die Berechnung der Lösungsgüte von Genomen verantwortlich. Konkret wird auf einem Evaluationskernel eine Instanz der DES-Agentenumgebung ausgeführt. Nachdem eine Episode endet, werden verschiedene Kennzahlen aus der Umgebung ausgelesen und in einer Fitnessfunktion verarbeitet, welche die Lösungsgüte des analysierten Genoms ausgibt. Die Fitnessfunktion ist ebenfalls benutzerdefiniert und gilt als das Pendant zur Belohnungsfunktion gradientenabhängiger Verfahren. Fitnessfunktionen für verschiedene Anwendungsfälle werden in Abschnitt 5.3.5 spezifiziert. Die Anzahl von Evaluationskernels ist grundsätzlich frei wählbar, wird jedoch durch die Anzahl verfügbarer Prozessorkerne begrenzt. Sinnhafter Weise kann jeder logische Prozessorkern einen Evaluationskernel ausführen. Im letzten Schritt in Abbildung 5.26 wird der NEAT-Algorithmus gestartet. Hierbei wird gewöhnlich eine Anzahl von Generationen definiert, nach der die Ausführung des Algorithmus spätestens terminiert. Ferner können weitere Terminierungskriterien in der Konfigurationsdatei definiert werden, z. B. die Überschreitung (bzw. Unterschreitung) einer benutzerdefinierten oberen (bzw. unteren) Schranke bezogen auf den Rückgabewert der Fitnessfunktion bei Maximierungsproblemen (bzw. Minimierungsproblemen). Im Gegensatz zu ihren gradientenabhängigen Verwandten erscheint es nicht sinnvoll ein gradientenfreies RL-Verfahren von Grund auf eigenständig zu implementieren. Die Intention ein gradientenabhängiges RL-Verfahren selbst zu implementieren ist, dass einige der in Abschnitt 5.3.2 dargestellten Varianten zur Formulierung von ML-Aufgaben nicht die Eigenschaften eines MDP erfüllen. Die vorimplementierten RL-Algorithmen aus den etablierten Programmbibliotheken funktionieren für diese Formulierungsvarianten nicht, weil sie unter der Annahme implementiert wurden, dass die Agentenumgebung den Konventionen von OpenAI-Gym und den Eigenschaften eines MDP folgt. Aufgrund der Tatsache, dass gradientenfreie RL-Verfahren insbesondere stochastische Suchtechniken für die Anpassung der Agentenparameter verwenden, sind sie nicht darauf angewiesen, dass die Agentenumgebung den Eigenschaften eines MEP folgt. Aus dieser Beobachtung kann ebenfalls abgeleitet werden, dass alle in Abschnitt 5.3.2 dargestellten Varianten zur Formulierung von ML-Aufgaben für alle gradientenfreien RL-Verfahren problemlos implementiert werden können.

5.3.5 Gestaltung von Belohnungsfunktionen

Das Training mithilfe von RL-Verfahren wird stets über eine Belohnungsfunktion gesteuert. In gradientenabhängigen RL-Verfahren dient die Belohnungsfunktion der Generierung von Lernsignalen. Das Lernsignal wird rückwärts durch das Agentenmodell propagiert, um die Gradienten aller trainierbaren Parameter zu berechnen. Die Parameter des Agenten werden sodann mittels Gradientenabstieg (bei ANB-Verfahren – siehe Abschnitt 3.3.3) oder Gradientenaufstieg (bei EPA-Verfahren – siehe Abschnitt 3.3.4) aktualisiert.

In gradientenfreien RL-Verfahren wird die Belohnungsfunktion auch als Fitnessfunktion bezeichnet. Sie dient der Bewertung einer vollständigen Lösung, die nach Ablauf einer Episode und in Abhängigkeit der Sequenz von Agentenentscheidungen resultiert. In gradientenfreien RL-Verfahren übt die Belohnungsfunktion nur einen geringen Einfluss auf die Steuerung des Agententrainings aus. Sie beeinflusst u. a., wann das jeweilige RL-Verfahren terminiert (z. B. sobald ein bestimmter Fitnessschwellenwert über- oder unterschritten wurde), welche Lösungskandidaten (respektive Agentenparameterkombination) temporär beibehalten werden (die Top n Lösungen) und welche Lösungskandidaten weiteren Veränderungen unterzogen werden (alle weiteren Lösungen). Hingegen hat die Belohnungsfunktion keinen Einfluss auf die Veränderungsrichtung und -ausprägung der trainierbaren Agentenparameter, da beide Aspekte zufallsgesteuert sind.

Entgegen der Reihenfolge in den vorausgegangenen Abschnitten soll die Gestaltung von Belohnungsfunktionen zunächst für gradientenfreie RL-Verfahren dargelegt werden, weil diese konzeptionell einfacher sind als die Belohnungsfunktionen ihrer gradientenabhängigen Verwandten. Die geringere Komplexität rührt daher, dass lediglich am Ende einer Episode (und nicht nach jeder Aktion, wie bei gradientenabhängigen RL-Verfahren) eine Belohnung vergeben werden muss. Die Belohnung dient lediglich der Bewertung der Gesamtlösung. Aus diesem Grund kann die Zielfunktion des Optimierungsproblems ebenfalls als Belohnungsfunktion für gradientenfreie RL-Verfahren verwendet werden. Die geläufigsten Zielfunktionen zur Optimierung von Ablaufplänen wurden bereits in Abschnitt 2.3.1.3 gelistet. Ferner ist es möglich eine multikriterielle Zielfunktion, die mehrere Optimierungskriterien berücksichtigt, gemäß Formel (6) aus Abschnitt 2.3.2.1 als Belohnungsfunktion zu definieren. In diesem Zusammenhang kann es für das Training ebenfalls förderlich sein, zusätzliche Kennzahlen in die Belohnungsfunktion mitaufzunehmen, die über eine vollständige Episode ermittelt werden, bei welchen es sich jedoch um keine direkten Optimierungskriterien handelt. Ein Beispiel für eine solche Kennzahl ist die Anzahl der

Rüstvorgänge auf einer Produktionsressource, deren Verringerung häufig mit einer Erhöhung des Produktivzeitanteils einhergeht. Eine Erhöhung des Produktivzeitanteils kann bspw. wiederum zur Minimierung der Gesamtdauer eines Produktionsablaufplans oder zur Verringerung der Verspätung einzelner Aufträge beitragen.

Für gradientenabhängige RL-Verfahren ist es notwendig, dass eine Belohnung für jede berechnete Aktion ausgegeben wird. Demzufolge ist es nicht möglich die zu optimierende Zielfunktion als Belohnungsfunktion zu verwenden, weil diese lediglich im finalen Zustand S_T eine Belohnung ausgeben könnte. Stattdessen muss die Belohnungsfunktion so gestaltet sein, dass in jedem Zustand bzw. für jede Aktion die vergebene Belohnung auf die zu optimierende Zielfunktion schließt. Konkret müssen die Belohnungsfunktion und die zu optimierende Zielfunktion insofern miteinander korrelieren, dass steigende Belohnungen bzw. sinkende Bestrafungen mit einer Verbesserung der Zielfunktion einhergehen, während steigende Bestrafungen bzw. sinkende Belohnungen zu einer Verschlechterung der Zielfunktion führen. In Abhängigkeit davon, ob der Agent die Allokation oder Sequenzierung von Aufträgen verantwortet, eignen sich für die Bildung der Belohnungsfunktion unterschiedliche Arten von Kennzahlen.

Um die Prinzipien der Gestaltung von Belohnungsfunktionen für gradientenabhängige RL-Verfahren zu beleuchten, kommt es dem intuitiven Verständnis zugute, einen beliebigen Produktionsprozess von dessen Senke bis zu dessen Quelle zu betrachten. Zur Veranschaulichung sei das Modell eines Produktionsbereichs mit lokalen Warteschlangen für jede Produktionsressource angenommen (Abbildung 5.3 in Abschnitt 5.2). Dieses Modell ist für die Diskussion von verschiedenen Belohnungsstrategien besonders geeignet, da die Steuerung der Produktion sowohl Allokations- als auch Sequenzierungsentscheidungen erfordert. Wird der entsprechende Produktionsprozess entgegengesetzt der Richtung des Materialflusses analysiert, so liegt der Fokus der Betrachtung zunächst auf Belohnungsfunktionen für Sequenzierungsentscheidungen.

5.3.5.1 Zeitbasierte Belohnungsfunktionen

Für die Bewertung von Sequenzierungsentscheidungen eignen sich grundsätzlich zeitbasierte Kennzahlen. Der primäre Grund ist, dass es sich bei den in Abschnitt 2.3.1.3 aufgeführten Zielfunktionen um zeitbasierte Optimierungskriterien handelt. Durch die Verwendung von zeitbasierten Kennzahlen ist grundsätzlich der Entwurf einer Belohnungsfunktion möglich, die mit der Zielfunktion korreliert.

Optimierungskriterien, welche die Minimierung von Durchlaufzeiten zum Ziel haben, z. B. die kumulierte Durchlaufzeit F oder die Gesamtdauer eines Ablaufplans C_{max}, können u. a. auf Basis der verstrichenen Simulationszeit t_{sim} bewertet werden. Dieser Ansatz wurde in der Literatur u. a. von Minguillon und Lanza (2019) untersucht. In diesem Fall wird jede Sequenzierungsentscheidung mit der verstrichenen Simulationszeit bestraft, indem sie mit dem Faktor (-1) multipliziert wird. Zusammengefasst ergibt sich die folgende Belohnungsfunktion, sofern die Belohnungsvergabe erst dann erfolgt, wenn der Auftrag die Produktionsressource verlässt.

$$\mathcal{R}_C(j) = (-1) * C_{j,o} = (-1) * t_{sim} \tag{5.1}$$

Formel (5.1) entspricht somit der Fertigstellungszeit von Operation o des selektierten Auftrags j. Die Intention ist, dass der Agent die Ermittlung ebensolcher Entscheidungen erlernt, durch die jegliche Zeitverbräuche möglichst vermieden und die erhaltenen Bestrafungen minimiert werden. Somit ist zum Zeitpunkt T die theoretisch beste Belohnung bzw. geringste Bestrafung das negative Äquivalent der minimalen Gesamtdauer eines Ablaufplans. Auf dieser Basis lassen sich diverse weitere Belohnungsfunktionen herleiten, welche die Durchlaufzeit von einzelnen Aufträgen oder des gesamten Ablaufplans verringern. Wie Formel (5.2) zeigt, kann bspw. ebenfalls die Durchlaufzeit $F_{j,o}$ der zuletzt bearbeitete Operation o des selektierten Auftrags j als Belohnungsfunktion herangezogen werden, um auf diese Weise die kumulierte Durchlaufzeit F als Optimierungskriterium zu adressieren.

$$\mathcal{R}_F(j) = (-1) * F_{j,o} = (-1) * (t_{sim} - r_{j,o}) \tag{5.2}$$

Die Freigabezeit $r_{j,o}$ in Formel (5.2) entspricht dem Eintrittszeitpunkt von Auftrag j im Produktionsbereich, der für die Bearbeitung der aktuellen Operation o zuständig ist. Zeitgleich entspricht $r_{j,o}$ dem Eintrittszeitpunkt von Auftrag j in der entsprechenden Warteschlange der allozierten Produktionsressource, da im vorliegenden Beispiel die Allokation von Aufträgen ohne Zeitverzug nach deren Ankunft im Produktionsbereich vonstattengeht. Formel (5.2) berechnet nur dann die Durchlaufzeit der Auftragsoperation, wenn t_{sim} nach abgeschlossener Bearbeitung von Operation o gemessen wird. Erfolgt hingegen die Erhebung von t_{sim} vor Bearbeitungsbeginn von Operation o, berechnet Formel (5.2) nicht die Durchlaufzeit, sondern die Wartezeit des entsprechenden Auftrags in der Warteschlange. In beiden Fällen eignet sich Formel (5.2), um auf durchlaufzeitorientierte Optimierungskriterien zu schließen.

Falls der Agent Entscheidungen entsprechend der ersten Variante zur Formulierung von Sequenzierungsproblemen als ML-Aufgabe trifft (Auswahl des nächsten zu produzierenden Auftrags mittels Priorisierung, siehe Abbildung 5.12 in Abschnitt 5.3.2.2), wird ein Aktionsvektor mit Auftragsprioritäten berechnet, dessen Größe der Anzahl von Aufträgen in der betrachteten Warteschlange entspricht. Obgleich lediglich ein Auftrag gemäß dem Maximum des Aktionsvektors zur Bearbeitung ausgewählt wird, können ebenfalls die verbleibenden Aufträge mit einer entsprechend gestalteten Belohnungsfunktion bewertet werden. Dies hat den Vorteil, dass eine höhere Anzahl von Trainingsdaten zu ein und demselben Entscheidungszeitpunkt generiert werden kann. Als Beispiel soll die Belohnungsfunktion aus Formel (5.1) für eine beliebige Anzahl von Aufträgen in einer Warteschlange generalisiert werden. Die Aufträge einer Warteschlange können zu einer angenommenen endgültigen Sequenz gemäß der absteigenden Reihenfolge ihrer zugeordneten Prioritäten (Aktionen) sortiert werden. Die Belohnungsfunktion in Formel (5.3) zeigt, dass nun für jeden Auftrag an beliebiger Position k in der angenommenen Sequenz ein Erwartungswert für die Fertigstellungszeit der jeweiligen Operation berechnet werden kann.

$$\mathcal{R}_{\mathbb{E}[C]}(i,k) = (-1) * \mathbb{E}_\pi[C_{k,o}(i)|A_t = a] = (-1) * \mathbb{E}_\pi\left[t_{sim} + \tilde{p}_i + \sum_{q=1}^{k} p_{q,o} \middle| A_t = a \right] \tag{5.3}$$

In Kontrast zu Formel (5.1) liegt Formel (5.3) die Annahme zugrunde, dass die Belohnungsvergabe nicht erst dann erfolgt, wenn der Auftrag die Produktionsressource verlassen will, sondern genau zu dem Zeitpunkt, bevor der Auftrag auf die Produktionsressource gelangt. Das bedeutet, dass die Verweilzeit auf der Produktionsressource nicht in der Simulationszeit t_{sim} enthalten ist und dementsprechend prognostiziert werden muss. Zu diesem Zweck sei für Formel (5.3) und folgende Formeln die Prozesszeiten $p_{j,o}$ und $\tilde{p}_i$ eingeführt. Die Prozesszeit $p_{j,o}$ (bzw. $p_{q,o}$ in Formel (5.3)) subsumiert alle Zeitanteile eines Auftrags j, welche für die Bearbeitung einer Operation o anfallen. In jedem Fall beinhaltet $p_{j,o}$ stets die Operationszeit o_j. Je nach Problemart können zusätzliche Zeitanteile anfallen, bspw. Rüst- und Transferzeiten. Sofern verschiedene Zeitanteile stochastisch sind, gehen ebenjene als Erwartungswerte der jeweils zugrundeliegenden Wahrscheinlichkeitsverteilung in $p_{j,o}$ ein. Die Prozesszeit $\tilde{p}_i$ beschreibt die Restlaufzeit für die Bearbeitung der aktuellen Operation auf

Produktionsressource i, zu welcher der Auftrag zugeordnet wurde. In der praktischen Umsetzung wird der Belohnungsfunktion in Formel (5.3) der Index der Produktionsressource, zu welcher der Auftrag zugeordnet wurde, als Parameter übergeben. Für die Berechnung der erwarteten Fertigstellungszeit der aktuellen Operation des k-ten Auftrags in der betrachteten Warteschlange werden die Restlaufzeit der Produktionsressource, die erwarteten Prozesszeiten der vorgelagerten Aufträge $(1, \ldots, k-1)$ sowie des betrachteten k-ten Auftrags kumuliert und mit der Simulationszeit t_{sim} addiert. Ungeachtet dessen, ob die Bearbeitung von Aufträgen stochastischen Zeitanteilen unterliegt, handelt es sich bei den in Formel (5.3) kalkulierten Fertigstellungszeiten stets um Erwartungswerte, da die Annahme getroffen wird, dass alle Aufträge gemäß der aus dem Aktionsvektor A_t resultierenden Reihenfolge bearbeitet werden. Tatsächlich wird jedoch nur ein Auftrag zur Bearbeitung ausgewählt. Die Reihenfolge der verbleibenden Aufträge zu den Folgezeitpunkten $t+1, t+2, \ldots, T$ kann aufgrund variierender Aktionsvektoren weiterhin Permutationen unterliegen. Eine Ausnahme kann der Erwartungswert der Fertigstellungszeit des zum Zeitpunkt t ausgewählten Auftrags bilden. Der vorkalkulierte Erwartungswert entspricht für diesen Auftrag dann und nur dann der tatsächlichen Fertigstellungszeit, wenn alle Prozessparameter der Produktionsressource deterministisch sind.

Für zeitbasierte Optimierungskriterien, welche die Unpünktlichkeit von Aufträgen messen – konkret die Gesamtverspätung über alle Aufträge T, die Anzahl verspäteter Aufträge U und die maximale Verspätung L_{max} –, müssen die Belohnungsfunktionen aus den Formeln (5.1) und (5.3) konzeptionell erweitert werden. Zunächst sei festzustellen, dass die drei Optimierungskriterien T, U und L_{max} sich aus der Unpünktlichkeit einzelner Aufträge L_j ergeben. Für einzelne Sequenzierungsentscheidungen eignet sich die Unpünktlichkeit $L_{j,o}$ einer Operation o von Auftrag j als maßgebliche Kennzahl zur Bildung von Belohnungsfunktionen, um die Verspätung von Aufträgen als Bewertungskriterium zu berücksichtigen. Der Berechnung der Unpünktlichkeit einer einzelnen Operation liegt zunächst die Überlegung zugrunde, dass die Fertigstellungsfrist eines Auftrags d_j gewöhnlich so dimensioniert ist, dass alle Operationen fristgerecht bearbeitet werden können. Vor diesem Hintergrund kann d_j zunächst auf einzelne Operationen heruntergebrochen werden. Formel (5.4) zeigt die im Rahmen dieser Arbeit präferierte Berechnungsvorschrift zur Ermittlung der Unpünktlichkeit $L_{j,o}$ einer beliebigen Operation o von einem beliebigen Auftrag j. newpage

$$L_{j,o} = \left(t_{sim} - d_j\right) * \left(\frac{p_{j,o}}{\sum_{u=1}^{|O_j|} p_{j,u}} \right) \tag{5.4}$$

Formel (5.4) basiert auf der Annahme, dass die Unpünktlichkeit der Operation erst dann berechnet wird, sobald der ausgewählte Auftrag die Produktionsressource verlassen will (d. h. $C_{j,o} = t_{sim}$). Die Unpünktlichkeit des Auftrags $\left(t_{sim} - d_j\right)$ wird mithilfe eines Faktors anteilsmäßig auf die aktuelle Operation heruntergebrochen. Der Faktor setzt die Prozesszeit der aktuellen Operation ins Verhältnis zur Summe aller (erwarteten) Prozesszeiten der am Auftrag auszuführenden Operationen.

Sofern Entscheidungen entsprechend der ersten Variante zur Formulierung von Sequenzierungsproblemen als ML-Aufgabe getroffen werden, ist es analog zu Formel (5.3) möglich eine Berechnungsvorschrift für die Unpünktlichkeit eines beliebigen Auftrags an einer beliebigen Sequenzposition k zu formulieren.

$$\mathbb{E}_\pi[L_{k,o}(i)|A_t = a] = \mathbb{E}_\pi\left[\left(\left(t_{sim} + \tilde{p}_i + \sum_{q=1}^{k-1} p_{q,o}\right) - d_k\right) * \left(\frac{p_{k,o}}{\sum_{u=1}^{|O_j|} p_{k,u}}\right) + p_{k,o} \middle| A_t = a\right] \tag{5.5}$$

Im Unterschied zu Formel (5.4), jedoch analog zu Formel (5.3), unterliegt Formel (5.5) der Annahme, dass die Belohnungsvergabe vor Berücksichtigung der eigentlichen Verweilzeit des Auftrags auf der Produktionsressource erfolgt. Vor diesem Hintergrund müssen zunächst die Restlaufzeit der betrachteten Produktionsressource i sowie die (erwarteten) Prozesszeiten aller Aufträge, die gemäß der aus dem Aktionsvektor resultierenden Sequenz vor Auftrag k zu bearbeiten sind, auf die aktuelle Simulationszeit addiert werden $\left(t_{sim} + \tilde{p}_i + \sum_{q=1}^{k-1} p_{q,o}\right)$. Das Zwischenergebnis ist die erwartete Startzeit von Operation o von Auftrag k. Von dieser wird die Fertigstellungsfrist d_k subtrahiert und mit dem gleichen Faktor aus Formel (5.4) gewichtet. Das Zwischenergebnis ist die Unpünktlichkeit von Operation o von Auftrag k, ohne die Prozesszeit der Operation selbst zu berücksichtigen. Erst jetzt wird die (erwartete) Prozesszeit $p_{k,o}$ aufaddiert, um die prognostizierte Unpünktlichkeit zu erhalten, welche ebenfalls die Prozesszeit der betrachteten Operation miteinbezieht.

Die Formeln (5.4) und (5.5) eignen sich zwar als Ausgangsbasis für den Entwurf von verspätungsbewertenden Belohnungsfunktionen, können jedoch nicht als eigenständige Belohnungsfunktionen dienen. Würden die Ergebnisse der Formeln (5.4) bzw. (5.5) unmittelbar als Belohnung verwendet werden, würden verspätete Aufträge positiv belohnt und verfrühte Aufträge negativ bestraft werden. Ferner ist es ebenfalls nicht ausreichend die Ergebnisse der Formeln (5.4)

und (5.5) mit dem Faktor (-1) zu multiplizieren. Auf diese Weise würden Verspätungen zwar bestraft werden, jedoch würden umso bessere Belohnungen resultieren, je früher ein Auftrag fertiggestellt wird. Eine solche Belohnungsvergabe ist aus zwei Gründen nicht sinnvoll. Aus produktionstheoretischer Sicht erzeugen zu früh fertiggestellte Aufträge zusätzliche Lagerkosten. Die Lagerkosten steigen in Korrelation mit der Zeitspanne, die ein Auftrag zu früh vor seiner Auslieferungsfrist fertiggestellt wird. Aus RL-theoretischer Sicht kann eine Belohnung von zu früh fertiggestellten Aufträgen dazu führen, dass Aufträge, die eine weit in der Zukunft liegende Fertigstellungsfrist besitzen, bevorzugt produziert werden, um die kumulative Belohnung zu maximieren. Dafür nimmt der Agent womöglich in Kauf, dass Aufträge mit einer Fertigstellungsfrist in naher Zukunft verspätet fertiggestellt werden. Idealerweise sollten alle Aufträge möglichst nah an, jedoch stets vor Eintritt ihrer Fertigstellungsfrist produziert werden.

Im Rahmen dieser Arbeit werden drei Strategien aufgezeigt, um auf Basis der Unpünktlichkeit von Aufträgen verspätungsbasierte Belohnungsfunktionen zu modellieren. Der ersten Strategie liegt die Idee zugrunde, dass Verspätungen bestraft werden, während verfrüht fertigstellte Operationen mit einer konstanten Belohnung $R_E \geq 0$ bewertet werden. Die resultierende Belohnungsfunktion bestraft somit lediglich das Ausmaß der Verspätungen von Aufträgen, vermeidet jedoch, dass Aufträge umso mehr belohnt werden, je früher sie eine Operation vollenden.

$$\mathcal{R}_T(j) = \mathcal{R}_{\mathbb{E}[T]}(j) = \begin{cases} R_E, & wenn\ L_{j,o} \leq 0 \\ (-1) * L_{j,o}, & sonst \end{cases} \tag{5.6}$$

Hierbei repräsentiert $L_{j,o}$ die wahre oder geschätzte Unpünktlichkeit der o-ten Operation von Auftrag j. Je nachdem berechnet Formel (5.6) entweder $\mathcal{R}_T$ (wahre Unpünktlichkeit) oder $\mathcal{R}_{\mathbb{E}[T]}$ (geschätzte Unpünktlichkeit). Die Idee der zweiten Strategie ist, dass sowohl verfrüht als auch verspätet vollendete Operationen bestraft werden. Jedoch werden verspätete Operationen durch einen Faktor $w_T > 1$ stärker bestraft als zu früh beendete Operationen.newpage

$$\mathcal{R}_L(j) = \mathcal{R}_{\mathbb{E}[L]}(j) = \begin{cases} L_{j,o}, & wennL_{j,o} \leq 0 \\ (-w_T) * L_{j,o}, & sonst \end{cases} \tag{5.7}$$

Analog zu Formel (5.6) berechnet Formel (5.7) $\mathcal{R}_L$ oder $\mathcal{R}_{\mathbb{E}[L]}$ in Abhängigkeit davon, ob es sich bei $L_{j,o}$ um die wahre oder geschätzte Unpünktlichkeit handelt. Die dritte Strategie normiert mittels Skalierung die erhaltenen Belohnungen in Wertebereiche, sodass sowohl verfrüht als auch verspätet beendete Operationen bestraft werden, wobei Verspätungen stets stärker bestraft werden als verfrühte Aufträge. Lediglich pünktliche Aufträge erhalten eine Belohnung von null. Aus dieser Überlegung resultiert die folgende Belohnungsfunktion in Abhängigkeit vom betrachteten Auftrag j.

$$\mathcal{R}_{Ls}(j) = \mathcal{R}_{\mathbb{E}[Ls]}(j) = \begin{cases} \frac{L_{j,o} - \hat{E}_{min}}{\hat{E}_{max} - \hat{E}_{min}} * \left(R_{\hat{E}min} - R_{\hat{E}max}\right) + R_{\hat{E}max}, & wenn\ L_{j,o} \leq 0 \\ \frac{L_{j,o} - \hat{T}_{min}}{\hat{T}_{max} - \hat{T}_{min}} * \left(R_{\hat{T}min} - R_{\hat{T}max}\right) + R_{\hat{T}max}, & sonst \end{cases} \tag{5.8}$$

wobei gilt…

$$\left(\hat{E}_{min} = 0\right) \wedge \left(\hat{E}_{max} > 0\right) \wedge \left(\hat{T}_{min} \approx 0 \wedge \hat{T}_{min} > 0\right) \wedge \left(\hat{T}_{max} > \hat{T}_{min}\right) \wedge \left(R_{\hat{E}min} > R_{\hat{E}max} > R_{\hat{T}min} > R_{\hat{E}max}\right) \tag{5.9}$$

Auch im Fall von Formel (5.8) wird entweder $\mathcal{R}_{Ls}(j)$ oder $\mathcal{R}_{\mathbb{E}[Ls]}$ in Abhängigkeit davon ermittelt, ob $L_{j,o}$ exakt berechnet oder lediglich prognostiziert wurde. Die Schranken $\hat{E}_{min}$ bzw. $\hat{E}_{max}$ sowie $\hat{T}_{min}$ bzw. $\hat{T}_{max}$ beschreiben die geschätzte minimale bzw. maximale Verfrühung eines Auftrags sowie die geschätzte minimale und maximale Verspätung eines Auftrags. Da der erste Fall in Formel (5.8) nicht nur für Auftragsoperationen gilt, die zu früh fertiggestellt wurden, sondern ebenfalls für solche, die exakt zur Fertigstellungsfrist vollendet werden ($L_{j,o} = 0$), wird im Rahmen dieser Arbeit die untere Schranke $\hat{E}_{min}$ stets gleich null gesetzt. Für die benutzerdefinierbare untere Schranke $\hat{T}_{min}$ wird angenommen, dass es sich um einen Wert nahezu gleich, jedoch größer als null handelt.

Die oberen Schranken $\hat{E}_{max}$ und $\hat{T}_{max}$ können über eine geeignete Berechnungsvorschrift geschätzt werden, welches auf das jeweilige Produktionsablaufplanungsproblem zugeschnitten ist. Beispielhaft sei ein Flexible-Job-Shop-Problem angenommen, bei dem alle Aufträge auf allen Produktionsressourcen bearbeitet werden können. Jede Operation eines jeden Auftrags kann auf jeder Produktionsressource bearbeitet werden, wobei die Operationszeiten zwischen

den Produktionsressourcen variieren. Ferner sei angenommen, dass reihenfolgeabhängige Rüstzeiten zwischen den Aufträgen existieren. Für das skizzierte Problem kann $\hat{E}_{max}$ bspw. über die folgende Formel geschätzt werden.

$$\hat{E}_{max} = \max_{j} \left(d_j - \sum_{o=1}^{|O_j|} \min_{i} o_{i,j} \right) \tag{5.10}$$

Formel (5.10) liegt die Überlegung zugrunde, dass alle Operationen desjenigen Auftrags, bei dem die Differenz zwischen seiner Fertigstellungsfrist und der Summe all seiner Operationszeiten maximal ist, ohne zusätzliche Wartezeiten und ohne vorhergehende Aufträge (d. h. $s_{j,k} = 0$) bearbeitet werden. Um die Differenz zwischen der Fertigstellungsfrist und der Summe der Bearbeitungszeiten eines Auftrags zu maximieren, wird in Formel (5.8) stets für jede Operation diejenige Produktionsressource gewählt, bei welcher die Operationszeit minimal ist.

Demgegenüber kann $\hat{T}_{max}$ unter der Annahme geschätzt werden, dass der Agent im ungünstigsten Fall alle Operationen von allen Aufträgen stets ein und derselben Produktionsressource (ggf. pro Bearbeitungsstufe) zuweist. Infolgedessen kann $\hat{T}_{max}$ bspw. gemäß der folgenden Formel berechnet werden.

$$\hat{T}_{max} = \sum_{j=1}^{|J|} \max_{k} s_{j,k} + \max_{i} \left(\sum_{j=1}^{|J|} \sum_{o=1}^{|O_j|} \overline{o}_{i,j} \right) - \min_{j} d_j \tag{5.11}$$

Die Idee von Formel (5.11) ist, dass alle Operationen desjenigen Auftrags, der die engste Fertigstellungsfrist aufweist, als letztes auf derjenigen Produktionsressource bearbeitet werden, zu welcher der Agent exklusiv Auftragsoperationen alloziert. Formel (5.11) impliziert zwei Annahmen, nämlich *(i)* dass der Agent alle Aufträge auf diejenige Produktionsressource alloziert, bei welcher die Summe aller Operationszeiten über alle Aufträge maximal ist und *(ii)* dass für jeden Auftrag stets die maximale Rüstzeit veranschlagt wird, d. h. dass jeder Auftrag j immer nach demjenigen Auftrag k bearbeitet wird, bei welchem die Rüstzeit von Auftrag j maximiert wird. Es sei angemerkt, dass Annahme *(ii)* lediglich einer vereinfachten Berechnung von $\hat{T}_{max}$ dient. In den meisten Fällen würde Annahme *(ii)* zu nicht realisierbaren Lösungen für das Produktionsablaufplanungsproblem führen, weil nicht ausgeschlossen wird, dass ein Auftrag k mehrere nachfolgende Aufträge j besitzt, obgleich jedem Auftrag k nur genau ein Auftrag j nachfolgen

kann. Um eine noch präzisere Schätzung für $\hat{T}_{max}$ abzugeben, müsste korrekterweise diejenige Fertigungssequenz ermittelt werden, bei der die Summe aller Rüstzeiten maximiert wird. Hierbei handelt es sich jedoch um ein eigenständiges Optimierungsproblem, dessen exakte Lösung zur Ermittlung einer oberen Schranke zu aufwändig ist. Zudem beeinträchtigt die geschilderte Annahme nicht die Verwendung von Formel (5.11) als obere Schranke, sondern garantiert, im Gegenteil, dass die ermittelte obere Schranke $\hat{T}_{max}$ tatsächlich größer als die wahre maximal mögliche Verspätung ist. Aufgrund der Tatsache, dass in Formel (5.11) stets die maximalen Rüstzeiten kumuliert werden, ungeachtet der Gültigkeit der resultierenden Reihenfolge, kann das gültige Maximum lediglich eine gleich große oder kleinere kumulierte Rüstzeit aufweisen.

Die dargestellten Berechnungsvorschriften sollen an dieser Stelle ausreichen, um die Grundprinzipien der Gestaltung von Belohnungsfunktionen für Sequenzierungsentscheidungen darzustellen. Wie bereits erwähnt, bilden die Formeln (5.1–5.6) die Möglichkeit zur Herleitung vieler weiterer problemspezifischer Belohnungsfunktionen. Dies wurde beispielhaft an Formel (5.1) anhand der Abwandlung in Formel (5.2) demonstriert.

5.3.5.2 Belastungsorientierte Belohnungsfunktionen

Wird der Produktionsprozess weiter entgegengesetzt des Materialflusses verfolgt, so liegt der Fokus der Betrachtung nun auf der Allokation von Aufträgen zu Produktionsressourcen. In die Überlegungen zur Gestaltung von Belohnungsfunktionen für Allokationsentscheidungen muss miteinbezogen werden, dass ein Auftrag, nach dessen Allokation, zunächst in einer Warteschlange gepuffert wird und erst durch eine Sequenzierungsentscheidung zur Bearbeitung auf die entsprechende Produktionsressource gelangt. Im Umkehrschluss bedeutet dies, dass sich zeitbasierte Produktionskennzahlen nur bedingt zur Bewertung von Allokationsentscheidungen eignen, weil die Fertigstellungszeiten von Aufträgen maßgeblich durch Entscheidungen des nachgelagerten Sequenzierungsagenten beeinflusst werden. Somit erlaubt das Belohnungssignal keinen eindeutigen Rückschluss auf die Entscheidungsgüte des Allokationsagenten. Zeitbasierte Produktionskennzahlen eignen sich lediglich dann für Allokationsentscheidungen, wenn die nachgelagerte Sequenzierung von Aufträgen einer deterministischen und statischen Steuerungslogik folgt, z. B. einer Prioritätsregel. In diesem Ausnahmefall kann die Güte der Allokationsentscheidung auf Basis des Fertigstellungszeitpunkts bewertet werden, da sich zwischen den Zeitschritten lediglich die Entscheidungspolitik für die Allokation wandelt. Hingegen bleibt die Entscheidungspolitik für die Sequenzierung unverändert und vorberechenbar. Abseits des geschilderten

Ausnahmefalls gilt es somit andere Arten von Produktionskennzahlen zu identifizieren, mit welchen zum einem Allokationsentscheidungen eindeutig bewertet werden können und welche zum anderen mit der zu optimierenden Zielfunktion korrelieren.

Produktionskennzahlen, die intuitiv für die Bewertung von Allokationsentscheidungen prädestiniert scheinen, sind ebensolche, die sich aufgrund und unmittelbar nach der Allokation eines Auftrags verändern. Hierzu zählt insbesondere die Arbeitslast auf einer Produktionsressource. Eine Belohnungsfunktion, die auf Basis der Arbeitslast $W_i^{(l)}$ von Ressource i in Produktionsbereich l Allokationsentscheidungen bewertet, kann wie folgt definiert werden.

$$\mathcal{R}_W(i,l) = (-1) * W_i^{(l)} = (-1) * \left(\tilde{p}_i^{(l)} + \sum_{q=1}^{|Q_i|} p_{i,j,o}^{(l)} \right) \tag{5.12}$$

Hierbei beschreibt $\tilde{p}_i^{(l)}$ die verbleibende Laufzeit derjenigen Operation, die zum Erhebungszeitpunkt auf Ressource i in Produktionsstufe l bearbeitet wird. Analog zu Formel (5.1) ist es für die Belohnungsvergabe notwendig, dass die Arbeitslast $W_i^{(l)}$ mit dem Faktor (-1) multipliziert wird. Die Intention ist, dass jegliche Arbeitslast auf einer Produktionsressource bestraft wird. Aufgrund der Tatsache, dass ein Auftrag in jedem Fall einer Produktionsressource zugeordnet werden muss, minimiert der Agent die erhaltene Bestrafung, indem er die Aufträge möglichst gleichmäßig entsprechend ihrer Arbeitslast auf die Produktionsressourcen verteilt. Eine gleichmäßige Verteilung der Arbeitslast resultiert wiederum in einem höheren Ausnutzungsgrad aller Ressourcen derselben Produktionsstufe. Auf diese Weise werden zudem die Pufferzeiten von Aufträgen in den Warteschlangen vor den Produktionsressourcen verringert. Eine weitere Kennzahl für die Bewertung der Gleichmäßigkeit der Belegung von Produktionsressourcen ist die Standardabweichung der Arbeitslast W_σ, die über alle Ressourcen desselben Produktionsbereichs ermittelt wird. Eine auf der Standardabweichung der Arbeitslast W_σ basierende Belohnungsfunktion kann in Abhängigkeit vom betrachteten Produktionsbereich l folgendermaßen definiert werden.

$$\mathcal{R}_{W\sigma}(l) = (-1) * \sqrt{\frac{1}{\left|M^{(l)}\right| - 1} * \sum_{i=1}^{\left|M^{(l)}\right|} \left(W_i^{(l)} - \frac{\sum_{h=1}^{\left|M^{(l)}\right|} W_h^{(l)}}{\left|M^{(l)}\right|} \right)^2} \tag{5.13}$$

Die Standardabweichung der Arbeitslast einer Produktionsstufe ist ebenfalls als Bestrafungsfunktion zu verstehen. Sie wird daher mit dem Faktor (-1) multipliziert, um Agentenentscheidungen zu bewerten. Ein Anstieg der Standardabweichung bedeutet eine zunehmend ungleichmäßige Belegung der Produktionsressourcen und wird dementsprechend bestraft. Die beste Belohnung, die ein Allokationsagent gemäß Formel (5.13) erhalten kann, ist eine Standardabweichung von null. In diesem Fall sind alle Produktionsressourcen absolut gleichmäßig belastet.

5.3.6 Training von Agenten

Sofern alle Schritte des Vorgehensmodells aus Abbildung 5.6 (Abschnitt 5.3) bis hierhin durchlaufen wurden, sind nun alle Komponenten der agentenbasierten Produktionsablaufsteuerung implementiert, um das Agententraining zu starten. Das Training von Agenten ist kein geradliniger Prozess. Das Konvergenzverhalten des Agenten während des Trainings, respektive die Konvergenzrichtung und die Konvergenzgeschwindigkeit, hängen von einer Vielzahl von Einflussgrößen ab. Hierbei impliziert bereits das Vorgehensmodell aus Abbildung 5.6 eine empfohlene Reihenfolge für das experimentelle Ausloten der verschiedenen Einflussgrößen. Demnach sollen zunächst *(i)* die Belohnungsfunktion, danach *(ii)* die Hyperparameter des gewählten RL-Verfahrens und schließlich *(iii)* das RL-Verfahren selbst experimentell variiert werden, um den Trainingsprozess zu optimieren. Dem vorgeschlagenen Vorgehen liegt die Überlegung zugrunde, dass die Belohnungsfunktion den größten Einfluss auf das Konvergenzverhalten des Agenten während des Trainings besitzt, weil diese stets problem- und benutzerspezifisch implementiert wird. Hingegen sind die meisten RL-Verfahren (und deren Hyperparameter) generisch und problemunabhängig konzipiert. Die Priorisierung von Hyperparameter-Experimenten gegenüber der Variation des RL-Verfahrens gilt insbesondere dann, wenn das RL-Verfahren eigenständig implementiert wird, sodass jedes zu testende Verfahren mit einem erheblichen Zeitaufwand für dessen Implementierung einhergeht. Besteht hingegen die Möglichkeit vorgefertigte RL-Verfahren aus einer Programmbibliothek zu verwenden, können alternativ zunächst verschiedene RL-Verfahren mit standardmäßigen Parameterkonfigurationen ausgelotet werden.

In den vorherigen Abschnitten wurde bereits die Auswahl von RL-Verfahren sowie die Gestaltung von Belohnungsfunktionen diskutiert. Vor diesem Hintergrund konzentriert sich der folgende Abschnitt auf die Anpassung von Hyperparametern von RL-Verfahren, um das Konvergenzverhalten des Agenten

während des Trainings zu verbessern. Es existiert eine Vielzahl von Methoden für die Untersuchung und Optimierung von Hyperparametern. Eine ausführliche Übersicht und Diskussion findet sich bspw. in (Yu und Zhu 2020). Anhand der folgenden vier beispielhaften Verfahren sollen verschiedene Prinzipien und Herangehensweisen für die Untersuchung und Optimierung von Hyperparametern veranschaulicht werden:

- **Rastersuche:** Es handelt sich um ein automatisiertes Verfahren für die strukturierte Erprobung verschiedener Wertekombinationen von Hyperparametern. Voraussetzung ist, dass die möglichen Werte eines jeden Hyperparameters über eine diskrete Menge abgebildet werden können. Kontinuierliche, nicht beschränkte Hyperparameter müssen zunächst auf eine endliche diskrete Untermenge von Werten eingegrenzt werden. Angenommen dass n Hyperparameter in Wertebereiche mit jeweils m Werten diskretisiert werden, so ergibt sich ein $(n \times m)$-Raster von Parameterwerten und m^n möglichen Parameterkombinationen, die vollständig enumeriert werden.
- **Zufallssuche:** Es werden zufällige Kombinationen von Hyperparametern durchprobiert. Im Unterschied zur Rastersuche müssen kontinuierliche Hyperparameter nicht diskretisiert werden. Die Zufallssuche kann im einfachsten Fall über eine Menge von Wahrscheinlichkeitsverteilungen realisiert werden, aus welchen Stichproben für jeden Hyperparameter gezogen werden. Ferner können ebenfalls metaheuristische Verfahren für die Steuerung der Zufallssuche von Hyperparameterkombinationen eingesetzt werden.
- **Bayes'sche Optimierung (BO):** Als eines der gradientenfreien, modellsuchenden RL-Verfahren wurde das Funktionsprinzip der BO bereits in Abschnitt 3.4.1 kurz erläutert. Eine detaillierte Beschreibung der BO ist als Anhang E dem elektronischen Zusatzmaterial beigefügt. Das BO-Verfahren generiert Parameterkombinationen entweder strukturiert, bspw. über eine Rastersuche, oder zufällig, bspw. via Monte-Carlo-Simulation. Haupteigenschaft der BO ist, dass die Evaluation von Parameterkombinationen nicht arbiträr erfolgt, sondern eine Akquisitionsfunktion über den nächsten zu evaluierenden Lösungskandidaten entscheidet. Die Arbeitsweise der Akquisitionsfunktion wird in Anhang E am Beispiel der »Erwarteten Verbesserung« EI dargestellt.
- **Sensitivitätsanalyse:** Das Verfahren dient der Evaluation einzelner Hyperparameter hinsichtlich ihres Einflusses auf den Trainingsprozess. Zu diesem Zweck wird pro Experiment lediglich der Wert eines Hyperparameters variiert und dessen Auswirkung auf den Trainingsprozess beobachtet. Im Kontrast zur Raster- und Zufallssuche handelt es sich bei der Sensitivitätsanalyse um kein Verfahren für die Optimierung von Hyperparametern. Vielmehr dient die

Sensitivitätsanalyse der Entwicklung einer Intuition dafür, welche Hyperparameter im Rahmen einer manuellen Experimentplanung priorisiert untersuchen werden sollten.

Im Folgenden sollen die wichtigsten Hyperparameter von gradientenabhängigen und gradientenfreien RL-Verfahren dargestellt und ihr Einfluss auf den Trainingsprozess diskutiert werden. Zunächst sollen die Hyperparameter von gradientenabhängigen RL-Verfahren betrachtet werden. Tabelle 5.4 stellt die wichtigsten Hyperparameter für gradientenfreie RL-Verfahren vor und diskutiert deren Bedeutung für den Lernprozess. Darüber hinaus existieren eine Vielzahl weiterer RL-verfahrensspezifischer Hyperparameter, deren präzise Erläuterung jedoch über den Rahmen dieser Arbeit hinausgeht. Stellvertretend seien im Folgenden zwei Beispiele angeführt: *(i)* Für alle gradientenabhängigen RL-Verfahren, deren Lernprozess nicht auf N-Step-Bootstrapping basiert, muss die Größe des Erfahrungsspeichers definiert werden. Diese bestimmt die Gesamtmenge an Transitionen, aus welcher ein Agent ein zufälliges Trainingslos generieren kann. *(ii)* Die meisten EPA-Verfahren verwenden stochastische Techniken, um die Exploration des Lösungsraums zu forcieren. Im A2C- und A3C-Algorithmus (stochastische EPA-Verfahren) wird zu diesem Zweck die Entropie der Entscheidungspolitik im Performanzgradienten mitberücksichtigt (Mnih et al. 2016). Über einen benutzerdefinierten Koeffizienten kann die entropie-basierte Exploration feinjustiert werden. Hingegen addieren der DDPG- und TD3-Algorithmus (deterministische EPA-Verfahren) einen zufälligen Term auf berechnete Aktionen, um die Ergebnisse zu verrauschen. Vor diesem Hintergrund kann u. a. die Standardabweichung des Rauschens als weiterer Hyperparameter berücksichtigt werden.

Hinsichtlich der Auswahl von Neuronenmodellen für die versteckte KNN-Architektur sei auf Anhang A im elektronischen Zusatzmaterial verwiesen, in welchem nicht nur die Grundlagen zu KNN, sondern ebenfalls die in Tabelle 5.4 angeführten Neuronenmodelle »Perzeptron«, »LSTM« und »GRU« im Detail erklärt werden. Zusammengefasst wird das Perzeptron für die Modellierung von rein vorwärts gerichteten KNN-Strukturen verwendet. Das Perzeptron kann als Standard für die Modellierung von KNN-basierten RL-Agenten betrachtet werden. Es eignet sich ebenfalls für die meisten Anwendungsfälle in der Produktionsablaufplanung. Eine Ausnahme bildet bspw. die erste Formulierungsvariante von Sequenzierungsproblemen als ML-Aufgabe (Abbildung 5.12 in Abschnitt 5.3.2.2), bei der die Auswahl des nächsten zu produzierenden Auftrags mittels Priorisierung erfolgt. Bei n Aufträgen in einer Warteschlange muss

der Agent zunächst n Aktionen (Prioritäten) berechnen, bevor ein konkreter Auftrag ausgewählt werden kann. Intuitiv erscheint es sinnvoll, dass der Agent bei der Priorisierung eines spezifischen Auftrags in dessen Entscheidung miteinbezieht, welche Charakteristiken die umliegenden Aufträge im Puffer besitzen und wie der Agent diese priorisiert hat. Derartige sequenzielle Zusammenhänge können mit rein vorwärtsgerichteten KNNs nicht analysiert werden. Sogenannte rekurrente neuronale Netze (RNN) sind hierzu imstande, indem sie Informationen aus vorhergehenden Entscheidungen speichern können. Die Neuronenmodelle »LSTM« und »GRU« wurden speziell für die Modellierung von RNN-Strukturen entwickelt, die durch gradientenabhängige Lernverfahren trainiert werden.

Die Auswahl des Optimierungsverfahrens und dessen Parametrisierung repräsentieren ebenfalls benutzerdefinierbare Freiheitsgrade für die Verbesserung des Konvergenzverhaltens. Die meisten RL-Bibliotheken verwenden standardmäßig den Adam-Algorithmus (Kingma und Ba 2015) für das Training von Agenten. Der Grund ist zum einen, dass in Adam viele der methodischen Verbesserungen implementiert sind, die in den vergangenen Jahre für das ursprüngliche stochastische Gradientenabstiegsverfahren eingeführt wurden. Zum anderen überzeugt der Algorithmus durch ein relativ robustes Konvergenzverhalten, weitgehend ungeachtet der gewählten Hyperparameter (Goodfellow et al. 2016). Abgesehen von der Lernrate α ist vor diesem Hintergrund die experimentelle Untersuchung der Parametrisierung des Optimierungsverfahrens als niedrig zu priorisieren, sofern der Adam-Algorithmus für das Agententraining verwendet wird. Nichtsdestotrotz soll am Beispiel des Adam-Algorithmus kurz dargestellt werden, welche zusätzlichen Parameter benutzerdefinierbar sind, um das Agententraining zu beeinflussen. Wie viele stochastische Gradientenabstiegsverfahren verwendet auch Adam die sogenannte Momentum-Technik, um die Lernrate während des Trainingsprozesses adaptiv zu justieren. Das Momentum merkt sich die Gradienten aus vergangenen Trainingsschritten mittels Kumulation. Die Lernrate wird durch das Momentum erster Ordnung (kumuliertes Momentum) und zweiter Ordnung (kumuliertes Momentum zum Quadrat) korrigiert. Über die Koeffizienten β_1 (Standardwert $= 0,9$) bzw. β_2 (Standardwert $= 0,999$) kann der Einfluss des Momentums erster Ordnung bzw. zweiter Ordnung auf die Lernrate angepasst werden. Beim Adam-Algorithmus werden die Parameter des Agenten über den Quotienten aus dem Momentum erster Ordnung und der Wurzel des Momentums zweiter Ordnung aktualisiert. Die Wurzel des Momentums zweiter Ordnung steht im Nenner des Quotienten und kann mitunter Werte annehmen, die dem Wert null entsprechen. Der benutzerdefinierbare Term ϵ (Standardwert $= 10^{-8}$) wird zu diesem Zweck dem Nenner hinzuaddiert, um die numerische Stabilität

des Adam-Algorithmus zu gewährleisten. Ferner erlauben einige Implementierungen des Adam-Algorithmus, z. B. die PyTorch-Implementierung, eine zusätzliche Regularisierung der Gradienten, wodurch das Risiko einer Überanpassung der Agentenparameter an die Trainingsdaten reduziert wird. Mithilfe des Koeffizienten λ (Standartwert $= 0$ – keine Regularisierung) kann die Ausprägung der Regularisierung benutzerspezifisch angepasst werden.

Entsprechend ihrer Vielfalt und Individualität sind ebenfalls die Hyperparameter für jedes gradientenfreie RL-Verfahren spezifisch. Vor diesem Hintergrund ist eine verfahrensübergreifende Darstellung der Hyperparameter nicht zweckmäßig. Analog zu den theoretischen Grundlagen in Abschnitt 3.4 werden stattdessen die Hyperparameter des NEAT-Algorithmus diskutiert, der im Rahmen dieser Arbeit als Stellvertreter der gradientenfreien RL-Verfahren untersucht wird. NEAT ist für die stellvertretende Diskussion der Hyperparameter von gradientenfreien RL-Verfahren prädestiniert, da der Algorithmus eine große Bandbreite verschiedenartiger Stellgrößen aufweist, welche sowohl die stochastische Suche nach einem geeigneten Agentenmodell als auch die stochastische Suche nach geeigneten Modellparametern adressieren. Tabelle 5.5 enthält eine Auflistung aller Hyperparameter von NEAT gemäß der Implementierung NEAT-Python. Ferner beinhaltet die Tabelle beispielhafte Werte für jeden Hyperparameter. Die Beispiele repräsentieren keine Empfehlungen, sondern dienen lediglich dem Zweck, ein Gespür zu vermitteln, in welchen ungefähren Wertebereichen sich die verschiedenen Hyperparameter ansiedeln. Im Folgenden soll die Bedeutung der einzelnen Hyperparameter für das Agententraining dargelegt werden.

Tabelle 5.5 – Sektion »Allgemeine Parameter«

Die Hyperparameter in dieser Sektion steuern insbesondere die Laufzeit von NEAT sowie die Exploration des Lösungsraums. Die Anzahl der Generationen entspricht der Anzahl der Iterationen, die NEAT durchläuft, bevor das Verfahren terminiert. Hierbei wird in jeder Iteration eine Anzahl von Lösungskandidaten entsprechend des Parameters »Populationsgröße« erzeugt und evaluiert. Eine hohe Anzahl von Generationen und eine hohe Populationsgröße gehen mit einer umfangreicheren Exploration des Lösungsraums einher, erhöhen jedoch ebenfalls die Laufzeit des Algorithmus. Das Fitnesskriterium bestimmt die Art und Weise, wie die Fitness der gesamten Population, basierend auf der Fitness einzelner Genome, berechnet wird. Wahlweise kann hierfür die minimale, maximale oder mittlere Fitness über alle Genome ermittelt werden. Der Fitnessschwellenwert ist ein optionales Terminationskriterium. Sofern die Fitness der Population den Fitnessschwellenwert überschreitet, endet der Algorithmus. Hieraus folgt, dass für die Bildung der Fitnessfunktion die Zielfunktion mit dem Faktor -1

Tabelle 5.4 Wichtige Hyperparameter für gradientenabhängige bestärkende Lernverfahren

Hyperparameter	Bedeutung
Episodenanzahl	Anzahl der Durchläufe der Agentenumgebung von $t = 0, \ldots, T$. Je höher die Episodenanzahl, desto intensiver kann der Agent den Lösungsraum untersuchen. Demgegenüber steigt mit der Anzahl der Episoden die Laufzeit des Trainings. Eine geeignete Episodenanzahl kann mithilfe einer Konvergenzanalyse ermittelt werden.
Lernrate α	Schrittweite der gradientenabhängigen Anpassung der Agentenparameter (vgl. Formel (3.9)). Große Lernraten gehen mit einer intensiveren Exploration des Lösungsraums einher. Demgegenüber werden (lokal) optimale Regionen nur grobmaschig abgetastet und somit weniger intensiv ausgelotet. Für kleine Lernraten gilt das Gegenteil. *Richtwert* $= 0,001$ (vgl. Kingma und Ba 2015).
Diskontierungsrate γ	Diskontiert die zukünftige (kumulierte) Belohnung in der Bellman-Gleichung (siehe Formeln (3.1) to (3.6) Eine große Diskontierungsrate bedeutet, dass etwaige zukünftige Belohnungen bei der Aktionswahl stärker berücksichtigt werden. Der Agent handelt auf diese Weise weitsichtiger und nimmt kurzfristig schlechtere Belohnungen in Kauf. Für kleine Diskontierungsraten gilt das Gegenteil. *Richtwert* $= 0,99$ (vgl. Mnih et al. 2015; Lillicrap et al. 2015).
Trainingslosgröße	Anzahl der Transitionen, die pro Trainingsschritt verarbeitet werden. Große Trainingslose resultieren in einer schnelleren Konvergenz zu einer (lokal) optimalen Entscheidungspolitik, reduzieren jedoch u. U. die Generalisierungsfähigkeit des Agenten. Für kleine Trainingslosgrößen gilt das Gegenteil.

(Fortsetzung)

Tabelle 5.4 (Fortsetzung)

Hyperparameter	Bedeutung
Versteckte Architektur des KNN-basierten Agenten	Abschnitt 5.3.2 diskutiert den Entwurf von Agenten nur hinsichtlich der Eingabe- und Ausgabeschicht abhängig von der Definition der ML-Aufgabe. Hingegen kann die Gestaltung der versteckten Schichten für das Training optimiert werden. Diesbezüglich können u. a. folgende Größen variiert werden: · Anzahl der versteckten Schichten, · Anzahl der Neuronen je versteckter Schicht, · Neuronenmodell (z. B. Perzeptron, LSTM, GRU), · ggf. die Aktivierungsfunktion (z. B. Sigmoid). Je mehr versteckte Schichten und / oder Neuronen je versteckter Schicht der Agent besitzt, desto spezifischere Lösungsstrategien kann dieser erlernen. Gewöhnlich findet der Agent auf diese Weise bessere Lösungen für die Trainingsdaten. Demgegenüber besteht das Risiko einer Überanpassung der Agentenparameter, d. h., dass der Agent seine Strategie auf die Trainingsdaten zuschneidet, zu Lasten seiner Generalisierungsfähigkeit.
Optimierungsverfahren	Steuert die gradientenabhängige Aktualisierung der Agentenparameter. Es handelt sich um ein stochastisches Gradientenabstiegsverfahren bzw. um eine Abwandlung davon (z. B. Adam, RMSProp). Die Parameter des Optimierungsverfahrens können ebenfalls als experimentierbare Hyperparameter berücksichtigt werden. Zu diesen zählt die bereits oben gelistete Lernrate α. *Richtwert* = *Adam-Algorithmus* (vgl. Dhariwal et al. 2017; Hill et al. 2018).

multipliziert werden muss, sofern das zugrundeliegende Optimierungsproblem zu minimieren ist. Der Grund ist, dass NEAT-Python stets bestrebt ist die Fitness der Population zu maximieren. Ferner existiert ein weiteres optionales Terminationskriterium. Gemäß diesem endet der Algorithmus vorzeitig, sofern die Population aufgrund von stagnierenden Fitnesswerten ausstirbt und der entsprechende Merker auf FALSCH gesetzt wurde. Besitzt der Merker hingegen den Wert WAHR, wird im Fall des Aussterbens eine neue Population erstellt und evolviert, sofern die maximale Anzahl von Generationen noch nicht erreicht wurde.

Tabelle 5.5 – Sektion »Genom-Parameter«

Die Hyperparameter in dieser Sektion definieren zum einen die Größe des untersuchbaren Lösungsraums, zum anderen die Schrittweite, mit welcher der Lösungsraum abgetastet wird. Die Größe des Lösungsraums resultiert aus den Wertebereichen und -mengen der verschiedenen Genattribute eines Genoms, aus welchen wiederum die Struktur und Parameter des jeweiligen KNN resultieren. Synaptische Gewichte sowie neuronale Bias-Terme und Reaktivitätswerte[1] werden über kontinuierliche Wertebereiche abgebildet und durch benutzerdefinierte Minima und Maxima begrenzt. Die möglichen Aktivierungs- und Netzeingabefunktionen werden wiederum als diskrete Wertemengen durch die Benutzer*innen definiert. Die Spannbreiten der Wertebereiche und die Kardinalitäten der Wertemengen bestimmen maßgeblich die Größe des Lösungsraums. Einen weiteren Einfluss auf die Größe des Lösungsraum besitzt der Merker für rekurrente synaptische Verbindungen. Sofern der Merker auf WAHR gesetzt wird, pflegen Neuronen nicht ausschließlich vorwärtsgerichtete Verbindungen zu nachgelagerten Neuronen, sondern ebenfalls rückwärtsgerichtete (rekurrente) Verbindungen zu sich selbst und / oder zu vorgelagerten Neuronen. Im Allgemeinen birgt die Vergrößerung des Lösungsraums das Potenzial, bessere Lösungen zu ermitteln. Gleichermaßen kann sich das Konvergenzverhalten verschlechtern, da aus der Vergrößerung des Lösungsraums ebenfalls eine höhere Anzahl qualitativ minderwertiger Lösungen resultieren kann. Die Schrittweite zur Abtastung des

[1] Es handelt sich um einen NEAT-spezifischen Koeffizienten, der für jedes Neuron individuell ist und das Ergebnis der Netzeingabefunktion zusätzlich gewichtet, bevor es in die Aktivierungsfunktion eingeht.

Tabelle 5.5 Hyperparameter des NEAT-Algorithmus der Programmbibliothek NEAT-Python (vgl. McIntyre et al. 2017b) stellvertretend für gradientenfreie bestärkende Lernverfahren

Hyperparameter		**Datentyp und -struktur**	**Beispielwerte**
Allgemeine Parameter			
Anzahl Generationen		Ganzzahlig	100
Populationsgröße		Ganzzahlig	250
Fitnesskriterium der Population		Funktion	Minimum
Fitnessschwellenwert		Gleitkommazahl	0,0
Merker für das Zurücksetzen bei aussterbender Population		Boolean	FALSCH
Genom-Parameter			
Merker für rekurrente synaptische Verbindungen		Boolean	FALSCH
Merker für einzelne strukturelle Mutationen		Boolean	WAHR
Merker zur Absicherung struktureller Mutationen		Boolean	WAHR
Initiale Verbindung von Neuronen		String	*„unconnected“*
Initiale Anzahl versteckter Neuronen		Ganzzahlig	3
Wahrscheinlichkeit für die Erstellung eines neuen Neurons		Gleitkommazahl	0,15
Wahrscheinlichkeit für die Eliminierung eines Neurons		Gleitkommazahl	0,1
Wahrscheinlichkeit für die Erstellung einer neuen Synapse		Gleitkommazahl	0,4
Wahrscheinlichkeit für die Eliminierung einer Synapse		Gleitkommazahl	0,3
Neuron Bias-Term Reaktivität Synapse Synaptisches Gewicht	Initiale Mittelwerte	Gleitkommazahl	0,0
	Initiale Standardabweichungen	Gleitkommazahl	0,5
	Mutationswahrscheinlichkeiten	Gleitkommazahl	0,8
	Mutationsstandardabweichungen	Gleitkommazahl	0,5
	Ersatzwahrscheinlichkeiten	Gleitkommazahl	0,05
	Minimale Werte	Gleitkommazahl	−40,0
	Maximale Werte	Gleitkommazahl	40,0
Initiale Aktivierungsfunktion		Funktion	ReLU

(Fortsetzung)

Tabelle 5.5 (Fortsetzung)

Hyperparameter	**Datentyp und -struktur**	**Beispielwerte**
Optionen für Aktivierungsfunktion	Liste von Funktionen	[Sigmoid, ReLU]
Mutationswahrscheinlichkeit für Aktivierungsfunktion	Gleitkommazahl	0,5
Initiale Netzeingabefunktion	Funktion	Summe
Optionen für Netzeingabefunktion	Liste von Funktionen	[Summe, Produkt]
Mutationsrate für Netzeingabefunktion	Gleitkommazahl	0,2
Initialer Status einer Synapse	Boolean	WAHR
Mutationswahrscheinlichkeit eines synaptischen Status	Gleitkommazahl	0,05
Spezies-Parameter		
Fitnesskriterium der Spezies	Funktion	Mittelwert
Maximale Anzahl stagnierender Generationen	Ganzzahlig	15
Anzahl elitärer Spezies	Ganzzahlig	0
Anzahl elitärer Genome pro Spezies	Ganzzahlig	0
Anteil reproduktionsfähiger Genome pro Spezies	Gleitkommazahl	0,2
Mindestanzahl Genome pro Spezies nach Reproduktion	Ganzzahlig	2
Kompatibilitätsschwellenwert	Gleitkommazahl	4,0
Kompatibilitätskoeffizient für neuronale Gene	Gleitkommazahl	1,0
Kompatibilitätskoeffizient für synaptische Gene	Gleitkommazahl	1,0

Lösungsraums wird implizit durch die Mutationswahrscheinlichkeiten und -standardabweichungen der neuronalen und synaptischen Attribute sowie durch die Wahrscheinlichkeiten für das Erstellen bzw. Eliminieren von Neuronen und Synapsen bestimmt. Im Fall der Mutation eines synaptischen Gewichts, eines neuronalen Bias-Terms oder eines neuronalen Reaktivitätswerts wird mithilfe der jeweiligen Mutationsstandardabweichung eine positive oder negative Zufallszahl aus einer um den Wert null zentrierten Normalverteilung gezogen und auf das jeweilige Attribut addiert. Im Fall der Mutation einer Aktivierungs- oder Netzeingabefunktion wird aus der entsprechenden Menge eine neue Aktivierungs- bzw. Netzeingabefunktion per Zufallsstichprobe gezogen. Sofern ein neues Neuron

oder eine neue Synapse erstellt wird, werden die jeweiligen initialen Hyperparameter angewandt, um das neue Gen zu parametrisieren. In NEAT können Synapsen zudem aktiv oder inaktiv sein. Zweiteres bewirkt, dass der Wert, der über die inaktive Synapse gesendet und gewichtet wird, nicht von der Netzeingabefunktion des am Ausgang angeschlossenen Neurons berücksichtigt wird. Je größer die Mutationswahrscheinlichkeiten und -standardabweichungen, desto mehr neigt NEAT dazu den Lösungsraum großflächig zu untersuchen. Zu viele gleichzeitige oder zu starke Mutationen steigern demgegenüber das Risiko, dass der Algorithmus einer lokal optimalen Region zu schnell entweicht, ohne deren Lösungsgüte ausreichend ausgelotet zu haben. Geringe Mutationswahrscheinlichkeiten und -standardabweichungen begünstigen, dass eine lokal optimale Region intensiver ausgelotet werden kann, da sich das Genom allmählich und partieller verändert. Demgegenüber kann es länger dauern, bis NEAT zu einer guten Lösung konvergiert. Das Konvergenzverhalten des Algorithmus kann des Weiteren durch den Merker für einzelne strukturelle Mutationen sowie durch den Merker zur Absicherung struktureller Mutationen feinjustiert werden. Sofern der erstgenannte Merker auf WAHR gesetzt wird, führt NEAT pro Generation maximal eine strukturelle Mutation an jedem Genom durch. KNN-Strukturen mutieren somit langsamer und kontrollierter, wodurch das Ausloten von lokalen optimalen Regionen begünstigt wird. Falls der zweitgenannte Merker auf WAHR gesetzt wird, werden zwei Techniken aktiviert, um strukturelle Mutationen zu forcieren. Erstens wird ein neues synaptisches Gen anstelle des neuronalen Gens hinzugefügt, sofern NEAT ein weiteres neuronales Gen zu einem Genom hinzufügen will, das noch keine synaptischen Gene besitzt. Zweitens wird die bereits existierende Synapse aktiv geschaltet, wenn NEAT eine synaptische Verbindung herstellen möchte, die bereits existiert. Abschließend bietet die Sektion einige Hyperparameter, um die Generierung von Startlösungen zu steuern. Hierunter zählen die initialen Mittelwerte und Standardabweichungen für die Parametrisierung von synaptischen Gewichten sowie von neuronalen Bias-Termen und Reaktivitätswerten. Wahlweise kann für jede Startlösung eine fixe Anzahl versteckter Neuronen definiert werden. Zudem können Benutzer*innen festlegen, ob und wie die Neuronen eines jeden Genoms der Anfangspopulation synaptisch verbunden sind (z. B. nicht verbunden, voll verbunden oder zufällig verbunden).

Tabelle 5.5 – Sektion »Spezies-Parameter«

Die Hyperparameter in dieser Sektion beeinflussen ebenfalls die Exploration des Lösungsraums sowie die Ausbeutung lokaler Optima. Wie bereits in Abschnitt 3.4.2 beschrieben, unterteilt NEAT die Gesamtpopulation in Spezies, um Genomen die Möglichkeit einzuräumen, die Parameter von topologischen

Innovationen über einige Generationen zu optimieren. Hierbei ist es möglich, dass eine komplette Spezies vollständig ausstirbt, sofern der Fitnesswert der jeweiligen Spezies über eine benutzerdefinierte Anzahl von Generationen stagniert. Das ebenfalls durch Benutzer*innen definierbare Fitnesskriterium für Spezies funktioniert analog zum Fitnesskriterium der Population. Je höher die Anzahl der Generationen ist, über welche die Fitness einer Spezies stagnieren darf, desto tiefer kann NEAT bestimmte Lösungsregionen untersuchen. Demgegenüber reduziert sich die Fähigkeit des Algorithmus, den Lösungsraum zeiteffizient in der Breite zu untersuchen. Benutzer*innen können des Weiteren jeweils eine fixe Anzahl von elitären Spezies und von elitären Genomen festlegen. Angenommen es werden n elitäre Spezies und m elitäre Genome je Spezies definiert, so sind die n besten Spezies vom Aussterben durch Stagnation ausgenommen, während die m besten Genome einer Spezies von jeglichen Mutationen ausgeschlossen werden. Auf diese Weise wird zum einen verhindert, dass während der Laufzeit die komplette Population ausstirbt. Zum anderen wird abgesichert, dass die bisher besten gefundenen Lösungen bewahrt bleiben. Eine hohe Anzahl elitärer Spezies und Genome mindert die Fähigkeit des Algorithmus, den Lösungsraum großflächig zu durchkämmen. Der Anteil reproduktionsfähiger Genome regelt, wie viel Prozent der besten Genome je Spezies für die Erzeugung der nächsten Generation gekreuzt werden können. Ferner muss definiert werden, wie viele Genome mindestens einer Spezies angehören, nachdem die Population evolviert wurde. Der Hauptindikator für die Zugehörigkeit eines Genoms zu einer Spezies stellt der Kompatibilitätsschwellenwert dar. Ein Genom gehört nur dann zu einer Spezies, wenn die genetische Distanz zwischen dem Genom und dem Speziesvertreter den Kompatibilitätsschwellenwert nicht überschreitet. Über die Kompatibilitätskoeffizienten kann der Einfluss der neuronalen und synaptischen Distanz auf die genetische Gesamtdistanz feinjustiert werden. Die entsprechenden Berechnungsvorschriften wurden bereits in Abschnitt 3.4.2 dargelegt (Formeln (3.21–23)).

5.4 Zusammenfassung der Methode

In diesem Kapitel wurde das Hauptartefakt der vorliegenden Arbeit beschrieben – eine Methode zum Einsatz von bestärkenden Lernverfahren für die Produktionsablaufplanung. Basierend auf einer Analyse der aktuell etablierten Produktionsablaufplanung gemäß des Aachener PPS-Modells wurden in Abschnitt 5.1 zunächst acht Anforderungen an die zu entwickelnde Methode

formuliert. Die erste wesentliche Anforderung (A1), dass die Methode zur Verwendung von RL-trainierten Agenten für die Produktionsablaufplanung anleiten soll, wurde in Abschnitt 5.2 adressiert. Der zweiten wesentlichen Anforderung (A2), dass die Methode die Konzeption und Implementierung von RL-Verfahren für die Produktionsablaufplanung beschreiben soll, wurde in Abschnitt 5.3 Rechnung getragen.

Um Anforderung A1 zu berücksichtigen, wurden in Abschnitt 5.2 die Grundzüge der agentenbasierten Produktionsablaufsteuerung hergeleitet. Die in diesem Kontext vorgestellten Prozessmodelle für die Ressourcenbelegungsplanung (Abbildung 5.4 in Abschnitt 5.2.1) sowie für die Reihenfolgeplanung und Losbildung (Abbildung 5.5 in Abschnitt 5.2.2) erlauben eine echtzeitfähige, datenbasierte und adaptive Produktionsablaufplanung (Anforderung A3). Die Begründung, inwiefern die agentenbasierte Produktionsablaufsteuerung diese drei Eigenschaften erfüllt, wurde einleitend in Abschnitt 5.2 skizziert. Des Weiteren berücksichtigen beide Prozessmodelle, dass der Mensch die letzte Entscheidungsinstanz darstellt und dass ein RL-trainierter Agent seine Entscheidungspolitik auf Basis menschlicher Expertise anpassen kann (Anforderung A4). Konkret erlauben beide Prozessmodelle, dass Agentenentscheidungen durch den Menschen evaluiert und verworfen und dass Alternativentscheidungen als Lernsignal an den Agenten zurückgespielt werden können. Ferner wurden zu Beginn von Abschnitt 5.2 zwei Modelle für die Beschreibung von Produktionsbereichen vorgestellt (Abbildung 5.3), durch deren Verknüpfung beliebige Probleme der Produktionsablaufplanung abgebildet werden können (Anforderung A5). Sowohl die Prozessmodelle als auch die Modelle zur Abbildung von Produktionsbereichen sind so gestaltet, dass agentenbasierte PPS-Entscheidungen in jedem Fall die Reihenfolge- und Losgrößenplanung integrieren. Das zweite Produktionsbereichsmodell erlaubt darüber hinaus eine integrierte Betrachtung der Ressourcenbelegungs-, Reihenfolge- und Losgrößenplanung (Anforderung A6).

Hinsichtlich der Erfüllung von Anforderung A2 wurde innerhalb von Abschnitt 5.3 ein siebenphasiges Vorgehensmodell zur Projektierung und Entwicklung von agentenbasierten Produktionsablaufsteuerungen entwickelt. Die Phasen des Vorgehensmodells umfassen *(i)* die Implementierung von Agentenumgebungen als DES-Modelle, *(ii)* die Definition von ML-Aufgaben und die resultierende Gestaltung von Agenten, *(iii)* die Integration und Inbetriebnahme von Agenten und Agentenumgebungen, *(iv)* die Auswahl und Implementierung von RL-Verfahren, *(v)* die Gestaltung von Belohnungsfunktionen sowie *(vi)* das Training von Agenten. Im Rahmen des Vorgehensmodells wird Anforderung A7 (Durchsuchung des Lösungsraums für alternative Lösungskandidaten) für

gradientenabhängige RL-Verfahren dadurch erfüllt, dass der Programmablaufplan für die Aktionsermittlung durch Vorwärtspropagierung (Abbildung 5.18 in Abschnitt 5.3.3.1) verschiedene zufallsbasierte Explorationsstrategien berücksichtigt. Die Exploration des Lösungsraums wird des Weiteren dadurch begünstigt, dass die in Abschnitt 5.2 vorgestellten Prozessmodelle für die Ressourcenbelegungsplanung sowie für die Reihenfolgeplanung und Losbildung eine manuelle oder heuristische Bestimmung von alternativen Produktionsablaufentscheidungen erlauben, wenn ein Agent keine anforderungsgerechte Entscheidung getroffen hat. Hinsichtlich der gradientenfreien RL-Verfahren ist die Erfüllung von Anforderung A7 trivial, da diese grundsätzlich den Lösungsraum unabhängig von Gradienten und i. d. R. zufallsbasiert durchsuchen. Zu guter Letzt wird ebenfalls Anforderung A8 (Skalierbarkeit von RL-trainierten Agenten auf unterschiedlich großen Probleminstanzen) durch das Vorgehensmodell adressiert. So erlauben alle in Abschnitt 5.3.2 vorgestellten Varianten zur Formulierung von Teilproblemen der Produktionsablaufplanung als ML-Aufgaben, dass Agentenentscheidungen für jede Systemlast berechnet werden können, ungeachtet der Anzahl von Aufträgen, die während des Trainings analysiert wurden.

Zusammenfassend erfüllt die Methode somit alle der in Abschnitt 5.1 postulierten Anforderungen, um Entscheidungsunterstützungssysteme für die Produktionsablaufplanung zu entwickeln, welche ebenfalls die Entscheidungsfindung unter Berücksichtigung eines dynamischen stochastischen Auftragshorizonts sowie enger Zeitfenster begünstigen.

Evaluation der entwickelten Methode

6

Die in Kapitel 5 vorgestellte Methode wurde mit der Zielstellung entworfen, Produktions- und Logistikplaner*innen sowie Wirtschaftsinformatiker*innen bei der Entwicklung und Integration von RL-trainierten agentenbasierten Produktionsablaufsteuerungen zu unterstützen. Die resultierenden Entscheidungssysteme sollen insbesondere eine kurzfristige operative Produktionsablaufplanung begünstigen, unter Berücksichtigung eines dynamischen stochastischen Auftragshorizonts sowie enger Zeitfenster für die Entscheidungsfindung. Im Folgenden soll anhand von drei Fallstudien nachgewiesen werden, dass durch die konsequente Anwendung der Methode Produktionsablaufsteuerungen entwickelt werden können, welche die beschriebenen Eigenschaften erfüllen. Zudem sollen anhand von jeder Fallstudie unterschiedliche Konzepte, die im Rahmen der Methode entwickelt wurden, evaluiert werden. Hierzu zählen insbesondere:

- die Produktionsbereichsmodelle (Abbildung 5.3 in Abschnitt 5.2) sowie die Prozessmodelle der Auftragserzeugung (Abbildung 5.8 in Abschnitt 5.3.1.1), des Auftragsflusses (Abbildung 5.9 in Abschnitt 5.3.1.2) sowie der Produktionsressourcen (Abbildung 5.10 in Abschnitt 5.3.1.3) für den Entwurf von Agentenumgebungen;
- die Konzepte zur Formulierung von Produktionsablaufentscheidungen als ML-Aufgaben aus Abschnitt 5.3.2;
- die für Probleme der Produktionsablaufplanung adaptierte OpenAI-Gym-Schnittstelle aus Abschnitt 5.3.3;

Ergänzende Information Die elektronische Version dieses Kapitels enthält Zusatzmaterial, auf das über folgenden Link zugegriffen werden kann https://doi.org/10.1007/978-3-658-41751-2_6.

S. Lang, *Methoden des bestärkenden Lernens für die Produktionsablaufplanung*,
https://doi.org/10.1007/978-3-658-41751-2_6

- die eigenentwickelte Trainingsprozedur für selbstimplementierte gradientenabhängige RL-Verfahren (Abbildung 5.25 in Abschnitt 5.3.4);
- das Konzept für die Entwicklung von Belohnungsfunktionen aus Abschnitt 5.3.5.

Hingegen wird im Rahmen dieser Arbeit auf ausführliche Hyperparameterstudien für die eingesetzten RL-Verfahren zum Zweck der Optimierung des Agententrainings verzichtet. Für jede Fallstudie wurden geeignete Hyperparameter für das Agententraining durch eine manuelle Suche ermittelt. Der Verzicht auf umfassende Hyperparameterstudien dient nicht nur der Eingrenzung dieser Arbeit, sondern soll ebenfalls als Indiz dafür dienen, dass die vorgestellten Verfahren bereits ohne großen experimentellen Aufwand robust zu guten Ergebnissen konvergieren. Es ist anzunehmen, dass mithilfe von ausführlichen Hyperparameterstudien die Lösungsgüte weiter verbessert werden kann.

6.1 Flexible-Job-Shop-Problem mit flexibler Operationsplanung

Die erste Fallstudie adressiert die agentenbasierte Produktionsablaufsteuerung in einem Flexible-Job-Shop (FJS) -Problem mit flexibler Operationsplanung. Der DQN-Algorithmus wird für das Agententraining eingesetzt. Wie in den theoretischen Grundlagen unter Abschnitt 2.3.1.1 dargelegt, ist ein Job-Shop-Problem die mathematische Formalisierung des Fertigungstyps »Werkstattfertigung«. In einer Werkstattfertigung sind die Produktionsressourcen nicht in Reihe aufgestellt, sondern als Inseln organisiert. Ferner existiert keine feste Verkettung der Produktionsressourcen durch Fördertechnik. Aufträge werden stattdessen durch Flurfördergeräte oder durch den Menschen selbst transportiert. Die Werkstattfertigung birgt somit ein hohes Maß an Flexibilität für die Produktion von Aufträgen. Heutzutage gewinnt das Prinzip der Werkstattfertigung wieder zunehmend an wissenschaftlicher Relevanz, unter anderem aufgrund des Trends zur »Losgröße 1« oder durch moderne Fabrikkonzepte wie die Matrix-Produktion (Bauernhansl et al. 2014; Greschke et al. 2014). Das Vorgehen und die Ergebnisse der folgenden Fallstudie basieren im Wesentlichen auf einer vorausgegangenen Publikation (Lang et al. 2020a), die im Rahmen dieser Arbeit entstanden ist. Es sei jedoch darauf hingewiesen, dass die Experimente in dieser Arbeit unter anderen Parametern und unter Verwendung einiger zusätzlicher RL-Techniken wiederholt wurden.

Vor diesem Hintergrund existieren unwesentliche Abweichungen zwischen den in Abschnitt 6.1.3 und den in (Lang et al. 2020a) dokumentierten Ergebnissen.

6.1.1 Problembeschreibung

Das in diesem Abschnitt behandelte FJS-Problem ist aus (Rajkumar et al. 2010) entnommen und wurde das erste Mal von Baykasoğlu und Özbakır (2009) beschrieben. Es handelt sich um ein FJS-Problem gemäß der Definition von bspw. Xie et al. (2019) oder Zhang et al. (2019). Dementsprechend ist das Problem als FJm charakterisiert, d. h., dass Fertigungsoperationen auf allen Ressourcen, unabhängig vom Produktionsbereich, bearbeitet werden können. Spezifischer kann das Problem als »Total Flexible-Job-Shop« klassifiziert werden, weil nicht nur einige, sondern alle Fertigungsoperationen von allen Produktionsressourcen bearbeitet werden können.

Das betrachtete Problem wird durch eine Menge von Aufträgen J und Produktionsressourcen M beschrieben. Die genaue Anzahl von Aufträgen $|J|$ und Produktionsressourcen $|M|$ variieren zwischen den Probleminstanzen. Alle Aufträge $j = (1, \ldots, |J|)$ sind zum Zeitpunkt null verfügbar, besitzen jeweils eine Fertigstellungsfrist d_j und müssen eine bestimmte Anzahl von Fertigungsoperationen durchlaufen, bevor sie das System verlassen.

Hinsichtlich der zu durchlaufenden Fertigungsoperationen wird das Problem zusätzlich durch eine flexible Operationsplanung erschwert, d. h., dass gewisse Freiheitsgrade hinsichtlich der Art und Durchführung von Fertigungsoperationen bestehen. Zu diesem Zweck müssen die mathematischen Formalismen, die in Abschnitt 2.3.1 eingeführt wurden, erweitert werden. Jeder Auftrag j besitzt eine Menge alternativer Operationspläne OP_j. Unabhängig von der Probleminstanz ist im vorliegenden Problem jeder Auftrag durch genau vier Operationspläne charakterisiert ($\left|OP_j\right| = 4 \forall j = (1, \ldots, |J|)$ bzw. $OP_{j,v} \forall v = (1, \ldots, 4) \forall j = (1, \ldots, |J|)$). Jeder Operationsplan $OP_{j,v}$ eines Auftrags J_j beschreibt wiederum eine geordnete Menge von Operationen. Hierbei repräsentiert $o_{j,v}$ Operation o von Operationsplan v von Auftrag j. Die Anzahl von Operationen eines Operationsplans variiert zwischen zwei und vier Operationen. Die Operationen eines Operationsplans müssen entsprechend ihrer Reihenfolge innerhalb des Operationsplans abgearbeitet werden. Jede Operation kann auf jeder Produktionsressource bearbeitet werden, wobei die Bearbeitungszeit für eine Operation zwischen den verschiedenen Produktionsressourcen variiert. Vor diesem Hintergrund definiert $o_{i,j,v}$ die Bearbeitungszeit der o-ten Operation des Operationsplans v von Auftrag j auf Produktionsressource i. Die Bearbeitungszeiten

Tabelle 6.1 Aggregierte Darstellung der FJS-Probleminstanzen (in Anlehnung an Lang et al. 2020a)

Probleminstanz	$\|J\|$	$\|M\|$	Mögliche d_j	Mögliche $o_{i,j,v}$
1 (5J×5M)	5	5	80, 100, 110, 120	5, 7, 10, 11, 12, 13, 14, 15, 17, 18, 19, 20, 21, 22, 23, 24, 26, 28, 30, 31, 38, 40, 42, 46, 47, 52, 55, 56, 57, 58, 60, 62, 64, 65, 66, 68, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 84, 85, 87, 88, 90, 91, 92, 93, 94, 95, 96, 97, 98
2 (8J×8M)	8	8		
3 (10J×5M)	10	5		
4 (20J×5M)	20	5	400, 500, 600, 700	

von Operationen sind hierbei lediglich auftrags- und ressourcenspezifisch, nicht jedoch spezifisch für Operationspläne. Somit gilt $o_{i,j,v} \in O_j$. Tabelle 6.1 beinhaltet eine aggregierte Sicht auf die Probleminstanzen. Eine vollständige Übersicht zu den Problemdaten ist als Anhang H dem elektronischen Zusatzmaterial beigefügt. Tabelle H-1 in Anhang H informiert über alle Operationspläne von allen Aufträgen aller Probleminstanzen. Tabelle H-2 in Anhang H enthält die Fertigstellungsfristen sowie die genauen Operationszeiten aller Aufträge auf allen Ressourcen für alle Probleminstanzen.

Als weitere Restriktion gilt, dass jede Produktionsressource nur einen Auftrag zur gleichen Zeit bearbeiten kann. Die Pufferkapazitäten vor den Produktionsressourcen werden als unendlich angenommen. Das Ziel ist die Ermittlung eines Produktionsablaufplans, in dem $|J|$ Operationspläne ausgewählt und $\sum_{j=1}^{|J|} |o_{j,v}|$ Operationen zu Produktionsressourcen alloziert werden. Die Güte eines Produktionsablaufplans soll sowohl anhand dessen Gesamtdauer C_{max} als auch anhand der resultierenden Gesamtverspätung über alle Aufträge T bewertet werden.

6.1.2 Anwendung des DQN-Algorithmus zur Lösung des Problems

Im Folgenden wird dargelegt, wie der DQN-Algorithmus zum Zweck einer agentenbasierten Produktionsablaufsteuerung für das vorliegende FJS-Problem eingesetzt wird.

Entwurf der Agentenumgebung

Für die Modellierung der Agentenumgebung kann die geschilderte Problemstellung auf das in Abschnitt 5.2 eingeführte Modell für Produktionsbereiche

mit lokalen Warteschlangen (Abbildung 5.3 (a)) reduziert werden. Gemäß diesem Modell kann die Problemstellung mithilfe der in Abschnitt 5.3.1 definierten Prozessmodelle als DES-basierte Agentenumgebung implementiert werden. Im Rahmen dieser Arbeit erfolgt die Implementierung mit der Python-Bibliothek Salabim. Zur Veranschaulichung der Implementierung zeigt Abbildung 6.1 einen Auszug aus dem animierten Simulationsmodell der 8 J $\times$ 8M-Probleminstanz.

Definition der maschinellen Lernaufgaben

Um eine RL-agentenbasierte Produktionsablaufsteuerung zu realisieren, wird die Problemstellung im Folgenden auf zwei Entscheidungsprobleme heruntergebrochen: *(i)* die Auswahl von Operationsplänen für Aufträge vor deren Freigabe, *(ii)* die Zuordnung von Aufträgen zu Produktionsressourcen. Die Teilprobleme *(i)* und *(ii)* sollen jeweils mithilfe unterschiedlicher RL-trainierter Agenten gelöst werden.

Es sei angemerkt, dass es ebenfalls möglich ist die Reihenfolgebildung vor jeder Produktionsressource mittels eines Agenten zu steuern. Im Rahmen dieser Fallstudie wird diese Möglichkeit jedoch vernachlässigt. Der Grund ist zum einen, dass die erzielten Ergebnisse der RL-Agenten mit dem Lösungsansatz von Rajkumar et al. (2010) verglichen werden, der ebenfalls keine Reihenfolgeplanung berücksichtigt. Stattdessen wird in (Rajkumar et al. 2010) die Reihenfolgebildung über die EDD-Prioritätsregel gesteuert. Zum anderen bilden sich im betrachteten System keine signifikanten Warteschlangen, da die meisten Probleminstanzen, im Vergleich zur Anzahl zu produzierender Aufträge, eine hohe Anzahl von Produktionsressourcen aufweisen. Im Unterschied zu (Rajkumar et al. 2010) werden Auftragsreihenfolgen während des Agententrainings nicht über die EDD-, sondern über die FIFO-Prioritätsregel gesteuert, da eine zeitbasierte Belohnungsfunktion für die Bewertung von Agentenentscheidungen verwendet wird und die FIFO-Regel eine wesentlich einfachere Vorausberechnung von Fertigstellungszeitpunkten erlaubt.

Für beide Agenten wird die Formulierung von Allokationsproblemen als ML-Aufgabe gemäß Abbildung 5.11 (Abschnitt 5.3.2.1) angewandt. Aufgrund der Tatsache, dass jeder Auftrag die gleiche Anzahl an Operationsplänen besitzt, kann deren Auswahl ebenfalls über einen diskreten Aktionsraum abgebildet werden. Vor diesem Hintergrund kann die für Allokationsprobleme ausgelegte Formulierungsvariante in Abbildung 5.11 ebenfalls für die Auswahl von Operationsplänen verwendet werden.

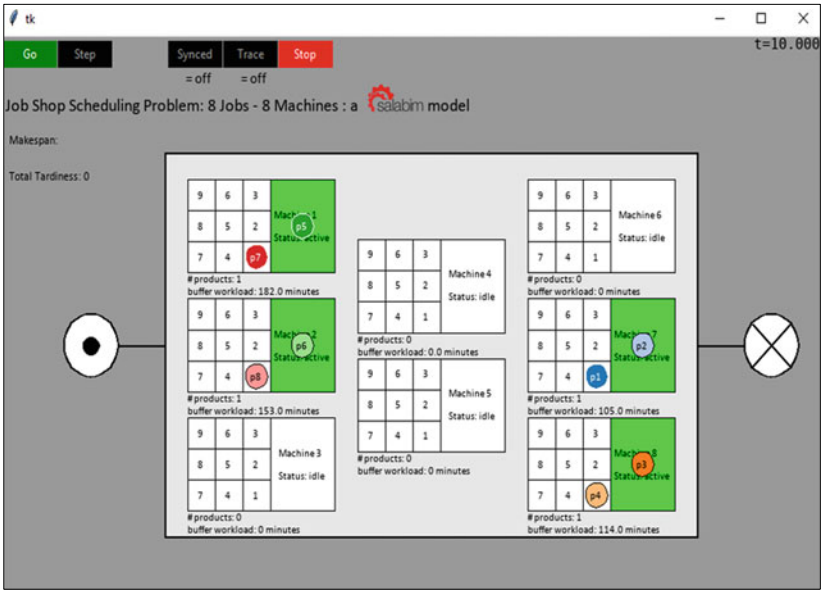


Abbildung 6.1 Animierte DES-Agentenumgebung des FJS-Problems mit acht Aufträgen und acht Produktionsressourcen (Lang et al. 2020a)

Implementierung des bestärkenden Lernverfahrens

Weil in beiden Problemen alle Entscheidungsalternativen jeweils über einen diskreten Aktionsraum abgebildet werden, eignet sich der DQN-Algorithmus zum Anlernen der Agenten. Eine ausführliche Beschreibung des DQN-Algorithmus befindet sich in Abschnitt 3.3.3 dieser Arbeit. Ergänzend zu der Beschreibung in Abschnitt 3.3.3 und der Implementierung in (Lang et al. 2020a) wurden zwei zusätzliche RL-Techniken berücksichtigt, die den Lernprozess stabilisieren:

- Anstelle des konventionellen DQN-Verfahrens wird der Double-DQN-Algorithmus (van Hasselt et al. 2016) für das Agententraining verwendet. Im Double-DQN-Algorithmus ändert sich die Berechnung der Zielwerte der Aktionsnutzen. Anstelle von Formel (3.7) findet die folgende Berechnungsvorschrift Anwendung:

$$U_t = R_{t+1} + (1 - ENDE_{t+1})\gamma + q_\pi\left(S_{t+1}, \underset{a}{\operatorname{argmax}}\, q_\pi(S_{t+1}, a; \theta); \theta^-\right) \tag{6.1}$$

Im Unterschied zum klassischen DQN-Algorithmus wird somit das diskontierte Maximum der durch den Ziel-Funktion-Approximator $q_\pi\left(s, a; \theta^-\right)$ prognostizierten Aktionsnutzen für den Folgezustand S_{t+1} auf die Belohnung R_{t+1} addiert. Stattdessen werden zunächst die Aktionsnutzen für S_{t+1} unter Verwendung des eigentlichen Aktionsnutzenfunktion-Approximator $q_\pi(s, a; \theta)$ prognostiziert. Der Index der Aktion, die den maximalen Nutzen verspricht, wird gespeichert. Erst darauffolgend werden die Aktionsnutzen für S_{t+1} unter Verwendung des Ziel-Agenten $q_\pi\left(s, a; \theta^-\right)$ prognostiziert. Anstelle des maximalen Aktionsnutzens wird derjenige Nutzen diskontiert und auf R_{t+1} addiert, der dem Index der Aktion entspricht, deren Nutzen unter $q_\pi(s, a; \theta)$ maximal ist. Die Intention hinter der aktualisierten Berechnungsvorschrift ist, die Bewertung und Auswahl von zukünftigen Aktionsnutzen Q_{t+1} zu entkoppeln und auf diese Weise das Risiko einer überoptimistischen Schätzung von Aktionsnutzen zu vermeiden.

- Der Ziel-Aktionsnutzenfunktion-Approximator $q_\pi\left(s, a; \theta^-\right)$ wird nicht nach c Zeitschritten durch den Aktionsnutzenfunktion-Approximator $q_\pi(s, a; \theta)$ überschrieben. Stattdessen wird $q_\pi\left(s, a; \theta^-\right)$ graduell durch Polyak-Mittelung aktualisiert. Das Verfahren zur Parameteraktualisierung mittels Polyak-Mittelung wird am Beispiel des DDPG-Algorithmus in Anhang D im elektronischen Zusatzmaterial erklärt.

Zustands- und Aktionsraum, versteckte Architektur sowie Belohnungsfunktion des Agenten

Tabelle 6.2 beinhaltet die wichtigsten Gestaltungsentscheidungen für die Implementierung der Agenten. Im Folgenden sollen diejenigen Gestaltungsentscheidungen näher beleuchtet werden, die nicht intuitiv verständlich sind. Die Ausgaben der versteckten Neuronenschichten des Agenten für die Ressourcenbelegungsplanung werden durch eine ELU (»Exponential Linear Unit«) -Aktivierungsfunktion verarbeitet. Hierbei handelt es sich um eine Aktivierungsfunktion, die im Rahmen des in Anhang A beigefügten Exkurses zu KNN (im elektronischen Zusatzmaterial) keine Erwähnung findet. Die ELU-Funktion ist folgendermaßen definiert.

$$\varphi(v) = \begin{cases} v, & wenn\ v > 0 \\ e^v - 1, & sonst \end{cases} \tag{6.2}$$

Ihr grafischer Verlauf ähnelt der ReLU-Funktion (vgl. Abbildung A-1 (d) in Anhang A des elektronischen Zusatzmaterials). Für positive Eingaben geben die ReLU- und ELU-Funktion dieselben Ergebnisse zurück. Für negative Eingaben gibt die ELU-Funktion, im Kontrast zur ReLU-Funktion, nicht stets den Wert null zurück, sondern eine Ausgabe ≥ -1, deren exakter Wert in Abhängigkeit von der Eingabe variiert. Die ELU-Funktion ist somit in der Lage negative Eingaben differenzierter zu verarbeiten als die ReLU-Funktion. In der vorliegenden Fallstudie ist diese Eigenschaft von Vorteil, da das Auftragsattribut »Aktuelle Unpünktlichkeit« sowohl positiv (Auftragsoperation ist verspätet) als auch negativ (Auftragsoperation ist verfrüht) ausgeprägt sein kann.

Ferner ist hervorzuheben, dass für die Auswahl von Operationsplänen ein Agentenmodell mit LSTM-Zellen innerhalb der versteckten Schichten verwendet wird. Der Aufbau und die Funktionsweise einer LSTM-Zelle wird in Anhang A im elektronischen Zusatzmaterial erklärt. Die Verwendung von LSTM-Zellen wird zum einen dadurch motiviert, dass für die Auswahl von Operationsplänen kaum geeignete Informationen für die Modellierung von Zuständen vorhanden sind, weil alle Aufträge zum Zeitpunkt null verfügbar sind. Damit einhergehend erfolgt ebenfalls die Auswahl aller Operationspläne zum Zeitpunkt null, zu welchem das Produktionssystem noch keine Aufträge enthält. Dementsprechend können keine Informationen des Produktionssystems bei der Zustandsbildung berücksichtigt werden. Zum anderen liegt die Annahme zugrunde, dass die Auswahl eines Operationsplans für einen Auftrag die Auswahl eines Operationsplans eines anderen Auftrags beeinflussen sollte, da die kombinierte Anwendung der ausgewählten Operationspläne maßgeblich die Ressourcenbelegung des Produktionssystems bestimmt. Da LSTM-Zellen rekurrente synaptische Verbindungen zu sich selbst pflegen, können sie sequenzielle Zusammenhänge zwischen aufeinanderfolgenden Entscheidungen für die Selektion von Operationsplänen berücksichtigen. Die Anzahl der versteckten Perzeptron-Neuronen (Agent für die Ressourcenbelegungsplanung) bzw. LSTM-Zellen (Agent für die Auswahl von Operationsplänen) pro Schicht wurde in grober Orientierung an die $(2n + 1)$-Regel ermittelt (Hecht-Nielsen 1987), wobei n der Anzahl von Eingabeneuronen entspricht. Eine weitere Herausforderung bei der Auswahl von Operationsplänen ist die Gestaltung der Belohnungsfunktion. Die in Abschnitt 5.3.5 angeführten Beispiele für Belohnungsfunktionen sind ungeeignet, weil die Auswahl von Operationsplänen weder einen unmittelbar quantifizierbaren Einfluss auf die Fertigstellungszeit von Aufträgen, geschweige denn auf die Arbeitslast der Produktionsressourcen ausübt. Als Kompromisslösung wird eine verzögerte Belohnungsstrategie implementiert, d. h. Belohnungen werden nicht unmittelbar nach der Auswahl eines Operationsplans vergeben, sondern erst nachdem alle Operationen des gewählten Plans bearbeitet wurden und der Auftrag das System verlässt.

Tabelle 6.2 Gestaltung der Agenten für die Ressourcenbelegungsplanung und für die Auswahl von Operationsplänen (in Anlehnung an Lang et al. 2020a)

	Agent für die Ressourcenbelegungsplanung	**Agent für die Auswahl von Operationsplänen**
Zustand (Eingabe-schicht)	Auftragsattribute (2 Neuronen) · Aktuelle Unpünktlichkeit ($t_{sim} - d_{j,v,o}$) (1 Neuron) · Fertigstellungsgrad (Anzahl abgeschlossener Operationen im Verhältnis zu der Anzahl noch abzuschließender Operationen) (1 Neuron)	Auftragsattribute (8×4 Neuronen) · Vier Operationspläne, wobei jeder Operationsplan durch vier Operations-indizes definiert ist (Operationspläne mit weniger als vier Operationen werden mit Nullwerten aufgefüllt) (4×4 Neuronen) · Durchschnittliche Bearbeitungszeit über alle Produktionsressourcen für jede Operation eines jeden Operationsplans (4×4 Neuronen)
	Systemattribute (10 oder 16 Neuronen) Für jede Produktionsressource: · Bearbeitungszeit der nächsten Operation des zu allozierenden Auftrags · (5 oder 8 Neuronen) · Arbeitslast (5 bis 8 Neuronen)	
Agent (Versteckte Schichten)	Zwei Perzeptron-Schichten (MLP) mit jeweils 32 Neuronen und ELU-Aktivierungsfunktion	Zwei LSTM-Schichten mit jeweils 64 LSTM-Zellen
Aktion (Ausgabeschicht)	Allokation zu Produktionsressource i (5 oder 8 Neuronen)	Auswahl von Operationsplan $OP_{j,v}$ (4 Neuronen)
Belohnung	Abgewandelte Version von $\mathcal{R}_T(j)$ (siehe Formel (5.6)). Es wird zunächst die Unpünktlichkeit $L_{j,v,o}$ ermittelt. Die Belohnung berechnet sich dann wie folgt: $+10$, sofern der Auftrag zwischen $0,8 * d_{j,v,o}$ und $1,0 * d_{j,v,o}$ fertiggestellt wird. ± 0, sofern der Auftrag früher als $0,8 * d_{j,v,o}$ fertiggestellt wird. $(-1) * L_{j,v,o}$, sofern der Auftrag später als $d_{j,v,o}$ fertiggestellt wird.	Abgewandelte Version von $\mathcal{R}_T(j)$ (siehe Formel (5.6)). Es wird zunächst die Unpünktlichkeit L_j ermittelt. Die Belohnung berechnet sich dann wie folgt: $+10$, sofern der Auftrag zwischen $0,8 * d_j$ und $1,0 * d_j$ fertiggestellt wird. ± 0, sofern der Auftrag früher als $0,8 * d_j$ fertiggestellt wird. $(-1) * L_j$, sofern der Auftrag später als d_j fertiggestellt wird.

Für beide Agenten wird eine abgewandelte Version von $\mathcal{R}_T(j)$ (siehe Formel (5.6)) als Belohnungsfunktion implementiert. Der Agent der Ressourcenbelegungsplanung erhält Belohnungen auf Basis der anteilsmäßigen Fertigstellungsfrist $d_{j,v,o}$ und der anteilsmäßigen Unpünktlichkeit $L_{j,v,o}$ von Operation o des gewählten Operationsplans v von Auftrag j. Hingegen erhält der Agent für die Auswahl von Operationsplänen Belohnungen auf Basis der auftragsbezogenen Fertigstellungsfrist d_j und der auftragsbezogenen Unpünktlichkeit L_j. Beide Agenten erhalten eine positive Belohnung, sofern die betrachtete Operation (Ressourcenbelegungsplanung) bzw. der gesamte Auftrag (Operationsplanung) pünktlich zum Fertigstellungszeitpunkt oder bis maximal 20 Prozent vor Eintritt der Fertigstellungsfrist abgeschlossen ist. Auf diese Weise soll der Agent angehalten werden, Aufträge mit engen Fertigstellungsfristen bevorzugt gegenüber Aufträgen mit toleranten Fertigstellungsfristen einzulasten.

6.1.3 Diskussion der Ergebnisse

Die Experimente dienen insbesondere der Beantwortung der folgenden Fragen: *(i)* Welche Lösungsgüte leisten die durch den DQN-Algorithmus trainierten Agenten im Vergleich zur Lösungsstrategie von Rajkumar et al. (2010)? *(ii)* Sind die DQN-trainierten Agenten für das Problem mit fünf Produktionsressourcen in der Lage, die erlernte Entscheidungspolitik auf Probleminstanzen mit fünf oder zehn Aufträgen zu interpolieren, sofern die Agenten anhand von zwanzig Aufträgen angelernt wurden? *(iii)* Sind die DQN-trainierten Agenten für das Problem mit fünf Produktionsressourcen in der Lage, die erlernte Entscheidungspolitik auf Probleminstanzen mit zehn oder zwanzig Aufträgen zu extrapolieren, sofern die Agenten anhand von fünf Aufträgen angelernt wurden?

Zur Beantwortung der Fragen werden in vier Experimenten jeweils zwei Agenten auf je einer der in Tabelle 6.1 (Abschnitt 6.1.1) gelisteten Probleminstanzen angelernt. In allen Experimenten werden die Agenten nacheinander trainiert. Der Agent für die Ressourcenbelegungsplanung wird in allen Experimenten zuerst trainiert.

Während dessen Training erfolgt die Auswahl von Operationsplänen randomisiert. Für das Training des Agenten zur Selektion von Operationsplänen wird wiederum stets der zuvor angelernte Agent für die Ressourcenbelegungsplanung miteingebunden. Tabelle 6.3 beinhaltet die Hyperparameter, mit welchen beide Agenten trainiert wurden. Tabelle 6.4 zeigt die Lösungsgüte der Agenten auf den unterschiedlichen Probleminstanzen, gemessen an der Gesamtdauer des resultierenden Ablaufplans C_{max} und der Gesamtverspätung über alle Aufträge T. Zum Vergleich sind ebenfalls die Ergebnisse der GRASP (Greedy Randomized Adaptive Search Procedure) -Metaheuristik gelistet, die von Rajkumar et al. (2010) für dieselben Probleminstanzen untersucht wurde.

Tabelle 6.3 Hyperparameter-Einstellungen für den DQN-Algorithmus

Hyperparameter	Wert
Anzahl Episoden	5.000
Lernrate α	0,0001
Diskontierungsrate γ	0,99
Wahrscheinlichkeit für zufällige Aktion ε	1
Diskontierungsrate für ε	0,999
Minimum von ε	0,001
Kapazität des Erfahrungsspeichers	400
Trainingslosgröße	40
Polyak-Faktor τ	0,01
Optimierungsverfahren	Adam

Die Ergebnisse zeigen, dass DQN für die jeweilige Probleminstanz, die während des Agententrainings beobachtet wurde, stets eine bessere Lösung erzielen kann als die GRASP-Metaheuristik. Darüber hinaus generiert DQN auch auf anderen Probleminstanzen gute Ergebnisse, die nicht während des Trainings beobachtet wurden. Allerdings ist kein Agent, der auf einer Probleminstanz mit fünf Produktionsressourcen angelernt wurde, in der Lage, auf allen 5M-Probleminstanzen bessere Ergebnisse zu erzielen als die GRASP-Metaheuristik. Diesbezüglich schneidet derjenige Agent am besten ab, welcher auf dem 20J×5M-Problem angelernt wurde. Dieser Agent leistet ebenfalls auf dem 10J×5M-Problem bessere Ergebnisse als die GRASP-Metaheuristik. Hervorzuheben ist, dass der auf dem 20J×5M-Problem trainierte Agent ebenso bessere Ergebnisse auf dem 10J×5M-Problem leistet als derjenige Agent, der direkt auf dem 10J×5M-Problem trainiert wurde. Dies könnte als Indiz gedeutet werden, dass der Agent besser auf unbekannte Zustände interpolieren als extrapolieren kann. Dagegen spricht, dass der auf dem 20J×5M-Problem trainierte Agent ebenfalls auf dem 5J×5M-Problem bessere Ergebnisse erzielt als der auf dem 10J×5M-Problem trainierte Agent. Dies ist bemerkenswert, da der auf dem 10J×5M-Problem trainierte Agent ebenfalls nur auf das 5J×5M-Problem interpolieren muss. In Hinblick auf die Fertigstellungsfristen und die Anzahl von Aufträgen weisen zudem das 5J×5M-Problem und das 10J×5M-Problem eine höhere Ähnlichkeit zueinander auf als das 5J×5M-Problem zum 20J×5M-Problem. Vor diesem Hintergrund ist es ebenfalls möglich, dass das Training des DQN-Agenten

Tabelle 6.4 Lösungsgüte des auf verschiedenen FJS-Probleminstanzen trainierten DQN-Algorithmus im Vergleich zum GRASP-Algorithmus von Rajkumar et al. (2010)

Problem Verfahren	**5 J × 5M**		**8 J × 8M**		**10 J × 5M**		**20 J × 5M**	
	C_{max}	T	C_{max}	T	C_{max}	T	C_{max}	T
GRASP-Metaheuristik	242	365	253	908	421	2.025	924	2.919
	Σ 607		Σ 1.161		Σ 2.446		Σ 3.843	
DQN (trainiert auf 5J × 5M)	189	305	–	–	538	2.390	1.073	3.572
	Σ 494		Σ –		Σ 2.928		Σ 4.645	
DQN (trainiert auf 8J × 8M)	–	–	184	494	–	–	–	–
	Σ –		Σ 678		Σ –		Σ –	
DQN (trainiert auf 10J × 5M)	279	432	–	–	423	1.866	1.015	3.095
	Σ 711		Σ –		Σ 2.289		Σ 4.100	
DQN (trainiert auf 20J × 5M)	244	458	–	–	371	1.864	699	427
	Σ 702		Σ –		Σ 2.235		Σ 1.126	

auf dem 10J×5M-Problem unter ungünstigen Hyperparametern verlief. Eine eindeutige Aussage, ob der Agent das erlernte Wissen besser interpolieren oder extrapolieren kann, ist auf Basis der Ergebnisse in Tabelle 6.4 nicht möglich.

In allen Experimenten wurden beide Agenten jeweils auf einem Kern eines Prozessors des Modells »AMD EPYC 7702P 64-Core« trainiert, wobei jeder Prozessorkern eine Taktfrequenz von maximal 2 GHz aufweist. Das Training des Agenten für die Ressourcenbelegungsplanung und des Agenten für die Auswahl von Operationsplänen benötigt für das 5J×5M-, 8J×8M-, 10J×5M-, und 20J×5M-Problem jeweils in etwa 17, 23, 25 bzw. 72 Minuten. Wie jedoch die Lernkurven am Beispiel der Probleminstanz 5J×5M in Abbildung 6.2 zeigen, werden die besten Ergebnisse mitunter wesentlich früher erzielt, nämlich nach ca. 1500 Episoden (Ressourcenbelegungsplanung) bzw. in den ersten 10 bis 100 Episoden (Auswahl von Operationsplänen). Der tatsächliche Trainingsaufwand ist somit geringer als die Rechenzeit vermuten lässt. Sobald das Training der Agenten beendet ist, benötigt die Generierung einer Lösung lediglich einen Simulationslauf, welcher weniger als eine Sekunde Rechenzeit beansprucht. In Abbildung 6.2 sticht hervor, dass die Lernkurven des Agenten für die Auswahl von Operationsplänen mit fortschreitenden Episoden kaum mehr Oszillationen unterliegen. Hingegen kann eine solche Entwicklung beim Training des Agenten für die Ressourcenbelegungsplanung nicht beobachtet werden. Dieser Umstand ist dadurch zu erklären, dass beim Training des Agenten für die Ressourcenbelegungsplanung die Operationspläne der Aufträge stets zufällig gewählt werden. Demgegenüber wird während des Agententrainings für die Auswahl von Operationsplänen der zuvor angelernte Agent für die Ressourcenbelegungsplanung stets miteingebunden. Sobald der Agent für die Auswahl von Operationsplänen zu einer fixen Entscheidungspolitik konvergiert und die Wahrscheinlichkeit für eine zufällige Aktion ε ihren Minimalwert erreicht, werden nur noch sehr selten Operationspläne zufällig ausgewählt. Hierdurch sinkt die Wahrscheinlichkeit für die Identifikation von besseren Lösungen und für eine erneute Anpassung der Entscheidungspolitik rapide ab. Die Trainingsmetriken für das 8J×8M-, 10J×5M-, und 20J×5M-Problem sind als Anhang H dem elektronischen Zusatzmaterial beigefügt.

6.1.4 Erweiterung des Problems um einen dynamischen Auftragshorizont

Am Beispiel des auf dem 20J×5M-Problem trainierten Agenten soll im Folgenden untersucht werden, ob die erlernte Entscheidungspolitik ebenfalls unter Berücksichtigung eines dynamischen Auftragshorizonts qualitativ hochwertige Ergebnisse liefert. Der auf dem 20J×5M-Problem trainierte Agent wird für

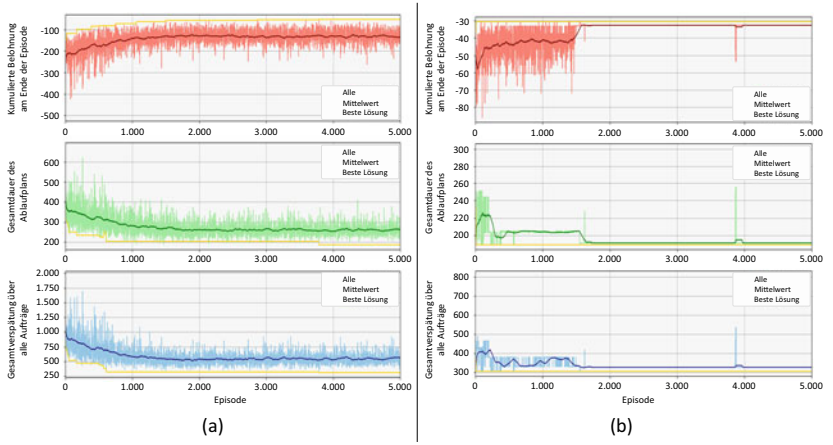


Abbildung 6.2 Trainingsmetriken der Probleminstanz 5J×5M (a) für den Agenten der Ressourcenbelegungsplanung und (b) für den Agenten für die Auswahl von Operationsplänen

die folgenden Experimente ausgewählt, weil die Ergebnisse in Abschnitt 6.1.3 darauf hinweisen, dass dessen Entscheidungspolitik auf die anderen Probleminstanzen mit fünf Produktionsressourcen am besten skaliert. Die in Abschnitt 6.1.1 skizzierte Problemstellung wird zu diesem Zweck wie folgt abgewandelt:

- Ausgangspunkt ist das 20J×5M-Problem, demgemäß initial 20 Aufträge mit Fertigstellungsfristen zwischen 400 und 700 Zeiteinheiten freigegeben sind.
- Zu zufälligen Zeitpunktpunkten gelangen neue Aufträge in das System. Zu jedem Zeitpunkt gelangt genau ein neuer Auftrag in das System.
- Die zufälligen Zwischenankunftszeiten von Aufträgen sind normalverteilt mit einem Erwartungswert von zehn Zeiteinheiten und einer Standardabweichung von fünf Zeiteinheiten. Der minimale zeitliche Abstand zwischen der Ankunft von zwei Aufträgen beträgt null Zeiteinheiten.
- Für die Bestimmung von möglichen Operationsplänen und -zeiten von neu eintreffenden Aufträgen wird jeweils ein Auftrag aus der Menge der zwanzig initial freigegebenen Aufträgen zufällig gleichverteilt ausgewählt. Der neu eintreffende Auftrag übernimmt sodann die Operationspläne und -zeiten des zufällig gewählten Auftrags.
- Die Fertigstellungsfrist eines neu eintreffenden Auftrags wird aus dem Intervall $[80, 600]$ zufällig gleichverteilt ausgewählt und ist stets ganzzahlig.

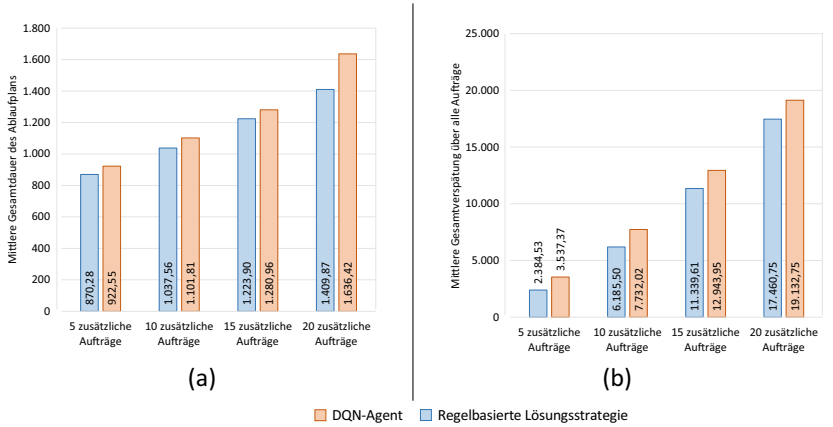


Abbildung 6.3 Lösungsgüte des DQN-Agenten auf der Probleminstanz 20J×5M mit dynamischen Auftragshorizont gemessen an (a) der mittleren Gesamtdauer des Ablaufplans und (b) der mittleren Gesamtverspätung über alle Aufträge

Es werden vier Szenarien untersucht, in welchen *(i)* fünf, *(ii)* zehn, *(iii)* fünfzehn und *(iv)* zwanzig Aufträge während der Simulation zusätzlich eingelastet werden. Aufgrund der stochastischen Attribute und Zwischenankunftszeiten von neu eintreffenden Aufträgen werden die Ergebnisse je Experiment über 100 Simulationsläufe gemittelt. Zum Vergleich wird zusätzlich jedes Experiment unter Anwendung von zwei statischen Regeln für die Auswahl von Operationsplänen und für die Allokation von Aufträgen durchgeführt. Die Regel für die Auswahl von Operationsplänen wählt denjenigen Operationsplan, bei welchem die mittlere Bearbeitungszeit für alle Operationen minimal ist. Existieren mehrere Operationspläne mit minimalen Bearbeitungszeiten, wird der erste Operationsplan aus der Teilmenge der Operationspläne mit minimalen Bearbeitungszeiten gewählt. Die Regel für die Allokation von Aufträgen weist einen Auftrag stets derjenigen Produktionsressource zu, welche zum Entscheidungszeitpunkt die minimale Arbeitslast aufweist. Abbildung 6.3 stellt die Gesamtdauer des Ablaufplans und die Gesamtverspätung über alle Aufträge für alle vier Szenarien im Vergleich zur regelbasierten Lösungsstrategie grafisch dar. Die Diagramme zeigen, dass der DQN-Agent im Mittel stets bessere Ergebnisse erzielt als die regelbasierte Lösungsstrategie. Es ist anzunehmen, dass sich die Ergebnisqualität des DQN-Agenten nochmals steigert, wenn dieser direkt auf einem dynamischen

Auftragshorizont angelernt wird. Das vorliegende Experiment soll jedoch an dieser Stelle hinreichend sein, um nachzuweisen, dass der angelernte DQN-Agent ebenfalls auf Probleminstanzen skalieren kann, die einen dynamischen Auftragshorizont aufweisen und in denen Aufträge mitunter zufällig attribuiert werden. Ergänzend zur Diagrammdarstellung in Abbildung 6.3 werden die Ergebnisse in Anhang H im elektronischen Zusatzmaterial tabellarisch zusammengefasst.

6.2 Dynamisches Parallel-Maschinen-Problem mit familienabhängigen Rüstzeiten und ressourcenabhängigen Bearbeitungsgeschwindigkeiten

Diese Fallstudie wurde konzipiert, um einen Funktionsnachweis für die erste Variante zur Formulierung von Sequenzierungsproblemen als ML-Aufgabe zu liefern. In dieser Variante erfolgt die Auswahl des nächsten Auftrags, indem zuvor alle verfügbaren Aufträge in einer Warteschlange priorisiert werden. Die entsprechende Formulierungsvariante wurde erstmals in Lang et al. (2021a) theoretisch vorgestellt. In der folgenden Fallstudie wird mithilfe des PPO-Algorithmus ein Agent angelernt, der die Steuerung der Produktionsabläufe als eine kombinierte Ressourcenbelegungs-, Reihenfolge- und Losgrößenplanung umsetzt.

6.2.1 Problembeschreibung

In einem Parallel-Maschinen-Problem (PMP) müssen eine Menge von Aufträgen J von einer Menge von Produktionsressourcen M bearbeitet werden. Die Produktionsressourcen sind parallel aufgestellt, d. h. es existiert genau eine Produktionsstufe und alle Produktionsressourcen gehören derselben Produktionsstufe an. Ferner wird jeder Auftrag von genau einer Produktionsressource aus M bearbeitet, bevor er das System verlässt. Das vorliegende PMP ist dynamischer Natur, d. h. nicht alle Aufträge stehen zum Zeitpunkt null zur Verfügung, sondern kommen losweise zu vordefinierten Zeitpunkten in das System. Ein Los b besteht aus n_b Aufträgen und besitzt eine individuelle Freigabezeit r_b, zu der alle Aufträge aus b in das System eintreten. Die Anzahl der Aufträge, die Anzahl der Produktionsressourcen und weitere Parameter sind abhängig von der jeweiligen Probleminstanz. Tabelle 6.5 enthält alle Parameter, die eine Probleminstanz charakterisieren, sowie mögliche Wertausprägungen.

Im Folgenden werden die Parameter und damit verbundene Restriktionen des PMP kurz dargestellt. Jeder Auftrag j ist durch genau eine Operationszeit o_j und eine Fertigstellungsfrist d_j charakterisiert. Im vorliegenden PMP sind alle Operationszeiten zufällig im Intervall $[5, 15]$ gleichverteilt (uniform). Alle Operationszeiten werden zu Beginn einer Episode ausgewürfelt. Fertigstellungsfristen werden ebenfalls zufällig nach dem Verfahren von Potts und van Wassenhove (1985) bestimmt. Konkret wird jede Fertigstellungsfrist aus der folgenden Gleichverteilung gezogen.

$$d_j = \text{Uniform}\left[\hat{W}_i * \left(1 - w_d - \frac{D}{2}\right), \hat{W}_i * \left(1 - w_d + \frac{D}{2}\right)\right] \tag{6.3}$$

Der Parameter $\hat{W}_i$ repräsentiert eine Schätzung der Arbeitslast, die von jeder Produktionsressource i im Mittel bewältigt werden muss. Für das vorliegende PMP wird $\hat{W}_i$ wie folgt geschätzt.

$$\hat{W}_i = \frac{1}{|M|}\left(\sum_{j=1}^{|J|} o_j + \left(\frac{|J| + |Fam|}{2}\right) * s\right) \tag{6.4}$$

Bei dem zweiten Term in Formel (6.4) handelt es sich um eine Schätzung der mittleren Gesamtrüstzeit. Die mittlere Gesamtanzahl der Rüstungen wird als Durchschnitt der Anzahl von Aufträgen und der Anzahl von Auftragsfamilien angenommen. Die Entscheidung, ob eine Produktionsressource gerüstet werden muss oder nicht, hängt von der Familie des ausgewählten Auftrags und der aktuellen Rüstung der Produktionsressource ab. Jeder Auftrag gehört genau einer Auftragsfamilie an und jede Produktionsressource ist stets genau für eine Auftragsfamilie gerüstet. Hierbei beschreibt Fam die Menge von Familien, f_j die Familie von Auftrag j und f_i die Rüstung von Produktionsressource i. Jeder Familie f gehören ein oder mehrere Aufträge an. Ferner können keine oder eine beliebige Anzahl von Produktionsressourcen mit derselben Familie gerüstet sein. Sofern ein Auftrag auf einer Produktionsressource eingelastet wird, dessen Familie nicht der Rüstung der Ressource entspricht ($f_i \neq f_j$), muss eine Rüstzeit s veranschlagt werden. Nach Ablauf der Rüstzeit besitzt die Produktionsressource die richtige Rüstung, um den eingelasteten Auftrag zu bearbeiten ($f_i = f_j$). Sofern f_i und f_j von Beginn an identisch sind, wird keine zusätzliche Rüstzeit veranschlagt.

Die Produktionsressourcen unterliegen verschiedenen Bearbeitungsgeschwindigkeiten. Bei dem ersten Typ von Produktionsressourcen handelt es sich um

Tabelle 6.5 Parameter des Parallel-Maschinen-Problems

Problemparameter	Mögliche Werte
Anzahl Produktionsressource $\|M\|$	{6, 7, 8}
Anzahl von initialen Aufträgen im System $\|b = 0\|$	{35, 40, 45, 50, 55, 60}
Anzahl von neuen Auftragslosen $(\|B\| - 1)$	3
Anzahl von Aufträgen in neu ankommenden Losen $n_{1,\ldots,\|B\|}$	5
Freigabezeiten aller Auftragslose $r_{0,\ldots,\|B\|}$	{0, 20, 40, 60}
Operationszeiten o_j	Uniform [5, 15]
Fertigstellungsfristen d_j	siehe Formeln (6.1) und (6.2)
Durchschnittlicher Verspätungsfaktor w_d	{0,1, 0,2}
Relativer Wertebereich der Fertigstellungsfristen D	{0,15, 0,25}
Anzahl Auftragsfamilien $\|Fam\|$	{5, 7}
Rüstzeit s	10
Bearbeitungsgeschwindigkeitsfaktor v_i von Produktionsressource i	{0,8, 1,0}

neue Maschinen, die einen Auftrag in kürzerer als der veranschlagten Operationszeit bearbeiten können ($v_i = 0, 8$). Der zweite Typ von Produktionsressourcen umfasst ältere Maschinen, welche exakt die veranschlagte Operationszeit für die Bearbeitung eines Auftrags benötigen ($v_i = 1, 0$).

Die Zielstellung ist die Minimierung der Gesamtverspätung über alle Aufträge T.

6.2.2 Anwendung des PPO-Algorithmus zur Lösung des Problems

Der folgende Abschnitt beschreibt den Einsatz des PPO-Algorithmus, um eine agentenbasierte Produktionsablaufsteuerung für das vorliegende PMP zu trainieren.

Entwurf der Agentenumgebung und Definition der maschinellen Lernaufgabe
Das vorgestellte PMP kann gemäß Abbildung 5.3 (Abschnitt 5.2) entweder als Produktionsbereich mit lokalen Warteschlangen für jede Produktionsressource

oder als Produktionsbereich mit zentraler Warteschlange für alle Produktionsressourcen modelliert werden. Wie bereits zu Beginn von Abschnitt 6.1.4 dargelegt, dient die vorliegende Fallstudie der Evaluation der ersten Formulierungsvariante von Sequenzierungsproblemen als ML-Aufgabe. Das volle Potenzial dieser Variante wird ausgeschöpft, wenn das PMP als Produktionsbereich mit einer zentralen Warteschlange für alle Produktionsressourcen modelliert wird (Abbildung 5.3 (b)). Auf diese Weise können die Ressourcenbelegungs- und Reihenfolgeplanung sowie die Losbildung kombiniert durch lediglich einen Agenten gesteuert werden. Hierbei wird ausschließlich die Reihenfolgeplanung explizit gesteuert, während die Ressourcenbelegungsplanung und Losbildung nur implizit berücksichtigt werden. Jedes Mal, wenn eine Produktionsressource verfügbar wird, ist der Agent angehalten, einen nachfolgenden Auftrag mittels Priorisierung auszuwählen (explizite Reihenfolgeplanung). Der ausgewählte Auftrag wird sofort auf der verfügbaren Produktionsressource eingelastet (implizite Ressourcenbelegungsplanung). Sofern der ausgewählte Auftrag einer anderen Familie angehört als der zuvor auf derselben Produktionsressource bearbeitete Auftrag, muss zunächst die Produktionsressource umgerüstet werden (impliziter Beginn eines neuen Produktionsloses). Andernfalls kann der Auftrag mit der aktuellen Ressourcenrüstung sofort bearbeitet werden (implizite Fortführung des aktuellen Produktionsloses).

Begründung der Auswahl des bestärkenden Lernverfahrens

Für das Training des Agenten wird der PPO-Algorithmus verwendet. Es handelt sich um ein stochastisches EPA-Verfahren, das ferner als AC-Verfahren klassifiziert ist. Der PPO-Algorithmus wird für das Training von Agenten zur Priorisierung von Aufträgen als besonders geeignet eingeschätzt, aufgrund des folgenden Sachverhalts: Vor der Auswahl eines spezifischen Auftrags aus n Aufträgen müssen zunächst n Prioritäten berechnet werden. Hierbei können bereits kleinste Veränderungen der Parameter der Entscheidungspolitik zu signifikanten Veränderungen der ausgegebenen Prioritäten führen. Infolgedessen unterscheiden sich die resultierenden Auftragssequenzen zwischen t und $t + 1$ zum Teil stark voneinander. Über einen längeren Zeitraum betrachtet neigt der Agent auf diese Weise deutlich zu einer Exploration des Lösungsraums und vernachlässigt die Tiefenuntersuchung von vielversprechenden Regionen. Dies kann dazu führen, dass der Agent zu keiner Entscheidungspolitik konvergiert. Der PPO-Algorithmus adressiert genau diesen Aspekt, nämlich dass bereits kleinste Veränderungen in den Agentenparametern große Veränderungen in der Entscheidungspolitik hervorrufen können, durch zwei Techniken. Erstens werden die Parameter des Actor-Modells, welches die stochastische Entscheidungspolitik parametrisiert,

stets im Verhältnis zu den Parametern der vergangenen Entscheidungspolitik aktualisiert. Zweitens wird die maximal tolerierte Abweichung von der vergangenen Entscheidungspolitik zusätzlich durch ein Clipping-Verfahren beschränkt. Eine ausführliche Beschreibung des PPO-Algorithmus ist als Anhang C dem elektronischen Zusatzmaterial beigefügt.

Zustandsraum des Agenten

Für die Priorisierung (Aktion) eines Auftrags werden die in Tabelle 6.6 gelisteten Attribute zu einem Zustandsvektor gebildet und vorwärts durch das Agentenmodell propagiert. Es müssen zunächst alle Aufträge in der Warteschlange priorisiert werden, bevor ein bestimmter Auftrag ausgewählt werden kann. Anstatt einzelne Zustandsvektoren vorwärts durch den Agenten zu propagieren, um Aufträge einzeln und nacheinander zu priorisieren, werden über eine Zustandsmatrix alle in der Warteschlange befindlichen Aufträge durch eine einzige Vorwärtspropagierung priorisiert. Hierbei repräsentiert jede Zeile den Zustand eines Auftrags und jede Spalte ein Attribut des jeweiligen Zustands. Abbildung 6.4 skizziert den Aufbau der Zustandsmatrix.

Tabelle 6.6 Attribute zur Bildung von Zuständen im Parallel-Maschinen-Problem

Auftragsattribute	**Produktionsressourcenattribute**
Operationszeit o_j	Verstrichene Simulationszeit t_{sim}
Fertigstellungsfrist d_j	Rüstung f_i
Familie f_j	Bearbeitungsgeschwindigkeit v_i

Obgleich jeder Zustand aus den sechs Attributen in Tabelle 6.6 gebildet wird, besitzt die Zustandsmatrix 19 Spalten, d. h. jeder Zustandsvektor in der Matrix besteht aus 19 Elementen. Der Grund ist, dass die Auftragsfamilien, die Ressourcenrüstung und die Bearbeitungsgeschwindigkeit der Ressourcen binär 1-aus-n-kodiert sind. Die Binärvektoren von Auftragsfamilien und Ressourcenrüstungen sind jeweils siebenstellig, da maximal sieben Familien in jeder Probleminstanz existieren. Die Bearbeitungsgeschwindigkeit von Produktionsressourcen wird über einen zweistelligen Binärvektor abgebildet, da in allen Probleminstanzen lediglich zwei Bearbeitungsgeschwindigkeiten unterschieden werden. Die verschiedenen Farben in der Zustandsmatrix kennzeichnen unterschiedliche Werte der jeweiligen Attribute zu ein und demselben Zeitpunkt. Die Auftragsattribute (1–9) sind jeweils auftragsspezifisch ausgeprägt, jedoch stets identisch zu verschiedenen Entscheidungszeitpunkten. Hingegen sind die

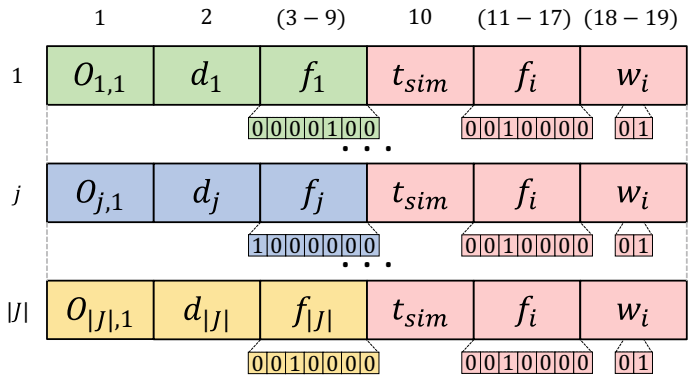


Abbildung 6.4 Zustandsmatrix zur Priorisierung von Aufträgen im Parallel-Maschinen-Problem

Attribute der verfügbaren Produktionsressource (10–19) zu ein und demselben Entscheidungszeitpunkt für alle auswählbaren Aufträge identisch, variieren jedoch zu unterschiedlichen Entscheidungszeitpunkten. Zu jedem Entscheidungszeitpunkt werden die gleiche Art und Anzahl von Attributen eingelesen. Die Anzahl der Spalten der Zustandsmatrix ist somit konstant. Jedoch reduziert sich die Anzahl der Matrixzeilen mit jedem ausgewählten Auftrag, weil ausgewählte Aufträge nach ihrer Bearbeitung das System verlassen und nicht nochmal ausgewählt werden können.

Architektur und versteckte Schichten des Agenten

Die Verwendung des PPO-Algorithmus erfordert, dass der Agent ein Actor- und ein Critic-Modell umfasst. Abbildung 6.5 visualisiert den Aufbau des Agenten.

Das Actor- und Critic-Modell teilen sich dieselbe Eingabeschicht, besitzen jedoch jeweils eine eigenständige versteckte Architektur und Ausgabeschicht. Die versteckte Architektur beider Modelle beinhaltet jeweils eine Schicht mit 32 bidirektionalen GRU-Zellen und daran anschließend eine Schicht mit 64 Perzeptron-Neuronen mit ReLU- Aktivierungsfunktion. Analog zur LSTM-Zelle gehört die GRU-Zelle zu den rekurrenten Neuronenmodellen und kann sequenzielle Zusammenhänge in Eingangsdatenströmen detektieren. Eine detaillierte Beschreibung der GRU-Zelle befindet sich in Anhang A im elektronischen

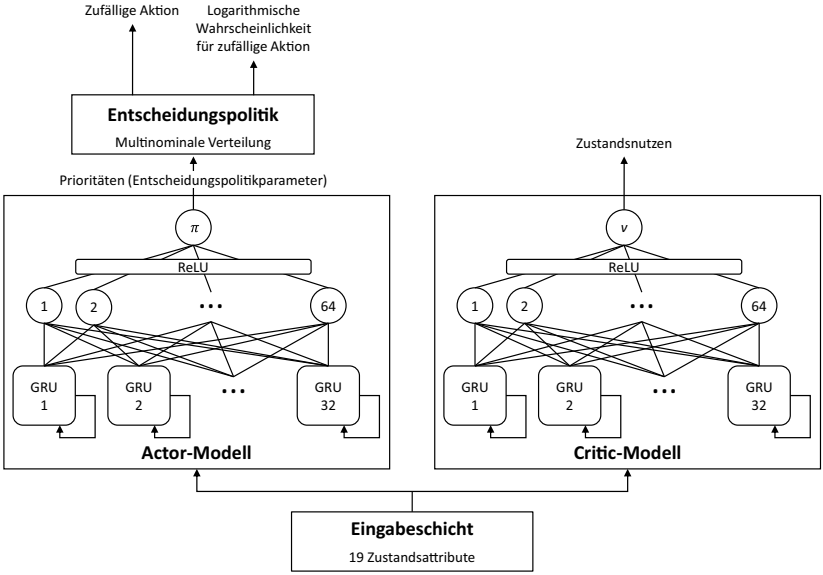


Abbildung 6.5 Architektur des Actor-Critic-Agenten für die Priorisierung von Aufträgen

Zusatzmaterial (Abbildung A-5). Durch Verwendung von bidirektionalen GRU-Zellen wird sichergestellt, dass der Agent für die Priorisierung eines Auftrags die Zustandsinformationen sowohl von vorgelagerten als auch von nachgelagerten Aufträgen berücksichtigt. Bidirektional bedeutet, dass die GRU-Zellen Zustandssequenzen beidseitig, von vorne nach hinten und von hinten nach vorn, einlesen. Folglich enthält der Ausgabevektor der GRU-Schicht doppelt so viele Werte wie die Anzahl der enthaltenen GRU-Zellen. Aus diesem Grund muss die daran anschließende Perzeptronen-Schicht doppelt so viele Neuronen beinhalten wie die vorgelagerte GRU-Schicht.

Aktionsraum des Agenten

Der Aktionsraum des Agenten ist kontinuierlich, d. h. sowohl das Actor- als auch das Critic-Modell besitzen lediglich ein Ausgabeneuron. Das Actor-Modell gibt die Priorität eines Auftrags aus, während das Critic-Modell den Nutzen für den eingegebene Zustand prognostiziert. Da der PPO-Algorithmus zu den stochastischen EPA-Verfahren zählt, wird der nächste zu produzierende Auftrag nicht unmittelbar auf Basis der maximalen Auftragspriorität gewählt. Stattdessen

werden die Prioritäten für die Parametrisierung einer multinominalen Verteilung verwendet, die als stochastische Entscheidungspolitik dient. Hierbei stehen die Auswahlwahrscheinlichkeiten der Aufträge im gleichen Verhältnis zueinander wie die eingehenden Auftragsprioritäten. Aufgrund der losweisen Verarbeitung von Zuständen, handelt es sich bei den Ausgaben des Actor- und Critic-Modells um Vektoren, deren Länge der Anzahl von Zeilen der eingegebenen Zustandsmatrix entsprechen. Darauffolgend wird mithilfe der multinominalen Verteilung eine zufällige Aktion gewählt. Zusätzlich wird ebenfalls die logarithmische Auswahlwahrscheinlichkeit der gewählten Aktion ausgegeben. Im PPO-Algorithmus werden die Auswahlwahrscheinlichkeiten von gewählten Aktionen benötigt, um das Lernsignal des Actor-Modells zu berechnen (für eine detaillierte Erklärung siehe Anhang C im elektronischen Zusatzmaterial).

Belohnungsfunktion und Training des Agenten

Abschließend soll die Gestaltung der Belohnungsfunktion dargelegt werden, mit der die Aktionen des Agenten bewertet werden. In der vorliegenden Fallstudie hat ein zweistufiges Belohnungssystem die besten Lernfolge erzielt. Im ersten Trainingslauf werden die Entscheidungen des Agenten lediglich danach bewertet, ob durch den ausgewählten Auftrag die Produktionsressource umgerüstet werden muss. Der Agent erhält eine Belohnung von $+1$, wenn der ausgewählte Auftrag mit der aktuellen Rüstung der Produktionsressource bearbeitet werden kann. Demgegenüber erhält der Agent eine Bestrafung von -1, wenn für den ausgewählten Auftrag die Produktionsressource umgerüstet werden muss und sich gleichzeitig Aufträge in der Warteschlange befinden, deren Bearbeitung keine Umrüstung erfordern würde. Die Aktion des Agenten wird mit ± 0 bewertet, sofern der ausgewählte Auftrag eine Umrüstung erfordert, sich jedoch keine anderen Aufträge in der Warteschlange befinden, die mit der momentanen Rüstung der Produktionsressource bearbeitet werden können. Der ersten Trainingsstufe liegt die Überlegung zugrunde, dass jeder Rüstvorgang zu Lasten der Produktivzeit geht und das Risiko für verspätete Aufträge ansteigen lässt. Der Agent soll auf der ersten Trainingsstufe lernen, Aufträge so auszuwählen, dass die Anzahl benötigter Rüstvorgänge minimiert wird.

Darauffolgend wird der in der ersten Trainingsstufe angelernte Agent in die zweite Trainingsstufe eingebettet. Erst in der zweiten Trainingsstufe adressiert die Belohnungsfunktion die Minimierung der Gesamtverspätung über alle Aufträge. Für jeden ausgewählten Auftrag wird dessen Verspätung berechnet, sobald dessen Bearbeitung auf der Produktionsressource abgeschlossen wurde. Im Fall einer

Verspätung wird der Agent mit der negativen Differenz zwischen dem Fertigstellungszeitpunkt und der Fertigstellungsfrist des Auftrags bestraft. Andernfalls erhält der Agent eine Belohnung von ± 0.

6.2.3 Diskussion der Ergebnisse

Der Agent wird in der ersten und zweiten Trainingsstufe lediglich auf einer Probleminstanz angelernt. Bezugnehmend auf Tabelle 6.5 (Abschnitt 6.2.1) wird die Probleminstanz für das Training mit minimalen Parametern konfiguriert. Vor diesem Hintergrund soll untersucht werden, ob der angelernte Agent auch auf größeren Probleminstanzen qualitativ hochwertige Lösungen generieren kann. Die Trainingsumgebung beinhaltet sechs Produktionsressourcen ($|M| = 6$), wobei die Ressourcen $i = (1, \ldots, 3)$ Aufträge in kürzerer Zeit $\left(w_{1,\ldots,3} = 0,8\right)$ und die Ressourcen $i = (4, \ldots, 6)$ Aufträge gemäß der geplanten Operationszeit bearbeiten $\left(w_{4,\ldots,6} = 1,0\right)$. In jeder Trainingsepisode werden $|J| = 40$ Aufträge aus $|Fam| = 5$ Auftragsfamilien bearbeitet. Für die Berechnung der Fertigstellungsfristen wird ein durchschnittlicher Verspätungsfaktor von $w_d = 0,1$ sowie ein relativer Wertebereich der Fertigstellungsfristen von $D = 0,15$ veranschlagt. Tabelle 6.7 zeigt die verwendeten Hyperparameter für das Agententraining. Die Bedeutung der meisten Hyperparameter wurde in Abschnitt 5.3.6 (insbesondere in Tabelle 5.4) diskutiert. Der Clipping-Term ϵ und die Anzahl der Epochen pro Trainingsschritt sind PPO-spezifische Parameter. Ihre Bedeutung wird in Anhang C im elektronischen Zusatzmaterial dargelegt.

Tabelle 6.7 Hyperparameter-Einstellungen für den PPO-Algorithmus

Hyperparameter	**Wert**
Anzahl Episoden	2.500 (Trainingsstufe I) bzw. 7.500 (Trainingsstufe II)
Lernrate α	0,0001
Diskontierungsrate γ	0,99
Clipping-Term ϵ	0,3
Trainingslosgröße	5
Anzahl Epochen pro Trainingsschritt	5
Optimierungsverfahren	Adam

Analog zur ersten Fallstudie in Abschnitt 6.1 wurde der Agent in der ersten und zweiten Trainingsstufe jeweils auf einem Kern eines »AMD EPYC 7702P 64-Core«-Prozessors trainiert. Hierbei beansprucht die erste Trainingsstufe eine Rechenzeit von ca. 31 Minuten, während die zweite Trainingsstufe etwa nach 106 Minuten abgeschlossen ist. Abbildung 6.6 zeigt die Lernkurven für beide Trainingsstufen. Die Lernkurve der ersten Trainingsstufe (Abbildung 6.6 (a)) verdeutlicht, dass die Verwendung einer Belohnungsfunktion, welche die Anzahl von Rüstvorgängen minimiert, ebenfalls mit einer Verringerung der Gesamtverspätung über alle Aufträge einhergeht. Jedoch konvergiert die Entscheidungspolitik erst in der zweiten Trainingsstufe, in der die Verspätung von Aufträgen als Belohnung herangezogen wird, zu einem Wert, der sich an $T = 0$ annähert (Abbildung 6.6 (b)).

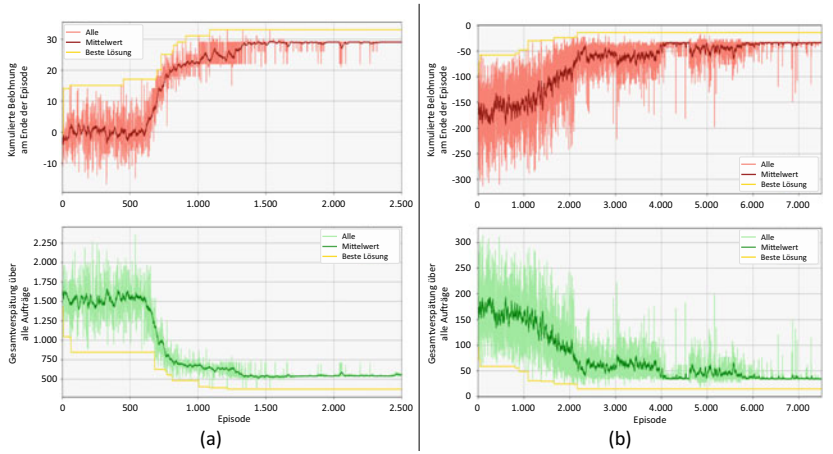


Abbildung 6.6 Trainingsmetriken des PPO-Algorithmus für (a) den Agenten der ersten Trainingsstufe und für (b) den Agenten der zweiten Trainingsstufe

Die Lösungsqualität des angelernten Agenten wird auf 120 Probleminstanzen mit anderen Lösungsverfahren verglichen. Aufgrund der stochastischen Bearbeitungszeiten und Fertigstellungsfristen durchläuft der Agent jede Probleminstanz zehn Mal, um statistische Schwankungen in der Lösungsgüte abzubilden. Bei den verglichenen Lösungsverfahren handelt es sich um die Prioritätsregeln EDD, LST, SPT und MAS_PAR. Die MAS_PAR-Regel ist eine auftragsfamilien-basierte Prioritätsregel, die speziell für das PMP mit Rüstfamilienrestriktion gestaltet

wurde (van der Zee 2015). Bei der MAS_PAR-Regel wird stets eine vollständige Familie auf der Produktionsressource bearbeitet. Sobald eine Produktionsressource verfügbar wird, wählt die MAS_PAR-Regel diejenige Familie, bei der das Verhältnis aus Umrüstzeit zur Anzahl der zu bearbeitenden Aufträge innerhalb der Familie am geringsten ist. Laut van der Zee (2015) erfolgt die Reihenfolgebildung innerhalb einer Familie gemäß der SPT-Regel. In der vorliegenden Arbeit wird stattdessen die EDD-Regel verwendet, da diese explizit die Minimierung von T adressiert.

Abbildung 6.7 veranschaulicht die mittlere Lösungsgüte der verschiedenen Ansätze über alle Probleminstanzen, kategorisiert nach verschiedenen Verspätungsfaktoren w_d und relativen Wertebereichen für die Fertigstellungsfristen D. Jede Konfiguration von w_d und D umfasst 30 Probleminstanzen, die sich hinsichtlich der Anzahl der initialen Aufträge $|b = 0|$, der Anzahl der Produktionsressourcen $|M|$ und der Anzahl von Auftragsfamilien $|Fam|$ unterscheiden. Die Gesamtverspätung über alle Aufträge ist somit stets über 30 Probleminstanzen gemittelt.

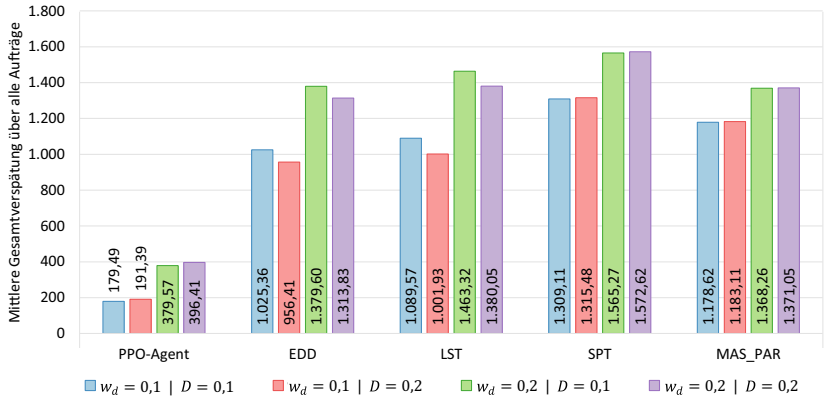


Abbildung 6.7 Lösungsgüte des PPO-Agenten im Vergleich zu verschiedenen Prioritätsregeln

Abbildung 6.7 demonstriert, dass die Lösungsgüte des PPO-Algorithmus um ein Vielfaches besser ist als die Lösungsgüte von herkömmlichen Prioritätsregeln. Hierbei ist der Agent in der Lage auf Probleminstanzen, die während

des Trainings nicht beobachtet wurden, gleichermaßen hochqualitative Ergebnisse zu erzielen. Eine erweiterte tabellarische Darstellung der Ergebnisdaten ist als Anhang I dem elektronischen Zusatzmaterial beigefügt. Hinsichtlich des Berechnungsaufwands können keine nennenswerten Unterschiede zwischen der Anwendung des trainierten Agenten und der Anwendung von Prioritätsregeln ausgemacht werden. Der angelernte Agent benötigt in etwa eine Sekunde Rechenzeit, um eine Probleminstanz vollständig zu lösen. Zusammengefasst kann dem PPO-Algorithmus eine absolute Dominanz gegenüber den untersuchten Prioritätsregel attestiert werden.

6.3 Zweistufiges Hybrid-Flow-Shop-Problem mit familienabhängigen Rüstzeiten

Die dritte Fallstudie repräsentiert eine reale Problemstellung aus der Industrie. Betrachtet wird das Produktionssystem eines Auftragsfertigers für elektronische Komponenten, in dem Bauelemente auf Leiterplatten verlötet werden und danach eine optische Qualitätskontrolle durchlaufen. Aus mathematischer Sicht handelt es sich um ein zweistufiges Hybrid-Flow-Shop (HFS) -Problem. Das Produktionsprogramm ist durch eine hohe Variantenvielfalt geprägt, aufgrund der das Unternehmen eine hohe Anzahl von Auftragsfamilien verwalten muss. Die Produktion einer Auftragsfamilie erfordert eine individuelle Rüstung des Bestückungsautomaten, wodurch zusätzliche Rüstzeiten entstehen. Derzeit basiert die Produktionsablaufplanung des Unternehmens maßgeblich auf menschlichen Erfahrungswissen. Bei der Erstellung eines Produktionsplans wird von einem statischen Auftragshorizont ausgegangen, d. h., dass eine fixe Menge von Aufträgen über eine bestimmte Planungsperiode eingelastet wird. Tatsächlich unterliegt die Produktion jedoch einem dynamischen Auftragshorizont, weil Kunden neue Aufträge direkt im ERP-System des Unternehmens platzieren können. Einige der wichtigsten Kunden setzen zudem sehr enge Lieferfristen, wodurch die Planung zusätzlich erschwert wird. Vor diesem Hintergrund würde das Unternehmen von einer automatisierten und echtzeitfähigen Produktionsablaufplanung profitieren.

Im Rahmen dieser Arbeit wird ein gradientenabhängiger und ein gradientenfreier RL-Ansatz für die agentenbasierte Steuerung der Produktion des Unternehmens untersucht. In der gradientenabhängigen Lösungsstrategie wird der A2C-Algorithmus angewandt, um einen Agenten hinsichtlich der Auswahl von Prioritätsregeln anzulernen (dritte Variante zur Formulierung von Sequenzierungsproblemen als ML-Aufgabe, siehe Abbildung 5.14 in Abschnitt 5.3.2.2). Im Zuge der gradientenfreien Lösungsstrategie wird der NEAT-Algorithmus zur

Gestaltung und Parametrisierung von Agenten für *(i)* die Allokation von Aufträgen zu Produktionsressourcen (Abbildung 5.11 in Abschnitt 5.3.2.1) sowie für *(ii)* die Konstruktion von Auftragssequenzen mittels Priorisierung (zweite Variante zur Formulierung von Sequenzierungsproblemen als ML-Aufgabe, siehe Abbildung 5.13 in Abschnitt 5.3.2.2) untersucht. Die jeweiligen Lösungsstrategien werden in den Unterabschnitten 6.3.2 und 6.3.3 diskutiert. Im Folgenden sollen zunächst die wichtigsten Aspekte des betrachteten Produktionssystems aus Sicht der Ablaufplanung beschrieben werden.

6.3.1 Problembeschreibung

In dem betrachteten HFS muss eine Menge von Aufträgen J zwei Produktionsstufen ($|L| = 2$) durchlaufen. Die erste Produktionsstufe umfasst vier identische Produktionsressourcen $i^{(l=1)} = (1, \ldots, 4)$. Es handelt sich um Maschinen für die automatisierte Leiterplattenbestückung, im Folgenden als SMD (vom englischen Begriff »Surface Mount Device«) abgekürzt. Die zweite Produktionsstufe besteht aus fünf identischen Ressourcen $i^{(l=2)} = (1, \ldots, 5)$ für die automatische optische Inspektion (AOI) der bestückten Leiterplatten. Die Operationszeiten sind individuell für jeden Auftrag und jede Produktionsstufe festgelegt $\left(o_j^{(l)} \forall j = (1, \ldots, |J|) \forall l = (1, 2)\right)$, jedoch sind innerhalb derselben Produktionsstufe die Operationszeiten eines Auftrags für alle Produktionsressourcen identisch.

In der ersten Produktionsstufe finden kleine und große Rüstprozesse statt. Für die Bearbeitung eines Auftrags muss eine SMD-Ressource entsprechend der Familie des Auftrags gerüstet werden. Analog zu dem in Abschnitt 6.2.1 beschriebenen PMP gehört jeder Auftrag genau einer Auftragsfamilie an und jede Produktionsressource ist zur selben Zeit genau für eine Auftragsfamilie gerüstet. Hierbei beschreibt Fam die Menge aller Familien, f_j die Familie von Auftrag j und f_i die Rüstung von SMD-Ressource $i^{(l=1)}$. Jeder Familie f gehören ein oder mehrere Aufträge an. Sofern ein Auftrag auf einer SMD-Ressource eingelastet wird, dessen Familie nicht der Rüstung der Ressource entspricht ($f_i \neq f_j$), muss eine große Rüstzeit $s_{major}^{(1)}$ von 65 Minuten veranschlagt werden. Nach Ablauf der Rüstzeit ist die SMD-Ressource mit der passenden Rüstung ausgestattet, um den aktuellen Auftrag zu bearbeiten ($f_i = f_j$). Sofern f_i und f_j von Beginn an identisch sind, muss weiterhin eine kleine Rüstzeit $s_{minor}^{(1)}$ von 20 Minuten für die Vorbereitung der Ressource berücksichtigt werden. Die AOI-Ressourcen der zweiten Produktionsstufe müssen ebenfalls für die Bearbeitung eines jeden

Auftrags vorbereitet werden. Die Vorbereitungszeit $s^{(2)}$ beträgt stets 25 Minuten und ist unabhängig von den Familien sowie der Reihenfolge der zu bearbeitenden Aufträge.

Jeder Auftrag besitzt eine Fertigstellungsfrist d_j, bis zu welcher die zweite Produktionsstufe abgeschlossen werden soll. Die Güte eines Produktionsablaufplans wird anhand dessen Gesamtdauer C_{max} und der resultierenden Gesamtverspätung über alle Aufträge T bewertet. Ferner gelten für das HFS-Problem die folgenden weiteren Nebenbedingungen:

- Alle Aufträge sind von Beginn an ($t_{sim} = 0$) verfügbar.
- Nach dem Start der Bearbeitung des ersten Auftrags in der ersten Produktionsstufe kommen keine neuen Aufträge in das System.
- SMD-Ressourcen sind zum Start der Produktion für keine Auftragsfamilie gerüstet.
- Jede SMD- und AOI-Ressource kann nur einen Auftrag zur gleichen Zeit produzieren.
- Die Bearbeitung von Aufträgen kann nicht unterbrochen werden. Ein Auftrag muss zunächst vollständig bearbeitet werden, bevor der nächste Auftrag für die Produktion ausgewählt werden kann.
- Zu jedem Zeitpunkt kann maximal eine SMD-Ressource mit derselben Familie gerüstet sein.
- Das Problem ist als Permutations-Flow-Shop klassifiziert, d. h. Aufträge, welche die erste Produktionsstufe abschließen, werden derjenigen Produktionsressource zugewiesen, die am frühesten verfügbar wird (Emmons und Vairaktarakis 2013). Diese Nebenbedingung spiegelt den Produktionsprozess des Unternehmens am besten wider und ist dadurch zu begründen, dass in der Leiterplattenbestückungsindustrie üblicherweise die Qualität der Ablaufplanung für SMD-Ressourcen den größten Einfluss auf die Gesamtleistung des Produktionssystems ausübt (Csaszar et al. 2000).

Im Rahmen dieser Arbeit werden vier Probleminstanzen für das geschilderte HFS-Problem untersucht. Die Probleminstanzen stammen aus dem ERP-System des Unternehmens und repräsentieren historische Produktionsdaten. Jede Probleminstanz deckt eine Planungsperiode von ca. drei Wochen ab. Tabelle 6.8 beschreibt die Probleminstanzen anhand von einigen statistischen Kennzahlen. Die Kurtosis K der jeweiligen Auftragsattribute und die mittlere normalisierte Kurtosis $\overline{K}$ der jeweiligen Datensätze sind ausschließlich für die Anwendung des NEAT-Algorithmus von Bedeutung. Beide Kennzahlen werden in Abschnitt 6.3.3 im Zuge der Experimente und Hyperparameter-Einstellungen

für das Agententraining mit NEAT erläutert. Die vollständigen Datensätze sind online verfügbar[1]. Weil es sich bei dem vorliegenden HFS-Problem um einen Permutations-Flow-Shop handelt, werden die in den Abschnitten 6.3.2 und 6.3.3 vorgestellten Lösungsansätze lediglich für das Training von Agenten für die erste Produktionsstufe (SMD) untersucht.

6.3.2 Anwendung des A2C-Algorithmus zur Lösung des Problems

Im Rahmen der gradientenabhängigen Lösungsstrategie wird der A2C-Algorithmus für das Training von Agenten hinsichtlich der Auswahl von Prioritätsregeln untersucht. Eine detaillierte Beschreibung des A2C-Algorithmus ist als Anhang B dem elektronischen Zusatzmaterial beigefügt. Die agentenbasierte Auswahl von Prioritätsregeln beschreibt die dritte Variante zur Formulierung von Sequenzierungsproblemen als ML-Aufgabe (Abbildung 5.14 in Abschnitt 5.3.2.2). Die folgende Darstellung der gradientenabhängigen Lösungsstrategie basiert im Wesentlichen auf einer vorausgegangenen Publikation, die im Rahmen dieser Arbeit entstanden ist (Gerpott, Lang et al. 2022).

Entwurf der Agentenumgebung

Wie im Folgenden aufgezeigt wird, erfordert das vorliegende HFS-Problem eine integrative Betrachtung des Entwurfs der Agentenumgebung und der Gestaltung der Belohnungsfunktion. Grundsätzlich kommen sowohl das Produktionsbereichsmodell mit lokalen Warteschlangen als auch das Produktionsbereichsmodell mit zentraler Warteschlange (Abbildung 5.3 (a) bzw. (b) in Abschnitt 5.2) für die Implementierung des HFS-Problems als Agentenumgebung in Frage. Da die dritte Formulierungsvariante für Sequenzierungsprobleme eine kombinierte Ressourcenbelegungs-, Reihenfolge- und Losgrößenplanung erlaubt, ist es theoretisch ausreichend, die SMD-Stufe als Produktionsbereich mit zentraler Warteschlange zu modellieren. Die Restriktion, dass zu jedem Zeitpunkt maximal eine SMD-Ressource mit derselben Familie gerüstet sein darf, erfordert jedoch eine Erweiterung dieses Modells. Sofern ein ausgewählter Auftrag die Umrüstung einer SMD-Ressource bedingt, muss zunächst überprüft werden, ob zum aktuellen Zeitpunkt keine benachbarte SMD-Ressource Aufträge derselben Familie produziert. Ist dies nicht der Fall, können zwei Ansätze verfolgt werden, welche die Einhaltung der Restriktion sicherstellen:

[1] https://www.ilm.ovgu.de/hicss_problem_instances.html.

Tabelle 6.8 Aggregierte Darstellung der HFS-Probleminstanzen (in Anlehnung an Lang et al. 2021b)

		Datensatz 1	Datensatz 2	Datensatz 3	Datensatz 4
Anzahl Aufträge $\|J\|$		164	170	175	143
Anzahl Auftragsfamilien $\|Fam\|$		41	37	36	35
Fertigstellungsfrist d_j	min	1.305	1.305	1.305	1.305
	max	27.405	27.405	27.405	27.405
	μ	13.877,56	13.799,26	14.034,34	14.072,10
	σ	8.051,04	7.852,94	7.835,24	7.690,56
	K	0,20	0,19	0,18	0,16
SMD-Bearbeitungszeiten $o_j^{(1)}$	min	4	2	4	4
	max	3.142	3.736	3.293	3.209
	μ	333,45	366,74	350,14	393,36
	σ	486,02	506,26	517,21	521,68
	K	69,17	56,08	62,68	34,97
AOI-Bearbeitungszeiten $o_j^{(2)}$	min	4	3	5	3
	max	4.351	5.590	3.528	4.300
	μ	442,24	521,79	427,07	554,50
	σ	648,89	800,88	584,09	736,85
	K	82,85	98,04	35,49	34,23
$\overline{K}$		0,4588	0,3928	0,3913	0,4950

(i) Die umzurüstende SMD-Ressource wartet solange, bis das angeforderte Rüst-Kit verfügbar wird. Hierbei kann der Fall eintreten, dass mehrere SMD-Ressourcen zur gleichen Zeit auf dasselbe Rüst-Kit warten. Um Konflikte zu vermeiden wird für jede Auftragsfamilie eine Warteschlange erstellt, in welche sich die SMD-Ressource einreiht, sofern das Rüst-Kit nicht unmittelbar zur Verfügung steht. Alle Warteschlangen für Rüst-Kits werden nach dem FIFO-Prinzip abgearbeitet.

(ii) Jede SMD-Ressource ist mit einem zusätzlichen lokalen Puffer ausgestattet. Sofern eine benachbarte SMD-Ressource zum gegenwärtigen Zeitpunkt einen anderen Auftrag derselben Familie bearbeitet, wird der ausgewählte Auftrag in den lokalen Puffer der bereits gerüsteten SMD-Ressource verschoben. Die verfügbare SMD-Ressource, für welche der verschobene Auftrag ursprünglich ausgewählt wurde, erhält durch den Agenten den Auftrag mit der nächsthöchsten Priorität. Hierbei ist es möglich, dass der alternativ zugewiesene Auftrag ebenfalls verschoben wird, sofern eine andere SMD-Ressource zum aktuellen Zeitpunkt einen anderen Auftrag derselben Familie bearbeitet. Der Prozess wiederholt sich solange, bis ein Auftrag ausgewählt wurde, für dessen Familie keine andere SMD-Ressource gerüstet ist oder bis keine Aufträge in der zentralen Warteschlange vorhanden sind. SMD-Ressourcen arbeiten stets zuerst die Aufträge in ihrem lokalen Puffer nach dem FIFO-Prinzip ab, bevor sie einen Auftrag aus der zentralen Warteschlange für SMD-Ressourcen anfordern.

Ansatz (i) besitzt gegenüber (ii) den Vorteil, dass Entscheidungen stets im Sinne des Agenten getroffen werden. Hingegen werden in Ansatz (ii) Agentenentscheidungen verworfen und durch alternative Entscheidungen ersetzt, sofern ein Rüstkonflikt eintreten würde. Dies kann den Lernprozess erschweren, da alternativ getroffene Entscheidungen nicht unmittelbar durch den Agenten beobachtet werden können. Hingegen erlaubt Ansatz (ii) eine einfachere Berechnung von zeitbasierten Belohnungen. Durch den A2C-Algorithmus trainierte Agenten sollen unter anderem auf Basis der Unpünktlichkeit von Aufträgen in der ersten Produktionsstufe belohnt werden. Die anteilsmäßige Unpünktlichkeit eines Auftrags kann in Ansatz (ii) gemäß Formel (5.4) berechnet werden, wenn der Auftrag auf der verfügbaren SMD-Ressource bearbeitet werden kann, bzw. gemäß Formel (5.5), sofern der Auftrag auf eine andere Ressource umverteilt werden muss. In Ansatz (ii) ist bei einem Rüstkonflikt Formel (5.5) nicht anwendbar, da keine zusätzlichen lokalen Puffer existieren. Alternativ müsste ein Berechnungsverfahren implementiert werden, das die voraussichtliche Wartezeit abschätzt, bis das angeforderte Rüst-Kit für die SMD-Ressource verfügbar wird. Eine exakte

Berechnung der Wartezeit ist nicht möglich, da nicht vorhersehbar ist, ob die SMD-Ressource, welche das angeforderte Rüst-Kit zum Zeitpunkt nutzt, nach dem aktuell in Bearbeitung befindlichen Auftrag einen weiteren Auftrag derselben Familie auswählt. Vor diesem Hintergrund wird für das Training mit dem A2C-Algorithmus die Agentenumgebung gemäß Ansatz (ii) angepasst, um Rüstkonflikte zu vermeiden. Demzufolge wird jede SMD-Ressource der ersten Produktionsstufe mit einem zusätzlichen lokalen Puffer ausgestattet.

Die zweite Produktionsstufe wird ebenfalls als Produktionsbereich mit zentraler Warteschlange modelliert. In der zweiten Produktionsstufe werden Aufträge nach dem FIFO-Prinzip abgearbeitet, d. h. sobald eine AOI-Ressource verfügbar ist, wird der vorderste Auftrag aus der zentralen Warteschlange für AOI-Ressourcen ausgewählt. Abbildung 6.8 zeigt einen Ausschnitt des animierten Simulationsmodells.

Definition der maschinellen Lernaufgabe und des Aktionsraums des Agenten

Wie bereits einleitend erwähnt, soll im Rahmen der gradientenabhängigen Lösungsstrategie die dritte Variante zur Formulierung von Sequenzierungsproblemen als ML-Aufgabe evaluiert werden. Jedes Mal, wenn eine SMD-Ressource verfügbar wird, entscheidet sich der Agent in Abhängigkeit vom aktuellen Zustandsvektor für eine Prioritätsregel. Durch die Anwendung der Prioritätsregel wird der nächste zu produzierende Auftrag für die verfügbare SMD-Ressource ausgewählt. Der Agent besitzt somit einen diskreten Aktionsraum.

Für die vorliegende Problemstellung werden sechs Prioritätsregeln definiert, zwischen denen sich der Agent entscheiden kann. Das Agentenmodell besitzt demzufolge sechs Ausgabeneuronen. Die untersuchten Prioritätsregeln werden in Tabelle 6.9 gelistet. In den mathematischen Definitionen der Prioritätsregeln bildet die Indexvariable q auf alle auswählbaren Aufträge in der SMD-Warteschlange ab. Die Prozesszeit $p_q^{(l)}$ subsumiert die Operationszeit $o_q^{(l)}$ und die Rüstzeit $s_{minor}^{(1)}$, $s_{major}^{(1)}$ oder $s^{(2)}$ (abhängig von der Produktionsstufe und ggf. von der Familie des Auftrags und der Rüstung der verfügbaren SMD-Ressource). Da die Güte eines Produktionsablaufplans sowohl anhand dessen Gesamtdauer C_{max} als auch anhand der resultierenden Gesamtverspätung über alle Aufträge T bewertet wird, werden im Aktionsraum des Agenten Prioritätsregeln für beide Optimierungskriterien berücksichtigt. Die Prioritätsregeln, die mit den Aktionsindizes 1–3 assoziiert werden, tragen zu einer Minimierung von T bei. Die mit den Aktionsindizes 4 und 5 assoziierten Prioritätsregeln adressieren die Minimierung von C_{max}.

Hinsichtlich der LPT-Regel hat sich im Rahmen von Voruntersuchungen herausgestellt, dass insbesondere in zweistufigen Flow-Shop-Systemen die Auswahl

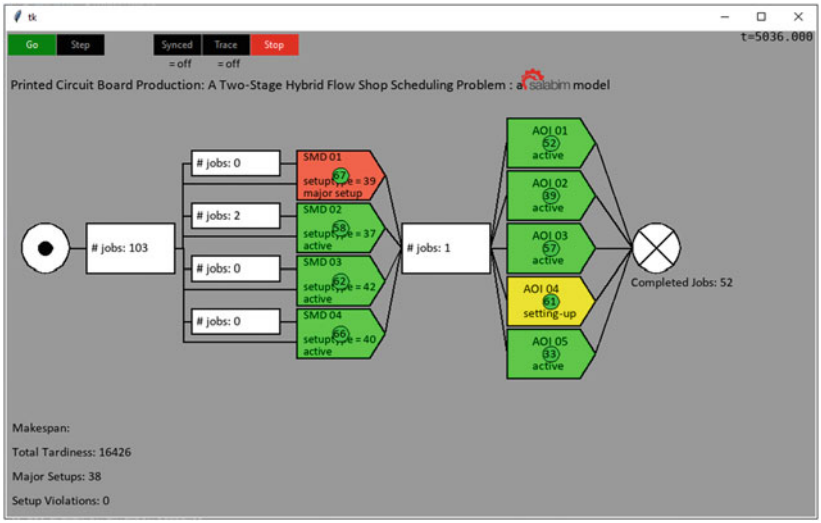


Abbildung 6.8 Animierte DES-Agentenumgebung des zweistufigen HFS-Problems für die A2C- Implementierung

des Auftrags mit der längsten Operationszeit teilweise zu einer Verringerung der Gesamtdauer des Ablaufplans führen kann. Die Anwendung der LPT-Regel kann u. U. verhindern, dass einzelne Aufträge mit besonders langen Operationszeiten erst am Ende des Produktionszyklus auf der AOI-Stufe bearbeitet werden, während das restliche Produktionsprogramm bereits abgearbeitet wurde und die verbleibenden AOI-Ressourcen sich im Leerlauf befinden. Die mit der sechsten Aktion assoziierte FAM_EDD-Regel stellt einen Kompromiss zwischen der Minimierung von C_{max} und T her, indem lediglich derjenige Auftrag mit der geringsten Fertigstellungsfrist gewählt wird, der keine große Umrüstung der verfügbaren SMD-Ressource erfordert. In Formel (6.11) bezeichnet der Index SMD die verfügbare SMD-Ressource, die den nächsten zu bearbeitenden Auftrag anfordert.

Zustandsraum des Agenten

Der in jedem Zeitschritt eingelesene Zustand sollte ebensolche Informationen berücksichtigen, durch die der Agent ein möglichst umfassendes Bild der Produktionsumgebung erhält, um auf geeignete Prioritätsregeln für die Auswahl von Aufträgen schließen zu können. In der vorliegenden Lösungsstrategie beinhaltet

Tabelle 6.9 Aktionsraum des Agenten im HFS-Problem für die A2C-Implementierung (in Anlehnung an Gerpott, Lang et al. 2022)

Aktionsindex	Prioritätsregel	Kürzel	Mathematische Definition	
1	Earliest Due Date (Früheste Fertigstellungsfrist)	EDD	$\min_q \left(d_q\right)$	(6.5)
2	Least Slack Time (Geringste Schlupfzeit)	LST	$\min_q \left(d_q - \left(t_{sim} + \sum_{l=1}^{\lvert L\rvert} p_q^{(l)}\right)\right)$	(6.6)
3	Smallest Critical Ratio (Kleinstes kritisches Verhältnis)	SCR	$\min_q \left(\frac{d_q - t_{sim}}{\sum_{l=1}^{\lvert L\rvert} p_q^{(l)}}\right)$	(6.7)
4	Shortest Processing Time (Kürzeste Operationszeit)	SPT	$\min_q \left(p_q^{(1)}\right)$	(6.8)
5	Longest Processing Time (Längste Operationszeit)	LPT	$\max_q \left(p_q^{(1)}\right)$	(6.9)
6	Family Earliest Due Date (Früheste Fertigstellungsfrist der gerüsteten Familie)	FAM_EDD	$\min_{\forall q: f_q = f_{SMD}} \left(d_q\right)$	(6.10)

jeder Zustandsvektor auftragsbezogene Warteschlangenstatistiken sowie verschiedene Attribute der SMD-Ressourcen. Tabelle 6.10 listet alle Warteschlangen- und SMD-Attribute auf, die in die Zustandsbildung einfließen.

Belohnungsfunktion des Agenten

Um gleichermaßen die Gesamtdauer des Ablaufplans C_{max} und die Gesamtverspätung über alle Aufträge T bei der Bewertung von Agentenentscheidungen zu adressieren, wird im Folgenden eine multikriterielle Belohnungsfunktion $\mathcal{R}_{Total}$ hergeleitet, welche die Optimierung beider Zielkriterien ausbalanciert.

$$\mathcal{R}_{Total} = \omega_C \mathcal{R}_C + \omega_T \mathcal{R}_T \tag{6.11}$$

Hierbei repräsentieren $\mathcal{R}_C$ und $\mathcal{R}_T$ die Belohnungsfunktionen für die Minimierung von C_{max} bzw. von T. Die Koeffizienten ω_C und ω_T gewichten den Einfluss beider Belohnungsfunktionen auf die Gesamtbelohnung. Im Rahmen dieser Arbeit werden beide Koeffizienten stets mit dem Wert 0,5 initialisiert, wodurch $\mathcal{R}_C$ und $\mathcal{R}_T$ gleichermaßen in $\mathcal{R}_{Total}$ eingehen. Die Teilbelohnungsfunktion $\mathcal{R}_C$ für die Optimierung von C_{max} ist so gestaltet, dass der Agent motiviert ist, alle Ressourcen einer Produktionsstufe möglichst hoch und gleichmäßig auszulasten.

$$\mathcal{R}_C = -\frac{Scale_C}{|L|} * \sum_{l=1}^{|L|} \frac{\left| \overline{o}_{last}^{(l)} - \overline{o}^{(l)} \right|}{o_{max}^{(l)} - \overline{o}^{(l)}} \tag{6.12}$$

mit

$$Scale_C = 10;\ \overline{o}_{last}^{(l)} = \frac{\sum_{k=1}^{5} o_k^{(l)}}{5};\ \overline{o}^{(l)} = \frac{\sum_{j=1}^{|J|} o_j^{(l)}}{|J|};\ o_{max}^{(l)} = \max_j \left(o_j^{(l)} \right) \tag{6.13}$$

Um eine möglichst hohe und gleichmäßige Ressourcenauslastung zu erreichen, bildet die Teilbelohnungsfunktion in Formel (6.13) für jede Produktionsstufe l die durchschnittliche Operationszeit über alle Aufträge $\overline{o}^{(l)}$ sowie über die letzten fünf ausgewählten Aufträge $\overline{o}_{last}^{(l)}$. Je Produktionsstufe wird die absolute Differenz zwischen beiden Mittelwerten berechnet und normalisiert. Über den Skalierungsfaktor $Scale_C$ wird die Belohnung in den Wertebereich $(-10, 0)$ verschoben. Die Intention hinter der Berechnungsvorschrift ist, dass der Agent umso kleinere Bestrafungen erhält, desto ausgewogener die Auswahl von Aufträgen aus den zentralen Warteschlangen der SMD- und AOI-Stufe erfolgt. Die Auswahl von Aufträgen ist umso ausgewogener, je dichter sich der gleitende Mittelwert der Bearbeitungszeiten über die zuletzt ausgewählten Aufträge am tatsächlichen Mittelwert der Bearbeitungszeiten über alle Aufträge befindet. Obgleich die Entscheidungen des Agenten unmittelbar nur die erste Produktionsstufe betreffen, bewertet die Belohnungsfunktion $\mathcal{R}_C$ ebenfalls die Ausgewogenheit der Auftragsauswahl in der zweiten Produktionsstufe. Dies hat den Hintergrund, dass die Auftragsauswahl in der ersten Stufe maßgeblich beeinflusst, wie viele und welche Aufträge in der zweiten Stufe zur Auswahl stehen. Vor diesem Hintergrund

Tabelle 6.10 Attribute zur Bildung von Zuständen im HFS-Problem für die A2C-Implementierung (in Anlehnung an Gerpott, Lang et al. 2022)

Attributsart	Eingabeneuronen-Indizes	Attributsbezeichnung	Attributsstatistiken
Warteschlangen-statistiken	1–5	Verspätung über alle Aufträge	#, min, max, μ, σ
	6–10	Verspätung über alle Aufträge mit derselben Familie wie die Rüstung der verfügbaren SMD-Ressource	#, min, max, μ, σ
	11–15	Verfrühung über alle Aufträge	#, min, max, μ, σ
	16–19	SMD-Operationszeiten über alle Aufträge	min, max, μ, σ
	20–24	SMD-Operationszeiten über alle Aufträge mit derselben Familie wie die Rüstung der verfügbaren SMD-Ressource	#, min, max, μ, σ
	25–28	AOI-Operationszeiten über alle Aufträge	min, max, μ, σ
	29–32	Anzahl von Aufträgen innerhalb einer Auftragsfamilie	min, max, μ, σ
	33	Anzahl von Aufträgen	#
	34	Anzahl von Auftragsfamilien	#
SMD-Ressourcen-attribute	35–37	Verbleibende Bearbeitungszeit der belegten SMD-Ressourcen	Wert1, Wert2, Wert3
	38–42	Verbleibende Bearbeitungszeit der AOI-Ressourcen	Wert1, Wert2, Wert3, Wert4, Wert5

– Anzahl, min – Minimum, max – Maximum, μ – Mittelwert, σ – Standardabweichung

kann der Fall eintreten, dass Auswahlentscheidungen, welche die erste Produktionsstufe nur suboptimal auslasten, zu einer besseren Auslastung der zweiten Produktionsstufe führen und somit in der Gesamtbilanz besser sind.

Die Teilbelohnungsfunktion $\mathcal{R}_T$ für die Optimierung von T basiert im Wesentlichen auf Formel (5.4) zur Berechnung der anteilsmäßigen Unpünktlichkeit einer Auftragsoperation $L_{j,o}$ und der hierauf aufbauenden Belohnungsfunktion $\mathcal{R}_L(j)$ aus Formel (5.7). Die konkreten Berechnungsvorschriften sind in den folgenden Formeln (6.15) und (6.16) dargelegt.

$$\mathcal{R}_T = \begin{cases} \frac{Scale_T * L_j^{(1)}}{\widehat{E}_{\max}}, & wenn\ L_j^{(1)} \leq 0 \\ -W_T * \left(\frac{Scale_T * L_j^{(1)}}{\widehat{T}_{\max}} \right), & sonst \end{cases} \tag{6.14}$$

mit

$$Scale_T = 10;\ w_T = 4;\ L_j^{(1)} = \left(t_{sim} - d_j\right) * \left(\frac{p_j^{(1)}}{\sum_{l=1}^{|L|} p_j^{(l)}} \right) \tag{6.15}$$

Formel (6.15) erweitert Formel (5.7) um ein Normalisierungsverfahren, das Bestrafungen in den Wertebereich $(-10, 0)$ verschiebt. Die maximale Verfrühung und Verspätung werden durch die in Abschnitt 5.3.5.1 entwickelten Formeln (5.10) und (5.11) geschätzt. Ferner verändert sich die Berechnungsvorschrift für die anteilsmäßige Verspätung. Anstelle der Verspätung einer Auftragsoperation wird die Verspätung des Auftrags auf der ersten Produktionsstufe berechnet. Diese Veränderung ist jedoch rein formell und der mathematischen Definition des HFS-Problems geschuldet. Da beide Produktionsstufen mit jeweils genau einer Auftragsoperation assoziiert werden, handelt es sich im Grunde genommen um dieselbe Berechnungsvorschrift. In Formel (6.15) werden sowohl verfrühte als auch verspätete Aufträge bestraft, wobei verspätete Aufträge um das Vierfache härter bestraft werden als verfrühte Aufträge.

Versteckte Architektur des Agenten und Hyperparameter-Einstellungen für das Agententraining

Für das Agententraining werden die Hyperparameter aus Tabelle 6.11 verwendet. Tabelle 6.11 gibt ebenfalls Auskunft über die versteckte Architektur des Agenten. Im Folgenden werden die wichtigsten Gestaltungsentscheidungen kurz erläutert.

Analog zum PPO-Algorithmus besteht der Agent aus einem Actor- und einem Critic-Modell. In Kontrast zur Architektur des Actor-Critic-Agenten für das in

Abschnitt 6.1.4 skizzierte PMP (Abbildung 6.5), teilen sich das Actor- und Critic-Modell keine gemeinsame Eingabeschicht. Ferner besitzen beide Modelle keine rekurrenten Neuronen. Die versteckte Architektur des Actor- und Critic-Modells ähnelt der des Agenten für die Ressourcenbelegungsplanung des in Abschnitt 6.1 untersuchten FJS-Problems. Anstatt der ELU-Funktion werden die Ausgaben der versteckten Schichten durch eine ReLU-Funktion verarbeitet.

Für die vorliegende Fallstudie wird der A2C-Algorithmus der Programmbibliothek Stable Baselines verwendet. Die Lernparameter c_V und c_H sind spezifische Parameter der Stable-Baselines-Implementierung. Die in Anhang B (im elektronischen Zusatzmaterial) definierte Lernsignalfunktion des Actor-Modells

Tabelle 6.11 Hyperparameter-Einstellungen für den A2C-Algorithmus (in Anlehnung an Gerpott, Lang et al. 2022)

Hyperparameter	**Wert**
Versteckte Architektur des Actor- und Critic-Modells	
Anzahl versteckter Schichten	2
Anzahl Neuronen je versteckter Schicht	32
Neuronale Aktivierungsfunktion	ReLU
Gemeinsame Eingabeschicht für Actor- und Critic-Modell	FALSCH
Lernparameter	
Anzahl Episoden	2.500
Lernrate α	0,00001
Diskontierungsrate γ	0,99
Zustandsnutzenfunktion-Koeffizient c_V	0,5
Entropie-Koeffizient c_H	0,0
Trainingslosgröße	10
Optimierungsverfahren	RMSProp

(Formel (B.3)) wird insofern erweitert, dass wahlweise ein Entropie-Term dem Lernsignal hinzugefügt werden kann, der die Exploration des Agenten forciert. Der Einfluss der Entropie wird über den Koeffizienten c_H gesteuert. Zusätzlich kann der Einfluss der Zustandsnutzenfunktion auf das Lernsignal über den Koeffizienten c_V feinjustiert werden. Die vollständige Lernsignalfunktion des Actor-Modells in der Stable-Baselines-Implementierung kann somit wie folgt formuliert werden.

$$\delta_A = \frac{1}{n} \sum_{k=t-n}^{t} (c_V * Adv_k * \log \pi(A_k|S_k;\theta_A) + c_H * H(\pi(S_k;\theta_A)) \quad (6.16)$$

Der Term $H(\pi(S_k;\theta_A)$ repräsentiert die Entropie der aus der Entscheidungspolitik $\pi(a|s;\theta_A)$ resultierenden Multinominalverteilung unter dem gegebenen Zustand S_k. In Vorexperimenten hat sich herausgestellt, dass sich die Lösungsgüte des Agenten nicht durch eine entropie-induzierte Exploration verbessert. Aus diesem Grund wird in den finalen Experimenten auf die Berechnung einer Entropie verzichtet ($c_H = 0$). Als Optimierungsverfahren wird der RMSProp-Algorithmus verwendet. Hierbei handelt es sich um das Standardoptimierungsverfahren für die A2C-Implementierung in Stable Baselines.

Trainingsergebnisse und Evaluation

Der Agent wurde auf einem Kern eines Prozessors des Modells »Intel i5–6500« trainiert. Der Prozessorkern besitzt eine maximale Taktfrequenz von 3,20 GHz. Die Simulation von 2.500 Trainingsepisoden beansprucht in etwa 40 Minuten Rechenzeit. Für das Training wurde ausschließlich Datensatz 1 verwendet. Die Datensätze 2–4 dienen der Evaluation des trainierten Agenten. Abbildung 6.9 veranschaulicht den Trainingsverlauf des Agenten anhand von sechs Lernkurven. Die linke Seite von Abbildung 6.9 zeigt die Gesamtdauer des Ablaufplans C_{max}, die Gesamtverspätung über alle Aufträge T sowie die kumulierte Belohnung am Ende einer Episode G_T. Die drei Kennzahlen zeigen, dass zwischen Episode 500 und 600 der Agent sich temporär einer lokalen optimalen Strategie annähert, im weiteren Verlauf jedoch zu einer schlechteren finalen Strategie konvergiert. Die Verschlechterung des Agenten betrifft ausschließlich das Optimierungskriterium C_{max}. Die Gesamtverspätung über alle Aufträge pendelt sich etwa nach 500 Episoden auf den Wert null ein.

Die rechte Seite von Abbildung 6.9 zeigt, für welche Prioritätsregeln sich der Agent in jeder Episode entscheidet, wie viele Aufträge verfrüht und verspätet fertiggestellt werden und wie sich die maximale und durchschnittliche Verfrühung E_{max} bzw. E_{avg} sowie die maximale und durchschnittliche Verspätung T_{max} bzw. T_{avg} über die Episoden entwickeln. In den besten Strategien zwischen Episode 500 und 600 wählt der Agent einigermaßen ausgewogen zwischen den verfügbaren Prioritätsregeln, wobei eine Präferenz für diejenigen Prioritätsregeln zu beobachten ist, welche die Gesamtverspätung über alle Aufträge minimieren. In der schlechteren finalen Strategie wählt der Agent in den meisten Fällen die LST-Prioritätsregel und in nur wenigen Entscheidungssituationen die EDD-Prioritätsregel. Die Ursachen, warum der Agent nach ca. 500 Episoden zu

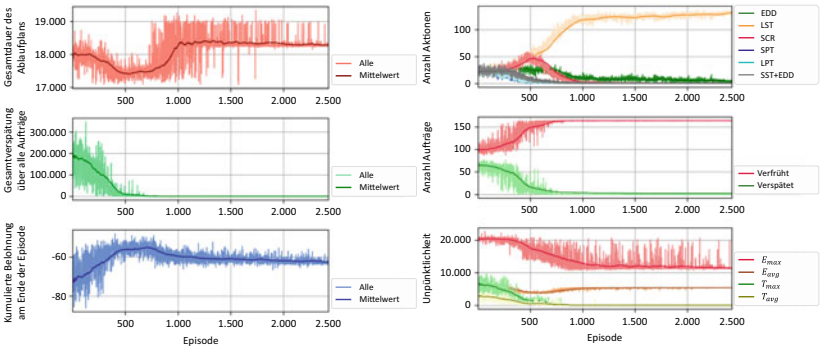


Abbildung 6.9 Trainingsmetriken des A2C-trainierten Agenten auf Datensatz 1 des HFS-Problems (in Anlehnung an Gerpott, Lang et al. 2022)

einer schlechteren Entscheidungspolitik konvergiert, bedarf einer tieferen Analyse und soll Gegenstand zukünftiger Forschungsarbeiten sein. Die Entwicklung der Unpünktlichkeit der Aufträge entspricht dem Verlauf der Gesamtverspätung über alle Aufträge. Sowohl die Kurve für T_{max} als auch für T_{avg} konvergieren im Verlauf der Episoden gegen null.

Für die Evaluation der Lösungsgüte auf den Datensätzen 2–4 und den Vergleich mit anderen Lösungsverfahren wird die als am besten identifizierte Entscheidungspolitik zwischen Episode 500 und 600 herangezogen. Tabelle 6.12 zeigt die Lösungsgüte des trainierten Agenten, gemessen an der Gesamtdauer des Ablaufplans C_{max} und der Gesamtverspätung über alle Aufträge T, auf allen vier Datensätzen.

Tabelle 6.12 Lösungsgüte des A2C-trainierten Agenten auf allen vier Datensätzen des HFS-Problems

	Gesamtdauer des Ablaufplans C_{max}	**Gesamtverspätung über alle Aufträge TT**
Datensatz 1	17.693	0
Datensatz 2	21.355	2.371
Datensatz 3	19.159	2.413
Datensatz 4	18.729	0

Die Ergebnisse verdeutlichen, dass der Agent ebenfalls auf Datensatz 4, der während des Trainings nicht beobachtet wurde, einen Produktionsplan identifizieren kann, bei welchem keine Verspätungen auftreten. Verspätungen treten hingegen in den Produktionsplänen für die Datensätzen 2 und 3 auf, die ebenfalls nicht während des Trainings beobachtet wurden. In Abschnitt 6.3.4 wird jedoch aufgezeigt, dass die Lösungsgüte des A2C-Algorithmus mit den meisten konventionellen Lösungsverfahren konkurrieren kann oder diese sogar übertrifft. Zuvor soll im Folgenden jedoch erst die gradientenfreie Lösungsstrategie beschrieben werden.

6.3.3 Anwendung des NEAT-Algorithmus zur Lösung des Problems

Der NEAT-Algorithmus wird stellvertretend für die Klasse gradientenfreier RL-Verfahren für das vorliegende HFS-Problem untersucht. Vor dem Hintergrund, dass das HFS-Problem den einzigen Untersuchungsgegenstand für gradientenfreie RL-Verfahren repräsentiert, werden im Folgenden mehrere Varianten zur Formulierung von Entscheidungsproblemen der Produktionsablaufplanung als ML-Aufgabe untersucht. Die nachfolgenden Ausführungen basieren im Wesentlichen auf zwei vorausgegangenen Publikationen, die im Rahmen dieser Arbeit entstanden sind (Lang et al. 2020b; Lang et al. 2021b).

Entwurf der Agentenumgebung

Zunächst sollen die Unterschiede hinsichtlich der Implementierung der Agentenumgebung im Vergleich zur gradientenabhängigen Lösungsstrategie dargestellt werden. Einleitend zeigt Abbildung 6.10 einen Ausschnitt aus dem animierten Simulationsmodell.

Erstens werden die SMD- und AOI-Stufe nicht als Produktionsbereiche mit zentraler Warteschlange implementiert. Stattdessen wird für die Modellierung das Produktionsbereichsmodell mit lokalen Warteschlangen für jede Produktionsressource (Abbildung 5.3 (b) in Abschnitt 5.2) verwendet. Der Grund für diese Entscheidung ist, dass der NEAT-Algorithmus anhand von verschiedenen Formulierungsvarianten für ML-Aufgaben untersucht werden soll. Durch das Produktionsbereichsmodell gemäß Abbildung 5.3 (b) kann der NEAT-Algorithmus sowohl für das Training von Agenten mit diskreten Aktionsraum (für die Auftragsallokation) als auch für das Training von Agenten mit kontinuierlichem Aktionsraum (für die prioritätsbasierte Auftragssequenzierung) untersucht werden. Zweitens ist es nicht notwendig die Restriktion, dass zu jedem Zeitpunkt

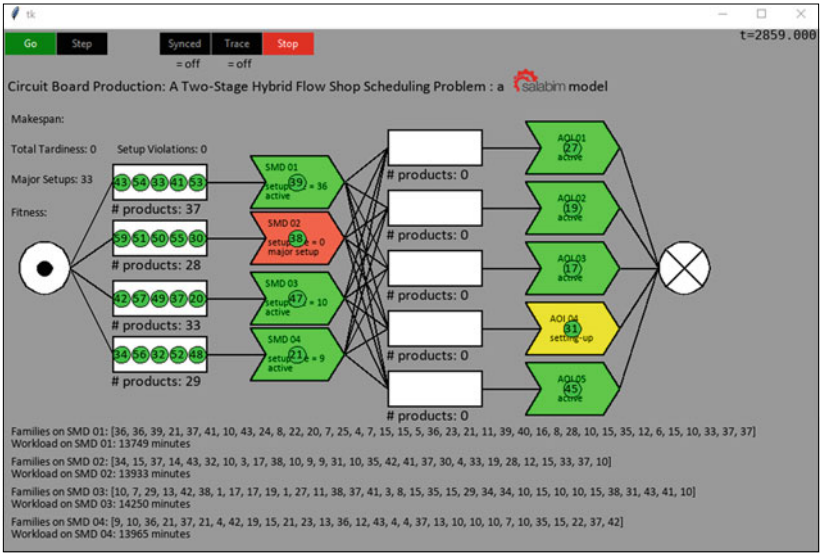


Abbildung 6.10 Animierte DES-Agentenumgebung des zweistufigen HFS-Problems für die NEAT-Implementierung

nur eine SMD-Ressource für dieselbe Familie gerüstet sein darf, über Modellierungsansatz *(ii)* zu implementieren. Wie bereits zu Beginn von Abschnitt 6.3.2 erläutert, besitzt Modellierungsansatz *(ii)* den Nachteil, dass ungültige Agentenentscheidungen verworfen und durch Alternativentscheidungen ersetzt werden, die nicht unmittelbar durch den Agenten beobachtet werden können. Bei Anwendung des A2C-Algorithmus kann durch Modellierungsansatz *(ii)* gewährleistet werden, dass für jede Aktion die (stufenbezogene) Fertigstellung mit einer Belohnung bewertet wird, die mit der Gesamtdauer des Ablaufplans C_{max} und der Gesamtverspätung über alle Aufträge T korreliert. Weil im NEAT-Algorithmus die Belohnung lediglich der Bewertung einer vollständigen Lösung dient, nicht jedoch die Richtung der Optimierung steuert, ist es nicht notwendig, dass jede Aktion durch eine spezifische Belohnung quittiert wird. Stattdessen ist es ausreichend, die Optimierungskriterien am Ende einer Episode für die Berechnung der Belohnung heranzuziehen. Demzufolge wird für die gradientenfreie Lösungsstrategie Modellierungsansatz *(i)* zur Vermeidung von Rüstkonflikten implementiert, bei dem keine Agentenentscheidungen verworfen werden und eine SMD-Ressource solange wartet, bis das angeforderte Rüst-Kit verfügbar ist. Drittens

muss das Simulationsmodell nicht zusätzlich mit der OpenAI-Gym-Schnittstelle ausgestattet werden. Da Belohnungen lediglich am Ende einer Episode vergeben werden, entfällt die Anforderung, dass die Umgebung schrittweise ausführbar sein muss.

Definition der maschinellen Lernaufgaben sowie Zustands- und Aktionsräume der Agenten

Abbildung 6.11 präsentiert vier Strategien, um durch den NEAT-Algorithmus angelernte Agenten für die Produktionsablaufplanung in der ersten Stufe des vorliegenden HFS-Problems einzusetzen. Im Folgenden sollen die Strategien kurz erläutert werden.

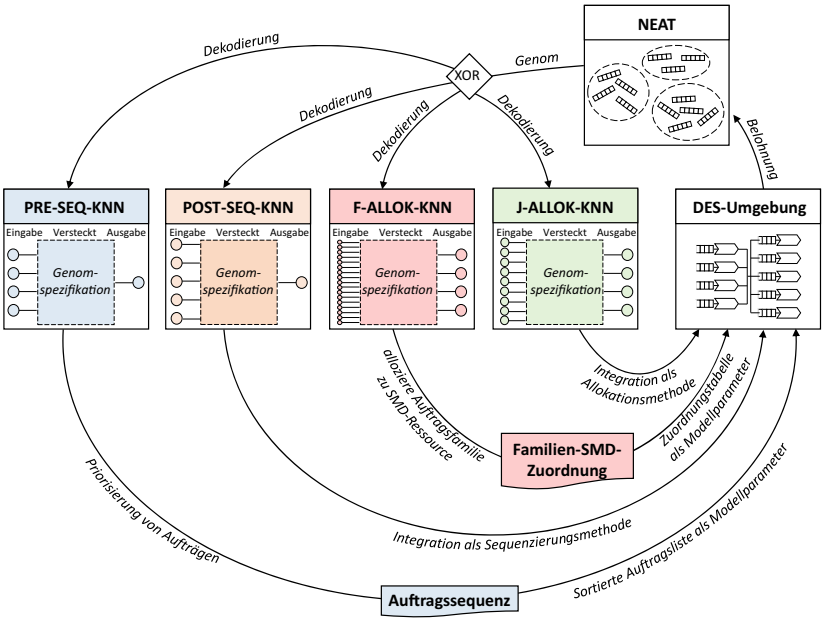


Abbildung 6.11 NEAT-Lösungsstrategien für die Produktionsablaufplanung im HFS-Problem (in Anlehnung an Lang et al 2021b)

PRE-SEQ-KNN und POST-SEQ-KNN beschreiben Strategien zur Sequenzierung von Aufträgen. Beide Strategien basieren auf der zweiten Variante zur Formulierung von Sequenzierungsproblemen als ML-Aufgabe, nach der eine

Auftragssequenz durch Priorisierung einzelner Aufträge iterativ konstruiert wird (Abbildung 5.13 in Abschnitt 5.3.2.2). Abbildung 6.12 veranschaulicht die Zustands- und Aktionsräume der Agenten für beide Strategien.

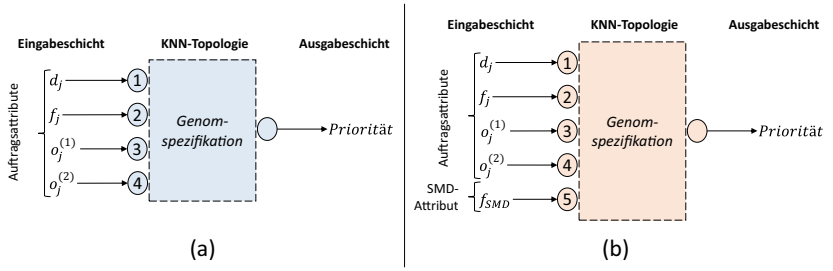


Abbildung 6.12 Zustands- und Aktionsraum der Strategien (a) PRE-SEQ-KNN und (b) POST-SEQ-KNN (in Anlehnung an Lang et al. 2021b)

In der Strategie PRE-SEQ-KNN wird eine Auftragssequenz noch vor der Initialisierung der Simulation konstruiert und an das Simulationsmodell übergeben. Es werden keine Attribute von Produktionsressourcen in der Zustandsbildung berücksichtigt, da diese vor dem Start der Simulation für alle Aufträge gleich wären und somit keine zusätzliche Informationen für die Priorisierung von Aufträgen böten. Somit besteht ein Zustand lediglich aus Auftragsattributen, respektive aus (1) der Fertigstellungsfrist d_j, aus (2) der Familie f_j sowie aus den Bearbeitungszeiten auf der (3) SMD- und (4) AOI-Produktionsstufe $o_j^{(1)}$ und $o_j^{(2)}$ eines Auftrags. Die Sequenz wird darauffolgend nach dem FIFO-Prinzip vom Produktionssystem verarbeitet. Hingegen erfolgt in der Strategie POST-SEQ-KNN die Priorisierung eines Auftrags erst, nachdem dieser einer Produktionsressource zugewiesen wurde. Ergänzend zu den Auftragsattributen der Strategie PRE-SEQ-KNN (1–4) wird die aktuelle Rüstung der zugewiesenen SMD-Ressource f_{SMD} (5) als Systemattribut in die Zustandsbildung miteinbezogen. Bei dieser Strategie wird der Agent als Eingabeparameter an das Simulationsmodell übergeben und in dieses eingebettet, um für Entscheidungen während der Simulation konsultiert zu werden.

Die Strategien F-ALLOK-KNN und J-ALLOK-KNN dienen der Allokation von Aufträgen zu Produktionsressourcen und modellieren das entsprechende Entscheidungsproblem als ML-Aufgabe gemäß Abbildung 5.11 (Abschnitt 5.3.2.1). Abbildung 6.13 zeigt die Zustands- und Aktionsräume der Agenten für beide Strategien. Die erste Allokationsstrategie F-ALLOK-KNN verteilt Auftragsfamilien anstatt

einzelner Aufträge auf SMD-Ressourcen. Alle Allokationsentscheidungen finden vor der Initialisierung der Simulation statt. Für jede beobachtete Familie f_{obs}, die einer SMD-Ressource zugewiesen werden soll, analysiert der Agent einen Zustand bestehend aus 17 Attributen. Es handelt sich hierbei um die Summe der Operationszeiten über alle Aufträge der beobachteten Familie für (1) die SMD- und (2) AOI-Produktionsstufe sowie um (3) die Summe, (4) das Minimum, (5) den Mittelwert, (6) das Maximum, und die (7–9) 0,25-, 0,5- (Median) und 0,75-Quantile der Fertigstellungsfristen über alle Aufträge der beobachteten Familie. Weiterhin analysiert der Agent auf Basis vorangegangener Allokationsentscheidungen, (10–13) welche Arbeitslast und (14–17) wie viele unterschiedliche Familien auf den SMD-Ressourcen zu erwarten sind. Das Ergebnis ist eine Zuordnungstabelle, die Auftragsfamilien auf SMD-Ressourcen abbildet und als

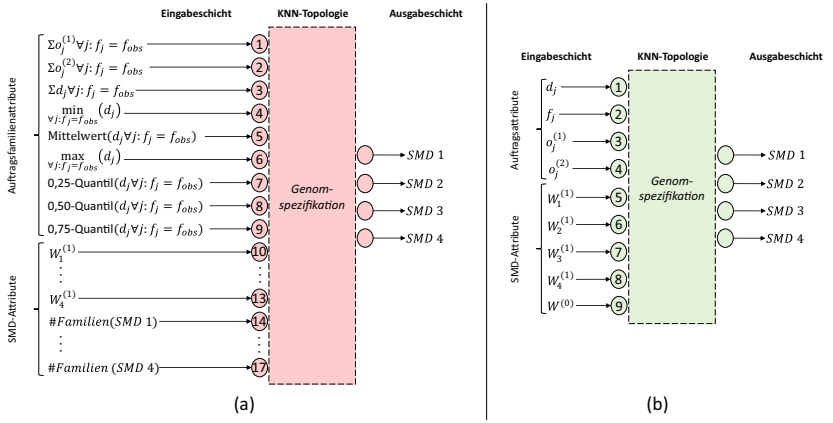


Abbildung 6.13 Zustands- und Aktionsraum der Strategien (a) F-ALLOK-KNN und (b) J-ALLOK-KNN (in Anlehnung an Lang et al. 2021b)

Eingabeparameter des Simulationsmodells dient. Während der Simulation wird für jede Allokationsentscheidung die Familie des Auftrags analysiert und gemäß der Zuordnungstabelle einer SMD-Ressource zugewiesen. Die zweite Allokationsstrategie J-ALLOK-KNN verteilt Aufträge unmittelbar zur Simulationslaufzeit auf die verschiedenen SMD-Ressourcen. Analog zur Sequenzierungsstrategie POST-SEQ-KNN wird der Agent als Eingabeparameter an das Modell übergeben und in dieses integriert. Für jede Allokationsentscheidung analysiert der Agent neun Zustandsinformationen. Die Auftragsattribute (1–4) sind deckungsgleich mit

dem Zustandsraum der Sequenzierungsstrategie PRE-SEQ-KNN. Ferner analysiert der Agent (5–8) die aktuelle Arbeitslast aller SMD-Ressourcen $W^{(1)}_{1,\ldots,4}$ sowie (9) die zum Zeitpunkt noch unverteilte Arbeitslast $W^{(0)}$, gemessen an der SMD-Bearbeitungszeit aller Aufträge, die noch keiner SMD-Ressource zugewiesen wurden.

Experimente und Hyperparameter-Einstellungen für das Agententraining

Die Trainingsexperimente konzentrieren sich auf die Untersuchung verschiedener Kombinationen von Allokations- und Sequenzierungsstrategien. Tabelle 6.13 enthält eine Übersicht zu allen durchgeführten Experimenten. In den Experimenten, in denen eine Prioritätsregel für die Sequenzierung zum Einsatz kommt, wird stets vor der Initialisierung der Simulation eine Auftragsreihenfolge gemäß der gewählten Regel konstruiert (analoges Prinzip zu PRE-SEQ-KNN). Auf eine Untersuchung von Prioritätsregeln für die Sequenzierung von Aufträgen nach der Allokation zu einer SMD-Ressource wird im Rahmen dieser Arbeit verzichtet, da in Voruntersuchungen keine signifikanten Ergebnisunterschiede zwischen der Anwendung von Prioritätsregeln vor und nach der Allokation beobachtet werden konnten. Aufgrund der stochastischen Suche unterliegt das Agententraining mit NEAT statistischen Schwankungen. Aus diesem Grund werden für jedes der gelisteten Experimente zehn Beobachtungen durchgeführt. Während in den Experimenten 01–09 lediglich ein Agent trainiert wird, der entweder für die Allokation oder Sequenzierung von Aufträgen verantwortlich ist, werden in den Experimenten 10–13 stets zwei Agenten trainiert, die das Allokations- und Sequenzierungsproblem kombiniert lösen. Die Gestaltung des Ablaufs für das Training von zwei Agenten erfordert einige konzeptionelle Vorüberlegungen. Aus Implementierungssicht ist das Training von beiden Agenten nacheinander am einfachsten umzusetzen. Hierbei wird bspw. zunächst

der Allokations- und anschließend der Sequenzierungsagent trainiert. In diesem Fall würde der Allokationsagent unter Verwendung einer Prioritätsregel zur Auftragssequenzierung (z. B. FIFO) angelernt werden. Darauffolgend wird für das Training des Sequenzierungsagenten der beste Agent aus dem vorherigen Trainingslauf für die Allokation von Aufträgen verwendet. Der geschilderte Ablauf kann auch umgekehrt durchlaufen werden, indem NEAT zuerst den Agenten für die Sequenzierung und anschließend den Agenten für die Allokation von Aufträgen trainiert. Ein Problem dieser Strategie ist, dass der erste Agent stets in Verbindung mit einem regelbasierten Ansatz und nicht im Verbund mit dem zweiten Agenten trainiert wird. Auf diese Weise können beide Agenten ihre Entscheidungsstrategien nicht aufeinander abstimmen. Alternativ wird für die Experimente 10–13 eine kompetitive, rundenbasierte Ausführung des NEAT-Algorithmus implementiert, wodurch das Training beider Agenten integriert erfolgt.

Tabelle 6.13 Übersicht zu den durchgeführten NEAT-Experimenten (in Anlehnung an Lang et al. 2021b)

Experiment	Sequenzierung	Allokation	Kürzel
01	PRE-SEQ-KNN	Minimale Arbeitslast	PRE-SEQ-KNN&MIN-WL
02	FIFO	J-ALLOK-KNN	FIFO&J-ALLOK-KNN
03	LIFO	J-ALLOK-KNN	LIFO&J-ALLOK-KNN
04	EDD	J-ALLOK-KNN	EDD&J-ALLOK-KNN
05	SPT	J-ALLOK-KNN	SPT&J-ALLOK-KNN
06	FIFO	F-ALLOK-KNN	FIFO&F-ALLOK-KNN
07	LIFO	F-ALLOK-KNN	LIFO&F-ALLOK-KNN
08	EDD	F-ALLOK-KNN	EDD&F-ALLOK-KNN
09	SPT	F-ALLOK-KNN	SPT&F-ALLOK-KNN
10	PRE-SEQ-KNN	J-ALLOK-KNN	PRE-SEQ-KNN&J-ALLOK-KNN
11	POST-SEQ-KNN	J-ALLOK-KNN	POST-SEQ-KNN&J-ALLOK-KNN
12	PRE-SEQ-KNN	F-ALLOK-KNN	PRE-SEQ-KNN&F-ALLOK-KNN
13	POST-SEQ-KNN	F-ALLOK-KNN	POST-SEQ-KNN&F-ALLOK-KNN

Abbildung 6.14 zeigt das Flussdiagramm des kompetitiven, rundenbasierten Trainingsprozesses. Hierbei repräsentieren rot-grüne Blöcke den Trainingsprozess des Allokationsagenten und blau-orange Blöcke den Trainingsprozess des Sequenzierungsagenten. Der Trainingsprozess startet mit der Suche nach einer initialen KNN-Repräsentation für den Allokationsagenten, unter Verwendung der FIFO-Prioritätsregel für die Sequenzierung von Aufträgen. Diese ist gewöhnlich nach einer Generation gefunden, da zu Beginn die globale beste Belohnung mit einem hohen negativen Wert initialisiert wird, welche von den meisten Genomen der initialen Population bereits übertroffen wird. Die letzte Generation der Population sowie deren bestes Genom werden gespeichert. Die Fitness des besten Genoms wird fortan als neue globale beste Belohnung verwendet. Im Folgenden versucht der NEAT-Algorithmus für das Sequenzierungsproblem, eine KNN-Repräsentation zu finden, welche eine bessere Fitness erzielt als die neue globale beste Belohnung. Bei der Bewertung eines Sequenzierungsgenoms wird das beste Genomdes zuvor ausgeführten NEAT-Algorithmus für die Auftragsallokation verwendet und umgekehrt. Sobald der NEAT-Algorithmus für die Auftragssequenzierung ein Genom findet, das die globale beste Belohnung überschreitet, speichert der Algorithmus die aktuelle Population sowie das Genom

mit der besten Fitness, bevor das Verfahren terminiert. Die Fitness des besten Genoms wird als neue globale Belohnung gesetzt. Der NEAT-Algorithmus für das Allokationsproblem evolviert nun die letzte Generation des vorherigen Allokationsexperiments, anstatt eine neue Population von Grund auf zu erstellen. Dieser Prozess wird so lange wiederholt, bis beide Algorithmen die globale beste Belohnung innerhalb einer benutzerdefinierten Anzahl von Generationen nicht mehr verbessern können.

In allen Experimenten werden die Datensätze 2 und 3 für das Training sowie die Datensätze 1 und 4 für die Evaluation der trainierten Agenten verwendet. In Vorexperimenten hat sich herausgestellt, dass der Agent nicht in der Lage ist, seine Entscheidungspolitik auf andere Probleminstanzen zu generalisieren, sofern nur ein Datensatz für das Agententraining verwendet wird. Die Entscheidung zur Verwendung der Datensätze 2 und 3 für das Agententraining basiert auf einer ausführlichen statistischen Analyse in (Lang et al. 2021b). In dieser wurde für alle Datensätze die Anzahl von Ausreißern für jedes Auftragsattribut über die Kurtosis K und für den jeweils gesamten Datensatz über die gemittelte normalisierte Kurtosis $\overline{K}$ quantifiziert. Die Kurtosis beschreibt die Wölbung einer zufallsverteilten Größe. Eine geringe Kurtosis weist darauf hin, dass die Werte einer Datenreihe eher gleichmäßig verteilt sind, während eine hohe Kurtosis auf eine größere, unregelmäßigere Streuung der Daten und damit auf ein höheres Maß an Ausreißern hinweist. Tabelle 6.8 (Abschnitt 6.3.1) enthält die Kurtosis K für jedes Auftragsattribut sowie die gemittelte normalisierte Kurtosis $\overline{K}$ für jeden Datensatz. Die Datensätze 2 und 3 weisen die geringste gemittelte normalisierte Kurtosis auf und somit die geringste Anzahl an statistischen Ausreißern. Es ist somit anzunehmen, dass der Agent auf den Datensätzen 2 und 3 einfacher zu einer Entscheidungspolitik konvergiert. Aus diesem Grund werden die Datensätze 2 und 3 als Trainingsdatensätze gewählt.

Als Belohnungsfunktion wird die negative Gesamtverspätung über alle Aufträge $(-1) * T$ verwendet. In jeder Generation wird jedes Genom zuerst anhand von Datensatz 2 und danach anhand von Datensatz 3 bewertet. Die Belohnung eines Genoms entspricht der negativen Summe der Gesamtverspätung über alle Aufträge und Datensätze. Die Gesamtdauer des Ablaufplans C_{max} wird lediglich als sekundäres Optimierungskriterium berücksichtigt. Sofern mehrere Genome die niedrigste Gesamtverspätung über alle Aufträge aufweisen, wird von diesen Genomen dasjenige gewählt, bei dem die geringste Gesamtdauer des Ablaufplans resultiert.

Für das Agententraining werden zwei unterschiedliche Hyperparameter-Konfigurationen definiert. Zunächst werden die Experimente 01–13 mit den Hyperparameter-Einstellungen durchgeführt, die in Tabelle 6.14 gelistet sind. Die Hyperparameter-Einstellungen basieren auf dem Anwendungsbeispiel »OpenAI Lander«, das Bestandteil der Softwarebibliothek »neat-python« ist (McIntyre et al. 2017a). Im OpenAI-Lander-Beispiel muss ein Agent eine Mondlandefähre so steuern, dass diese in einem bestimmten Bereich auf dem Mond

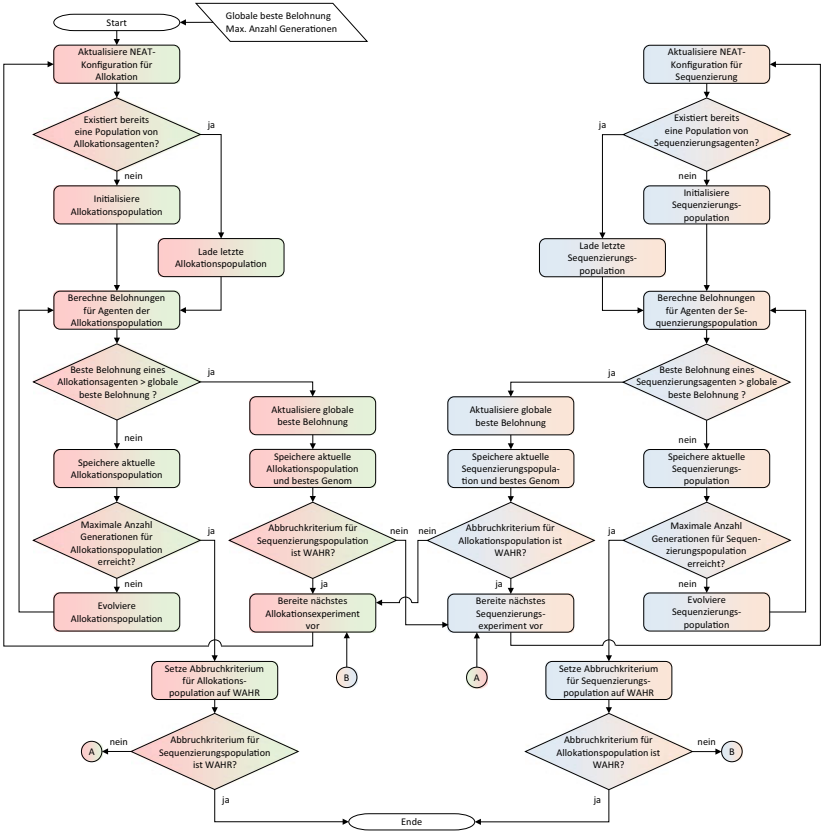


Abbildung 6.14 Algorithmischer Ablauf des kompetitiven, rundenbasierten NEAT-Trainingsprozesses (in Anlehnung an Lang et al. 2021b)

landet. Im Vergleich zu den anderen Beispielen erscheinen die Hyperparameter-Einstellungen des OpenAI-Lander-Beispiels am geeignetsten, da die Eingabe- und Ausgabeschicht des Agenten zur Steuerung der Mondlandefähre (acht Eingänge bzw. vier Ausgänge) den Eingabe- und Ausgabeschichten der Agenten für die Allokation von Aufträgen am meisten ähneln. Aufgrund der Tatsache, dass die beiden Strategien F-ALLOK-KNN und J-ALLOK-KNN die größte Anzahl von Eingabe- und Ausgabeneuronen besitzen, ist anzunehmen, dass die Identifikation einer geeigneten KNN-Architektur für beide Allokationsstrategien

anspruchsvoller ist als für beide Sequenzierungsstrategien. Aus diesem Grund wurde eine Referenzkonfiguration für die Hyperparameter gesucht, die insbesondere für das Training der Allokationsagenten geeignet erscheint. Es wurden wenige Adaptionen an der Hyperparameter-Konfiguration des OpenAI-Lander-Beispiels vorgenommen, die im Folgenden diskutiert werden sollen. In den Experimenten 01–09 wird die Population jeweils über 500 Generationen evolviert. In den Experimenten 10–13, in denen zwei NEAT-Algorithmen rundenbasiert gegeneinander antreten, muss jede NEAT-Instanz in jeder Runde innerhalb von maximal 50 Generationen eine bessere Belohnung erzielen als die zum Zeitpunkt global beste Belohnung. Die Populationsgröße wird von 150 auf 250 erhöht, um eine höhere Anzahl verschiedener Lösungskandidaten zu untersuchen, weil die Genome der Allokationsstrategien eine höhere Anzahl von Eingabe- und Ausgabeneuronen besitzen als die Genome des OpenAI-Lander-Beispiels. Ferner wird lediglich die Sigmoid-Funktion als mögliche Aktivierungsfunktion erlaubt, sodass alle Neuronen Ausgaben zwischen null und eins produzieren. Die Festlegung auf lediglich eine Aktivierungsfunktion dient als Maßnahme, um das Risiko einer Überanpassung auf die Trainingsdaten zu reduzieren. Mithilfe der zweiten Hyperparameter-Konfiguration soll erforscht werden, welche Lösungsqualität mithilfe des kompetitiven, rundenbasierten NEAT-Trainingsprozesses möglich ist, sofern entweder die Rechenzeit für das Agententraining vernachlässigbar ist oder u. U. die Hyperparameter des Agententrainings experimentell optimiert werden. Zu diesem Zweck werden einige Hyperparameter verändert, um eine größere Exploration des Lösungsraums zu Lasten der algorithmischen Laufzeit zu erlauben. So werden die Mutationswahrscheinlichkeiten von Neuronen und Synapsen verdoppelt. Ebenso werden die Wertebereiche und Mutationsraten von synaptischen Gewichten sowie von neuronalen Bias-Termen und Reaktivitätskoeffizienten verdoppelt. Ferner wird die Größe von jeder Population auf 1000 Genome erhöht. Schließlich wird die Anzahl der Generationen, in welcher eine NEAT-Instanz eine bessere Lösung als die globale beste Belohnung ermitteln muss, von 50 auf 100 erhöht.

Trainingsergebnisse und Evaluation

Wie in Abbildung 6.15 ersichtlich, variiert die Lösungsgüte der trainierten Agenten zwischen den Experimenten mitunter deutlich und unterliegt teilweise hohen Schwankungen. Abbildung 6.15 zeigt die resultierende Gesamtverspätung über alle Aufträge T (oben) bzw. die resultierenden Gesamtzeiten der Ablaufpläne C_{max} (unten) der trainierten Agenten als Boxplot-Diagramme. In beiden Boxplot-Diagrammen sind die Experimente nicht nach aufsteigenden Indizes, sondern jeweils aufsteigend nach T bzw. C_{max} sortiert. Die vollständigen Ergebnisdaten sind als Anhang J dem elektronischen Zusatzmaterial beigefügt. Bezogen auf das primäre Belohnungskriterium T ist zu erkennen, dass der Agent in allen Experimenten in der Lage ist, auf den Testdatensätzen ähnliche Ergebnisse zu erzielen

wie auf den Trainingsdatensätzen. Lediglich in einigen Allokationsexperimenten, in welchen eine statische Prioritätsregel für die Auftragssequenzierung zum Einsatz kommt, leistet der trainierte Agent auf dem vierten Datensatz erkennbar andere Ergebnisse als auf den verbleibenden Datensätzen. Ferner ist ersichtlich, dass das Training von Allokationsagenten in Kombination mit den meisten statischen Prioritätsregeln zu wesentlich schlechteren Ergebnissen führt, verglichen zu denjenigen Experimenten, in denen ein Sequenzierungsagent ausschließlich oder in Kombination mit einem Allokationsagenten trainiert wird. Eine nachvollziehbare Ausnahme bildet die EDD-Prioritätsregel, welche für die Minimierung der Gesamtdauer des Ablaufplans prädestiniert ist. Hinsichtlich des sekundären Belohnungskriteriums C_{max} pendeln sich die meisten der untersuchten Strategien in einer Zeitspanne zwischen 20.000 und 30.000 Minuten ein. Es ist zu beobachten, dass in einigen Replikationen Ablaufpläne mit einer Gesamtdauer

Tabelle 6.14 Hyperparameter-Einstellungen für NEAT (in Anlehnung an Lang et al. 2021b)

Hyperparameter	**Wert**
Allgemeine Parameter	
Anzahl Generationen	500 (EXP 01–09) bzw. 50 (EXP 10–13)
Populationsgröße	250
Fitnesskriterium der Population	Minimum
Fitnessschwellenwert	0,0
Merker für das Zurücksetzen bei aussterbender Population	FALSCH
Genom-Parameter	
Merker für rekurrente synaptische Verbindungen	FALSCH
Merker für einzelne strukturelle Mutationen	FALSCH
Merker zur Absicherung struktureller Mutationen	FALSCH
Initiale Aktivierungsfunktion	Sigmoid
Optionen für Aktivierungsfunktion	[Sigmoid]
Mutationswahrscheinlichkeit für Aktivierungsfunktion	0,0
Initiale Netzeingabefunktion	Summe

(Fortsetzung)

Tabelle 6.14 (Fortsetzung)

Hyperparameter	**Wert**
Optionen für Netzeingabefunktion	[Summe]
Mutationsrate für Netzeingabefunktion	0,0
Wahrscheinlichkeit für die Erstellung eines neuen Neurons	0,15
Wahrscheinlichkeit für die Eliminierung eines Neurons	0,1
Initialer neuronaler Bias-Term	Normal($\mu = 0{,}0$; $\sigma = 1{,}0$)
Mutationswahrscheinlichkeit für neuronalen Bias-Term	0,8
Mutationsrate für neuronalen Bias-Term	Normal($\mu = 0{,}0$; $\sigma = 0{,}4$)
Ersatzwahrscheinlichkeit für neuronalen Bias-Term	0,02
Minimaler neuronaler Bias-Term	$-30{,}0$
Maximaler neuronaler Bias-Term	30,0
Initiale neuronale Reaktivität	Normal($\mu = 1{,}0$; $\sigma = 0{,}0$)
Mutationswahrscheinlichkeit für neuronale Reaktivität	0,1
Mutationsrate für neuronale Reaktivität	Normal($\mu = 0{,}0$; $\sigma = 0{,}01$)
Ersatzwahrscheinlichkeit für neuronale Reaktivität	0,0
Minimale neuronale Reaktivität	$-30{,}0$
Maximale neuronale Reaktivität	30,0
Initiale Verbindung von Neuronen	*„partial 0,5“*
Wahrscheinlichkeit für die Erstellung einer neuen Synapse	0,15
Wahrscheinlichkeit für die Eliminierung einer Synapse	0,1
Initialer Status einer Synapse	WAHR
Mutationswahrscheinlichkeit eines synaptischen Status	0,05

(Fortsetzung)

Tabelle 6.14 (Fortsetzung)

Hyperparameter	**Wert**
Initiales synaptisches Gewicht	Normal($\mu = 0{,}0$; $\sigma = 1{,}0$)
Mutationswahrscheinlichkeit für synaptische Gewichte	0,8
Mutationsrate für synaptische Gewichte	Normal($\mu = 0{,}0$; $\sigma = 0{,}4$)
Ersatzwahrscheinlichkeit für synaptische Gewichte	0,02
Minimales synaptisches Gewicht	$-30{,}0$
Maximales synaptisches Gewicht	30,0
Spezies-Parameter	
Fitnesskriterium von Spezies	Mittelwert
Maximale Anzahl stagnierender Generationen	15
Anzahl elitärer Spezies	4
Anzahl elitärer Genome pro Spezies	2
Anteil reproduktionsfähiger Genome pro Spezies	0,2
Mindestanzahl Genome pro Spezies nach Reproduktion	3,0
Kompatibilitätskoeffizient für identische Gene	1,0
Kompatibilitätskoeffizient für verschiedene Gene	1,0

von weit unter 20.000 Minuten erzielt werden. Im Allgemeinen sind die statistischen Schwankungen von C_{max} wesentlich stärker ausgeprägt als von T, da C_{max} lediglich als sekundäres Belohnungskriterium bei der Evaluation von Genomen miteinbezogen wird.

Unter denjenigen Experimenten, in welchen sowohl ein Allokations- als auch ein Sequenzierungsagent angelernt werden, weist Experiment 12 (PRE-SEQ-KNN&F-ALLOK-KNN) die besten Ergebnisse für T und C_{max} auf. Vor diesem Hintergrund wird Experiment 12 unter Verwendung der zweiten Hyperparameter-Konfiguration wiederholt, um zu überprüfen, ob eine höhere Lösungsgüte erreicht werden kann, wenn ein größerer Lösungsraum, unter Inanspruchnahme von mehr Rechenzeit und -ressourcen, durchsucht wird. Die Ergebnisse der zehn Replikationen sind in Tabelle J-4 in Anhang J im elektronischen Zusatzmaterial

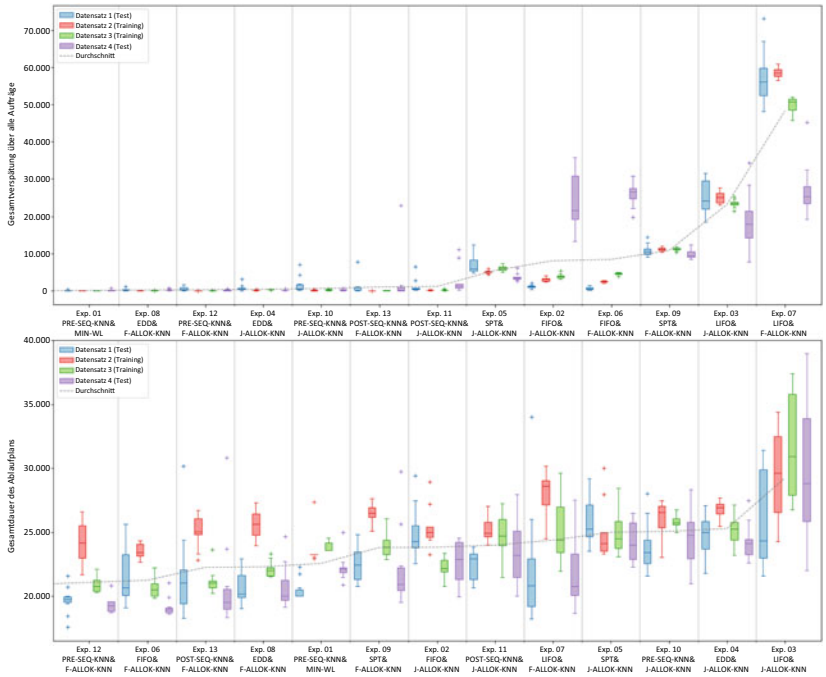


Abbildung 6.15 Boxplot-Diagramme für die Gesamtverspätung über alle Aufträge und für die Gesamtdauer der Ablaufpläne für die Experimente 01–13 (in Anlehnung an Lang et al. 2021b)

dargestellt. Durch die Vergrößerung des durchsuchbaren Lösungsraums können erstmals ein Allokations- und Sequenzierungsagent ermittelt werden, die sowohl auf den Trainings- als auch auf den Testdatensätzen Produktionspläne ohne verspätete Aufträge erzeugen. Zusammengefasst werden in Tabelle 6.15 die Ergebnisse der besten beiden NEAT-Experimente dargestellt. Neben der kombinierten Allokations- und Sequenzierungsstrategie PRE-SEQ-KNN&F-ALLOK-KNN leistet Experiment 1, in welchem nur ein Agent für die Sequenzierung angelernt wird, die besten Ergebnisse.

6.3.4 Vergleich mit anderen Lösungsverfahren

Abschließend zu dieser Fallstudie sollen die Lösungsgüte des A2C- und des NEAT-Algorithmus mit anderen Lösungsverfahren verglichen werden. In der

Tabelle 6.15 Lösungsgüte der besten NEAT-trainierten Agenten auf allen vier Datensätzen des HFS-Problems

	PRE-SEQ-KNN&MIN-WL (Experiment 1)		PRE-SEQ-KNN&F-ALLOK-KNN (Experiment 12)	
	C_{max}	T	C_{max}	T
Datensatz 1	20.663	0	18.557	0
Datensatz 2	22.938	0	24.467	0
Datensatz 3	24.272	0	21.878	0
Datensatz 4	20.886	51	19.115	0

Vergangenheit untersuchten Aurich et al. (2016) und Nahhas et al. (2016) die Metaheuristiken »Simulated Annealing« (SA), »Tabu Search« (TS) sowie einen genetischen Algorithmus (GA) für die Auftragsreihenfolge- und Ressourcenbelegungsplanung im vorliegenden HFS-Problem. Ferner untersuchten die Autoren eine eigens entwickelte und für das HFS-Problem spezifizierte Heuristik, die unter dem Namen »Integrated Simulation-Based Optimization« (ISBO) veröffentlicht wurde. Sowohl die Metaheuristiken als auch die ISBO-Heuristik lösen das Allokationsproblem auf der SMD-Stufe, ohne die daran anschließende AOI-Stufe zu betrachten. Ebenfalls sollen der A2C- und der NEAT-Algorithmus mit der EDD- und SPT-Regel verglichen werden. Beide Sequenzierungsregeln werden im Verbund mit der Allokationsregel »Minimale Arbeitslast« (MIN-WL) untersucht. Darüber hinaus werden die Ergebnisse der Fertigungsplanung (FP) des Unternehmens in den Vergleich miteinbezogen. Die Einlastung von Aufträgen erfolgt derzeit manuell durch die verantwortlichen Stellen in der Fertigungsplanung, wobei der Fokus auf der Vermeidung von großen Umrüstprozessen liegt.

Abbildung 6.16 bewertet alle genannten Lösungsverfahren anhand der Gesamtdauer des Ablaufplans C_{max} und anhand der Gesamtverspätung über alle Aufträge T. Die Ergebnisdaten in Abbildung 6.16 sind ebenfalls in Anhang J im elektronischen Zusatzmaterial tabellarisch hinterlegt (Tabelle J-5). Sowohl der A2C-Algorithmus als auch die besten beiden NEAT-Lösungsstrategien produzieren für beide Optimierungskriterien stets bessere Ergebnisse als die derzeitige Fertigungsplanung des Unternehmens und die auf statischen Planungsregeln basierenden Lösungsansätze. Hinsichtlich des

Optimierungskriteriums C_{max} wird der A2C-Algorithmus im Mittel nur durch den genetischen Algorithmus geschlagen. Der NEAT-Algorithmus liefert in Bezug auf C_{max} in etwa die gleiche Lösungsgüte wie die Metaheuristiken »Simulated Annealing« und »Tabu Search«. Die schlechtere Performanz

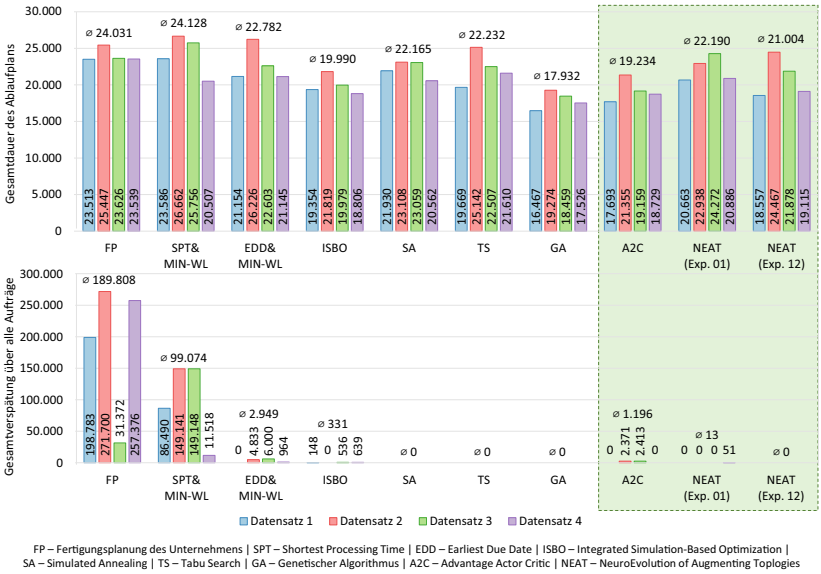


Abbildung 6.16 Vergleich des A2C- und des NEAT-Algorithmus mit anderen Lösungsverfahren

des NEAT-Algorithmus ist insofern nicht ungewöhnlich, da die Gesamtdauer des Ablaufplans in dessen Belohnungsfunktion nur ein sekundäres Auswahlkriterium darstellt. Hinsichtlich des Optimierungskriteriums T generieren beide NEAT-Strategien im Mittel bessere Ergebnisse als der A2C-Algorithmus und die ISBO-Heuristik. Die schlechtere Performanz des A2C-Algorithmus ist ebenfalls nicht ungewöhnlich, da dessen Entscheidungspolitik unter einer multikriteriellen Belohnungsfunktion angelernt wurde, die C_{max} und T gleichermaßen gewichtet. Lediglich die NEAT-Strategie, in der ein Allokations- und Sequenzierungsagent im Verbund angelernt werden, ist in der Lage, Ablaufpläne zu generieren, in denen keine Fertigstellungsfristen verletzt werden. Diesbezüglich erreicht NEAT dieselbe Lösungsgüte wie die drei verglichenen Metaheuristiken. Hinsichtlich des Berechnungsaufwands für die Generierung einer Lösung sind deutliche Unterschiede zwischen den heuristischen und agentenbasierten Verfahren auf der einen Seite und den Metaheuristiken auf der anderen Seite zu erkennen.

Tabelle 6.16 vergleicht den Berechnungsaufwand für die Generierung der präsentierten Lösungen anhand der Anzahl der benötigten Simulationsläufe. Mit Ausnahme der NEAT-Strategie PRE-SEQ-KNN&F-ALLOK-KNN ist zu beobachten, dass das Agententraining mithilfe des A2C- und NEAT-Algorithmus in

etwa denselben Berechnungsaufwand erfordert wie die Anwendung der Metaheuristiken »Simulated Annealing«, »Tabu Search« und »Genetischer Algorithmus« zur Generierung von lediglich einer lokalen besten Lösung für einen Datensatz. Die NEAT-Strategie PRE-SEQ-KNN&F-ALLOK-KNN birgt mutmaßlich noch hohe Verbesserungspotenziale zur Reduzierung des Berechnungsaufwands. Mögliche Maßnahmen sind z. B. die Verwendung von bereits vortrainierten KNN als Startlösung, die Initialisierung der globalen besten Belohnung mit besseren Werten oder die Einführung einer Restriktion, dass jedes NEAT-Verfahren die globale Belohnung nicht nur schlagen, sondern um einen benutzerdefinierten absoluten oder relativen Wert überbieten muss.

Tabelle 6.16 Berechnungsaufwand der Lösungsverfahren gemessen an der Anzahl der benötigten Simulationsläufe

Lösungsverfahren	**Anzahl von Simulationsläufen**
SPT&MIN-WL	1
EDD&MIN-WL	1
ISBO	1
Simulated Annealing	1.500
Tabu Search	3.150
Genetischer Algorithmus	6.000–8.000
Training	
A2C	2.500
NEAT (EXP 01)	750–3.500
NEAT (EXP 12)	220.000–360.000
Einsatz der angelernten Agenten	
A2C	1
NEAT (EXP 01)	1
NEAT (EXP 12)	1

Im Gegensatz zu den metaheuristischen Ansätzen und analog zu den heuristischen Lösungsstrategien sind die Agenten nach dem Training in der Lage innerhalb eines Simulationslaufs einen vollständigen Produktionsablaufplan für eine beliebige Anzahl von Aufträgen zu generieren. Darüber hinaus eignen sich die heuristischen und agentenbasierten Lösungsstrategien für eine echtzeitfähige Produktionssteuerung, da auftragsweise Allokations- und Sequenzierungsentscheidungen zur Laufzeit getroffen werden.

Obgleich die ISBO-Heuristik und die trainierten Agenten gleichermaßen nur ein Simulationslauf für die Generierung einer vollständigen Lösung benötigen,

weist die ISBO-Heuristik eine höhere Zeitkomplexität auf, was im Folgenden anhand der Anzahl benötigter Rechenschritte aufgezeigt werden soll. Bei der ISBO-Heuristik ist die Anzahl der Rechenschritte von der Anzahl der auftretenden Umplanungsereignisse abhängig. Immer dann, wenn ein Auftrag die SMD-Stufe abschließt, überprüft die ISBO-Heuristik für diejenige Familie, die den Auftrag mit der geringsten Fertigstellungsfrist enthält und die zum Zeitpunkt auf keiner SMD-Ressource gerüstet ist, ob die fristgerechte Bearbeitung des Auftrags mit der geringsten Fertigstellungsfrist gefährdet ist. Zu diesem Zweck sortiert die Heuristik alle Auftragsfamilien in aufsteigender Reihenfolge der Fertigstellungsfristen aller nicht produzierten Aufträge. Danach weist die ISBO-Heuristik die erste Familie der sortierten Menge der verfügbaren SMD-Ressource zu, sofern ansonsten der Auftrag mit der geringsten Fertigstellungsfrist innerhalb dieser Familie nicht fristgerecht bearbeitet werden könnte. Andernfalls wird die Bearbeitung der Aufträge der aktuell gerüsteten Familie fortgesetzt.

Um die Zeitkomplexität der ISBO-Heuristik für den ungünstigsten Fall zu approximieren, seien $|J|$ Aufträge angenommen, wobei alle Aufträge unterschiedlichen Familien angehören ($|J| = |Fam|$). Die Zeitkomplexität für die Sortierung aller Familien nach aufsteigenden Fertigstellungsfristen hängt maßgeblich vom verwendeten algorithmischen Verfahren ab. Nachfolgend sei angenommen, dass für die Sortierung der Auftragsfamilien der Bubblesort-Algorithmus angewandt wird. Ferner sei angenommen, dass $|Fam| = n$ Auftragsfamilien zu sortieren sind. Im schlechtesten Fall benötigt der Algorithmus $\left(n^2 - n\right)/2$ Vergleichsoperationen bzw. $\left(n^2 - n\right) * 1,5$ Austauschoperationen für die vollständige Sortierung der Auftragsliste (Wirth 1983, S. 105). In beiden Termen repräsentiert n^2 die höchste Potenz, sodass dem Bubblesort-Algorithmus eine Zeitkomplexität von $O\left(n^2\right)$ zugeschrieben werden kann. Insgesamt finden $|J|$ Sortiervorgänge während eines Simulationslaufs statt. In Bezug auf die ISBO-Heuristik muss berücksichtigt werden, dass die Anzahl der zu sortierenden Aufträge mit der Anzahl der Umplanungsentscheidungen abnimmt, weil immer nur dann ein Umplanungsereignis stattfindet, sobald ein Auftrag die SMD-Bearbeitungsstufe abgeschlossen hat. Daher variiert die Anzahl der zu sortierenden Aufträge zwischen 1 und $|J|$. Für dieses Beispiel sei angenommen, dass n die Laufvariable für Aufträge repräsentiert. Folglich finden im ungünstigsten Fall $\sum_{n=1}^{|J|}\left(n^2 - n\right)/2$ Vergleichsoperationen bzw. $\sum_{n=1}^{|J|}\left(n^2 - n\right) * 1,5$ Austauschoperationen für die Berechnung einer Lösung statt. Auch in diesem Fall repräsentiert n^2 die höchste Potenz in beiden Termen, sodass ebenfalls der ISBO-Heuristik eine Laufzeitkomplexität von $O\left(n^2\right)$ zugeschrieben werden kann. Somit wächst im schlechtesten

Fall der Berechnungsaufwand mindestens quadratisch in Abhängigkeit der Anzahl einzulastender Aufträge.

Bei der agentenbasierten Produktionsablaufplanung hängt die Anzahl der Berechnungsschritte von der Anzahl der einzulastenden Aufträge und der Anzahl der Matrixmultiplikationen für die Vorwärtspropagierung der Zustände ab. Die Anzahl der Matrixmultiplikationen ist jedoch für ein und dasselbe KNN stets konstant und daher für die Berechnung der Zeitkomplexität vernachlässigbar. Aus diesem Grund kann die Zeitkomplexität eines jeden KNN für die Berechnung einer vollständigen Lösung auf $O(n)$ reduziert werden, wobei $n = |J|$ gilt. Der Berechnungsaufwand steigt somit lediglich linear zur Anzahl der einzulastenden Aufträge. Die unterschiedlichen Komplexitäten können anhand der Rechenzeit für ein Simulationsexperiment bemessen werden. So benötigt ein A2C- bzw. NEAT-trainierter Agent weniger als eine Sekunde, um einen vollständigen Produktionsablaufplan zu erstellen und zu evaluieren. Hingegen benötigt die ISBO-Heuristik für dieselbe Aufgabe 8 bis 18 Sekunden. Die Differenz kann jedoch nicht ausschließlich auf die unterschiedlichen Zeitkomplexitäten zurückgeführt werden. Ein weiterer Einflussfaktor ist die Verwendung von unterschiedlichen Werkzeugen für die Implementierung der Simulationsmodelle.

Zusammengefasst liefern der A2C- und der NEAT-Algorithmus bessere Ergebnisse als die derzeitige Fertigungsplanungsstrategie des Unternehmens und als die Verwendung von regelbasierten Ansätzen. Zudem ist die Lösungsgüte beider ML-Verfahren mit der selbstentwickelten und auf das Problem zugeschnittenen ISBO-Heuristik sowie mit den metaheuristischen Verfahren vergleichbar. Mitunter generieren der A2C- und der NEAT-Algorithmus sogar bessere Ergebnisse bei einem geringeren Berechnungsaufwand als die ISBO-Heuristik und die metaheuristischen Verfahren. Hinsichtlich des Berechnungsaufwands sind die trainierten Agenten mit den regelbasierten Ansätzen vergleichbar.

Im Vergleich zum NEAT-Algorithmus scheint die durch den A2C-Algorithmus erlernte Entscheidungspolitik besser auf unbekannte Daten generalisieren zu können. Der A2C-Algorithmus wurde lediglich auf Datensatz 1 angelernt, wohingegen der NEAT-Algorithmus mindestens zwei Datensätze beobachten musste, um ebenfalls gute Ergebnisse auf anderen Datensätzen zu erzielen.

Schlussbetrachtung

7

Abschließend soll ein Resümee der vorliegenden Forschungsarbeit gezogen werden. Hierbei ist Abschnitt 7.1 der inhaltlichen Zusammenfassung der Arbeit sowie der Beantwortung der in Abschnitt 1.2 formulierten Forschungsfragen gewidmet. In Abschnitt 7.2 wird ein Ausblick auf weitere Forschungsbedarfe und zukünftige Forschungsarbeiten gegeben.

7.1 Zusammenfassung und Diskussion

Im Rahmen dieser Forschungsarbeit wurde eine Methode entwickelt, die ein Vorgehen zur Adaption, Integration und Anwendung von RL-Verfahren für die Produktionsablaufplanung beschreibt. Vor dem Hintergrund aktueller Herausforderungen und Tendenzen im Bereich der Produktion und Logistik wurde die Entwicklung der Methode durch den Umstand motiviert, dass derzeitige Lösungsverfahren der Produktionsablaufplanung keinen Kompromiss zwischen Lösungsgüte, Rechenzeit und Implementierungsaufwand erzielen. Die Entwicklung der Methode erfolgte in Anlehnung an das »Information Systems Research Framework« von Hevner et al. (2004), bei welchem es sich um eine gestaltungsorientierte Forschungsmethodik handelt.

Im Zuge des Grundlagenkapitels zur Produktionsablaufplanung wurde der Untersuchungsbereich der Arbeit konkretisiert. Haupterkenntnis von Kapitel 2 war, dass sich die Produktionsablaufplanung auf die drei mathematischen Optimierungsprobleme »Ressourcenbelegungsplanung«, »Reihenfolgeplanung« und »Losbildung« reduzieren lässt.

Im Grundlagenkapitel zu RL-Verfahren wurden deren Charakteristiken sowie Unterschiede zu angrenzenden ML-Paradigmen herausgearbeitet. Die zentrale

S. Lang, *Methoden des bestärkenden Lernens für die Produktionsablaufplanung*,
https://doi.org/10.1007/978-3-658-41751-2_7

Erkenntnis aus Kapitel 3 war, dass RL-Verfahren grundsätzlich in gradientenabhängige und gradientenfreie Verfahren unterteilt werden können. Beide Verfahrensarten unterscheiden sich in der Interaktion zwischen Agent und Umgebung, insbesondere hinsichtlich der Art und des Zeitpunkts der Belohnungsvergabe. Anschließend wurde in Kapitel 4 eine Literaturrecherche zur Anwendung von RL-Verfahren für die Produktionsablaufplanung präsentiert. Die Literaturrecherche diente insbesondere der Konkretisierung der Forschungslücke und partiell der Beantwortung der ersten Forschungsfrage.

Das fünfte Kapitel widmete sich der Darlegung der Methode zum Einsatz von RL-Verfahren für die Produktionsablaufplanung. Diesbezüglich wurden zunächst in Abschnitt 5.1 die Grenzen des konventionellen stufenweisen Vorgehens für die Produktionsablaufplanung aufgezeigt und hierauf aufbauend acht Anforderungen an die zu entwickelnde Methode postuliert. Die Methode adressiert zwei Aspekte: *(i)* Die Integration und den Einsatz von RL-Verfahren in bestehende Produktionsabläufe sowie *(ii)* den Entwurf und die Implementierung von RL-basierten Entscheidungssystemen für die Produktionsablaufplanung. Bezugnehmend auf Aspekt *(i)* wurden in Abschnitt 5.2 die Grundzüge der agentenbasierten Produktionsablaufsteuerung hergeleitet, auf deren Basis RL-trainierte Agentenmodelle sowohl für Ablaufentscheidungen im Produktionsbetrieb als auch für die Vorberechnung von vollständigen Produktionsablaufplänen eingebunden werden können. Hinsichtlich Aspekt *(ii)* wurde in Abschnitt 5.3 ein siebenstufiges Vorgehensmodell vorgestellt, das zur Projektierung und Entwicklung von agentenbasierten Produktionsablaufsteuerungen anleitet. Abschließend wurde in Abschnitt 5.4 dargelegt, inwiefern die entwickelte Methode die in Abschnitt 5.1 postulierten Anforderungen erfüllt.

In Kapitel 6 wurden drei Fallstudien präsentiert, in denen unterschiedliche RL-agentenbasierte Lösungsstrategien für verschiedene Probleme der Produktionsablaufplanung untersucht wurden. Die Fallstudien dienten zum einen dem grundsätzlichen Nachweis der in Abschnitt 1.1 formulierten Forschungshypothese, dass RL-Methoden einen Ausgleich zwischen Lösungsgüte, Rechenzeit und Implementierungsaufwand bei der Berechnung von Produktionsablaufentscheidungen bewahren können. Zum anderen wurde anhand der Fallstudien die Funktionalität von verschiedenen Komponenten der entwickelten Methode aufgezeigt.

Bezugnehmend auf die in Abschnitt 1.1 formulierte Forschungshypothese ist hervorzuheben, dass die untersuchten RL-Verfahren nicht nur bessere Lösungen als die ebenfalls echtzeitfähigen Prioritätsregeln generieren, sondern mitunter sogar bessere Ergebnisse erzielen als die problemspezifische ISBO-Heuristik

sowie diverse Metaheuristiken. Diese Erkenntnis ist bemerkenswert, insbesondere weil im Vorfeld die Annahme bestand, dass problemspezifische Heuristiken und Metaheuristiken eine höhere Lösungsgüte zu Lasten des Implementierungsaufwands und ihrer Übertragbarkeit (problemspezifische Heuristiken) bzw. ihrer Rechenzeit (Metaheuristiken) aufweisen. Metaheuristiken sind aufgrund ihrer iterativen stochastischen Suchweise nicht echtzeitfähig, während die ISBO-Heuristik einen hohen spezifischen Implementierungsaufwand erfordert. Demgegenüber sind gewöhnlich alle RL-trainierten Agenten in der Lage, einen vollständigen Produktionsablaufplan in weniger als einer Sekunde zu generieren, was für die meisten produktionslogistischen Anwendungsfälle eine ausreichend kleine Zeitschranke darstellt, um echtzeitfähig zu agieren. Überraschenderweise beansprucht das Agententraining lediglich eine in etwa genauso hohe oder gar geringere Rechenzeit als die meisten Metaheuristiken benötigen, um auf einer Probleminstanz eine anforderungsgerechte Lösung zu finden.

Hinsichtlich des Implementierungsaufwands von RL-Verfahren verglichen zu konventionellen Lösungsmethoden ist ein abschließendes Urteil schwierig, da sich dieser nur schwer objektiv quantifizieren lässt. Jedoch kann attestiert werden, dass die untersuchten RL-Verfahren hinsichtlich ihres Funktionsprinzips ähnlich generisch anwendbar sind wie Metaheuristiken. Allerdings müssen für gradientenabhängige RL-Verfahren eine Vielzahl von Besonderheiten bei der Modellierung der Agentenumgebung beachtet werden, insbesondere dass diese die Eigenschaften eines MEP erfüllen muss. Gradientenfreie RL-Verfahren hingegen sind von diesen Besonderheiten nicht betroffen, sodass ihr Implementierungsaufwand mit Metaheuristiken vergleichbar ist. Sowohl für gradientenabhängige als auch für gradientenfreie RL-Verfahren müssen der Zustands- und Aktionsraum sowie die Belohnungsfunktion stets problemspezifisch implementiert werden. Diesbezüglich besteht jedoch bei der Anwendung von Metaheuristiken ein ähnlicher Aufwand, nämlich für die problemspezifische Kodierung von Lösungskandidaten sowie für die problemspezifische Gestaltung der Fitnessfunktion.

Abschließend hat das wissenschaftliche Vorgehen zum Ziel, die in Kapitel 1 formulierten Forschungsfragen zu beantworten. Tabelle 7.1 beinhaltet die Forschungsfragen sowie deren Beantwortung. Zusammengefasst konnten die im Vorfeld formulierten Forschungsfragen im Rahmen dieser Arbeit beantwortet werden. Im Zuge der Entwicklung der Methode haben sich jedoch weitere Forschungsbedarfe ergeben, die im Ausblick dieser Arbeit skizziert werden.

Tabelle 7.1 Beantwortung der Forschungsfragen aus Abschnitt 1.2

Forschungsfrage	Antwort	Abschnitt
FF1: Auf welche Art und Weise können Methoden des bestärkenden Lernens für die Produktionsablaufplanung angewandt werden?	Für die Ressourcenbelegungsplanung, für die Auftragsreihenfolgeplanung, für die Losgrößenplanung sowie für die Reparatur von ungültigen Ablaufplänen.	4.1, 5.2
FF2: Wie können Entscheidungsprobleme der Produktionsablaufplanung als maschinelle Lernaufgabe formuliert werden, sodass eine Problemlösung mithilfe bestärkender Lernverfahren realisiert werden kann?	Ressourcenbelegungsplanung: Über diskreten Aktionsraum (Klassifikationsproblem), siehe Abbildung 5.11. Auftragsreihenfolgeplanung: Entweder über diskreten Aktionsraum, siehe Abbildung 5.14, oder über kontinuierlichen Aktionsraum (Regressionsproblem), siehe Abbildung 5.12 bzw. Abbildung 5.13. Losgrößenplanung: Wird implizit über die Auftragsreihenfolgeplanung berücksichtigt (Auswahl eines Folgeauftrags derselben Familie → Fortführung eines bestehenden Loses, andernfalls Start eines neuen Loses). Reparatur von ungültigen Ablaufplänen: Über diskrete Aktionsräume, wobei bspw. jedes Ausgabeneuron eine Umplanungsoperation repräsentiert (z. B. Verschiebung eines Auftrag um eine Position in der Sequenz nach hinten).	4.1.5 5.3.2
FF3: Welche produktionslogistischen Daten sind geeignet, um Entscheidungsmodelle für die Produktionsablaufplanung mithilfe von bestärkenden Lernen zu trainieren?	Siehe Tabelle 5.3 für eine allgemeine Übersicht bzw. Tabelle 6.2, Tabelle 6.6 und Tabelle 6.10 sowie Abbildung 6.12 und Abbildung 6.13 für fallstudienspezifische Beispiele.	5.3.2, 6.1.2, 6.2.2, 6.3.2, 6.3.3

(Fortsetzung)

Tabelle 7.1 (Fortsetzung)

Forschungsfrage	Antwort	Abschnitt
FF4: Wie müssen Belohnungsfunktionen gestaltet sein, um das bestärkende Anlernen von Entscheidungsmodellen für die Produktionsablaufplanung zu steuern?	Für die gradientenabhängige Ressourcenbelegungsplanung: Gewöhnlich belastungsorientierte Belohnungsfunktionen, die Agentenentscheidungen auf Basis der resultierenden Arbeitslast auf Produktionsressourcen bewerten. Für die gradientenabhängige Reihenfolge- und implizite Losgrößenplanung: Zeitbasierte Belohnungsfunktionen, die Agentenentscheidungen auf Basis von (erwarteten) Fertigstellungszeitpunkten bewerten. Für die gradientenabhängige Reparatur von Ablaufplänen: Bspw. durch Quantifizierung der Differenz in der Lösungsgüte /-zulässigkeit nach Umplanungsoperation. Für gradientenfreie Verfahren im Allgemeinen: Unmittelbare Verwendung der Zielfunktion des Optimierungsproblems.	4.1.5 5.3.5
FF5: Wie müssen Methoden des bestärkenden Lernens sowie durch bestärkendes Lernen trainierte Entscheidungsmodelle mit dem Prozess der Produktionsablaufplanung sowie mit den Prozessen des zugrundeliegenden produktionslogistischen Systems integriert werden?	Integration mit dem Prozess der Produktionsablaufplanung: Siehe Abbildung 5.2, Abbildung 5.4 und Abbildung 5.5. Integration mit den Prozessen des zugrundeliegenden produktionslogistischen Systems: Siehe Abbildungen 5.15 bis 5.20 für gradientenabhängige sowie Abbildung 5.21 und Abbildung 5.22 für gradientenfreie RL-Verfahren.	5.2, 5.2.1, 5.2.2, 5.3.3

7.2 Ausblick

Aufgrund von steigenden Kundenanforderungen und komplexeren Produktionsprozessen erreichen bereits heutzutage herkömmliche Lösungsverfahren für die Produktionsablaufplanung die Grenzen ihrer Leistungsfähigkeit. Vor diesem Hintergrund wurde in der vorliegenden Arbeit aufgezeigt, dass RL-Verfahren eine sinnvolle Alternative für die Produktionsablaufplanung unter beschränkter Zeit darstellen kann. Für viele klein- und mittelständische Unternehmen stellt der Einsatz von RL-Verfahren derzeit noch eine Herausforderung dar, insbesondere weil die agentenbasierte Entscheidungsfindung zeitpunktaktuelle Daten von Aufträgen und des Produktionssystems erfordert. Mit der fortschreitenden Digitalisierung der Produktion werden zukünftig RL-Verfahren einfacher, leistungsfähiger und schneller in die Anwendung gebracht werden können.

Durch die Einführung von ERP-Systemen können bereits heute viele Unternehmen auf Echtzeitdaten aus dem Produktionsbetrieb zugreifen. Eine durchgängige digitale Abbildung von Unternehmensprozessen innerhalb von ERP-Systemen vereinfacht die Datenbereitstellung für RL-Verfahren beträchtlich. Ferner werden konventionelle Werkzeugmaschinen und Montagearbeitsplätze zunehmend funktionsreicher, intelligenter, und digitaler. Durch den Wandel von Produktionsressourcen zu cyber-physischen Systemen kann eine Vielzahl von zusätzlichen Informationen aus dem Produktionsprozess erhoben und RL-Verfahren zur Verfügung gestellt werden. Hierunter zählen z. B. der Verschleißzustand von Werkzeugen, die Prognose von Wartungszeitpunkten, die Ausschussrate von Produkten u. v. m. In Zukunft gilt es zu erforschen, ob durch zusätzliche Informationen aus dem Produktionsbetrieb, die im Rahmen dieser Arbeit noch keine Berücksichtigung gefunden haben, die Lösungsgüte von RL-Verfahren nochmals verbessert werden kann. Ferner wird zukünftig der Einsatz von RL-Verfahren ebenfalls durch die zunehmende Verfügbarkeit von Cloud-Computing-Diensten attraktiver. Hierdurch können Unternehmen jederzeit auf leistungsfähige Hardware für das Training von Agenten zugreifen, ohne diese selbst erwerben, einrichten, administrativ verwalten und technisch warten zu müssen.

Der wissenschaftliche Beitrag dieser Arbeit ist die Formalisierung des Entwicklungsprozesses von RL-basierten Lösungsstrategien für die Produktionsablaufplanung. Eine mögliche Limitation der zu diesem Zweck entwickelten Methode ist, dass die Produktionsbereichsmodelle und die Prozessmodelle für den Entwurf von Agentenumgebungen keine innerbetrieblichen Transportprozesse berücksichtigen. Insbesondere in Fertigungssystemen, die nach dem Werkstattprinzip organisiert sind und in welchen Transporte zwischen Produktionsressourcen über freibewegliche Flurförderer abgewickelt werden, ist es u. U.

erforderlich innerbetriebliche Transporte mit in die Ablaufplanung zu integrieren. Diesbezüglich gilt es in Zukunft zu beforschen, wie die Produktionsbereichs- und Prozessmodelle erweitert werden müssen, sodass sie gleichermaßen für das Training von Agenten zur Steuerung von Transportressourcen geeignet sind.

Ferner dienten die in Kapitel 6 präsentierten Fallstudien lediglich als Nachweis, um die prinzipielle Tauglichkeit von RL-Verfahren für die Produktionsablaufplanung sowie verschiedene Aspekte der entwickelten Methode zu demonstrieren. Die vorliegende Arbeit trifft keine Aussage darüber,

- welches RL-Verfahren,
- welche Variante zur Formulierung von ML-Aufgaben,
- welche Zustandsinformationen, geschweige denn
- welche Belohnungsfunktionen

für welches Problem der Produktionsablaufplanung am geeignetsten sind, da der Schwerpunkt der Arbeit vornehmlich auf dem Vorgehen zum Entwurf und der Implementierung von RL-basierten Lösungsstrategien liegt. Dementsprechend wurde im Rahmen dieser Arbeit ebenfalls auf ausführliche Hyperparameter-Untersuchungen für einzelne RL-Verfahren verzichtet. Eine tiefgreifende Analyse von unterschiedlichen RL-Verfahren, Zustands- und Aktionsräumen sowie Belohnungsfunktionen für spezifische Problemen der Produktionsablaufplanung ist Gegenstand zukünftiger Arbeiten. Es wird erwartet, dass durch diese Maßnahmen die Lösungsgüte und die Konvergenzgeschwindigkeit von RL-basierten Lösungsstrategien nochmals verbessert werden kann.

Eine weitere Hürde für die Anwendung von RL-Verfahren im industriellen Umfeld ist die mangelnde Transparenz in den Entscheidungsprozessen von trainierten Agenten. Für Anwender*innen ist zwar ersichtlich, welche Zustände zu welchen Aktionen führen, jedoch kann nur schwer nachvollzogen werden, warum ein Agent für einen gegebenen Zustand auf eine bestimmte Entscheidung schließt. Die Intransparenz von Agentenentscheidungen erschwert die Akzeptanz von RL-Verfahren in der industriellen Anwendung. Ein tieferes Verständnis für die Entscheidungsprozesse von Agenten würde dazu beitragen, deren Einsatzgrenzen besser bewerten zu können. Des Weiteren bietet eine transparentere Entscheidungspolitik das Potenzial, diese gezielter für den Fall anzupassen, dass der Agent eine Fehlentscheidung generiert. Die Entwicklung von Methoden und Modellen, die Agentenentscheidungen verständlicher und die Verarbeitung von Zustandsdaten zu Aktionen für den Menschen transparent machen, ist eine der größten zukünftigen Herausforderungen in der RL-Forschung im Speziellen und der ML-Forschung im Allgemeinen.

Literaturverzeichnis

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C. et al. (2016): TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. Online verfügbar unter: https://arxiv.org/pdf/1603.04467.

Aczél, J. (1960): Vorlesungen über Funktionalgleichungen und ihre Anwendungen. Basel: Birkhäuser.

Adam, D. (2001): Produktions-Management. 9. Auflage. Wiesbaden: Gabler.

Altenmüller, T., Stüker, T., Waschneck, B., Kuhnle, A. und Lanza, G. (2020): Reinforcement Learning for an Intelligent and Autonomous Production Control of Complex Job-Shops under Time Constraints. In: *Production Engineering* 14 (3), S. 319–328.

Angeline, P. J., Saunders, G. M. und Pollack, J. B. (1994): An Evolutionary Algorithm that Constructs Recurrent Neural Networks. In: *IEEE Transactions on Neural Networks* 5 (1), S. 54–65.

Arinez, J., Ou, X. und Chang, Q. (2017): Gantry Scheduling for Two-Machine One-Buffer Composite Work Cell by Reinforcement Learning. In: Proceedings of the 12th International Manufacturing Science and Engineering Conference (ASME 2017). Volume 4: Bio and Sustainable Manufacturing. Los Angeles, CA, USA, 04.-08. Juni. New York City, NY, USA: American Society of Mechanical Engineers, S. 1–7.

Arulkumaran, K., Deisenroth, M. P., Brundage, M. und Bharath, A. A. (2017): Deep Reinforcement Learning: A Brief Survey. In: *IEEE Signal Processing Magazine* 34 (6), S. 26–38.

Arviv, K., Stern, H. und Edan, Y. (2016): Collaborative Reinforcement Learning for a Two-Robot Job Transfer Flow-Shop Scheduling Problem. In: *International Journal of Production Research* 54 (4), S. 1196–1209.

Aurich, P., Nahhas, A., Reggelin, T. und Tolujew, J. (2016): Simulation-based Optimization for Solving a Hybrid Flow Shop Scheduling Problem. In: Roeder, T. M., Frazier, P. I., Szechtman, R. und Zhou, E. (Hg.): Proceedings of the 2016 Winter Simulation Conference (WSC'16). Washington, DC, USA, 11.-14. Dezember. Piscataway, NJ, USA: IEEE, S. 2809–2819.

Aydin, M. und Öztemel, E. (2000): Dynamic Job-Shop Scheduling using Reinforcement Learning Agents. In: *Robotics and Autonomous Systems* 33 (2–3), S. 169–178.

S. Lang, *Methoden des bestärkenden Lernens für die Produktionsablaufplanung*,
https://doi.org/10.1007/978-3-658-41751-2

Bahdanau, D., Kyunghyun, C. und Bengio, Y. (2015): Neural Machine Translation by Jointly Learning to Align and Translate. In: Bengio, Y. und LeCun, Y. (Hg.): Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015). San Diego, CA, USA, 07.-09. Mai.

Baker, K. R. und Trietsch, D. (2009): Principles of Sequencing and Scheduling. Hoboken, NJ, USA: John Wiley.

Banks, J., Carson II, J. S., Nelson, B. L. und Nicol, D. M. (2010): Discrete-Event System Simulation. 5. Auflage. Upper Saddle River, N.J., London: Pearson Education.

Barto, A. G., Sutton, R. S. und Anderson, C. W. (1983): Neuronlike Adaptive Elements that can Solve Difficult Learning Control Problems. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 13 (5), S. 834–846.

Bauernhansl, T., Hompel, M. ten und Vogel-Heuser, B. (2014): Industrie 4.0 in Produktion, Automatisierung und Logistik. Anwendung, Technologien, Migration. Wiesbaden: Springer Vieweg.

Baykasoğlu, A. und Özbakır, L. (2009): A Grammatical Optimization Approach for Integrated Process Planning and Scheduling. In: *Journal of Intelligent Manufacturing* 20 (2), S. 211–221.

Becker, J., Niehaves, B., Olbrich, S. und Pfeiffer, D. (2009): Forschungsmethodik einer Integrationsdisziplin – Eine Fortführung und Ergänzung zu Lutz Heinrichs „Beitrag zur Geschichte der Wirtschaftsinformatik“ aus gestaltungsorientierter Perspektive. In: Becker, J., Krcmar, H. und Niehaves, B. (Hg.): Wissenschaftstheorie und gestaltungsorientierte Wirtschaftsinformatik. Heidelberg: Physica-Verlag Heidelberg.

Becker, J. und Pfeiffer, D. (2006): Beziehungen zwischen behavioristischer und konstruktionsorientierter Forschung in der Wirtschaftsinformatik. In: Zelewski, S. und Akca, N. (Hg.): Fortschritt in den Wirtschaftswissenschaften. Wissenschaftstheoretische Grundlagen und exemplarische Anwendungen. Wiesbaden: DUV Deutscher Universitäts-Verlag.

Belew, R. K., McInerney John und Schraudolph, N. N. (1990): Evolving Networks: Using the Genetic Algorithm with Connectionist Learning. CSE Technical Report #CS90–174. University of California San Diego, Computer Science and Engineering Department, Cognitive Computer Science Research Group (C-014).

Bellman, R. E. (1951): On Some Dynamic Linear Programming Problems. RAND Corporation. Santa Monica, CA, USA. Online verfügbar unter: https://apps.dtic.mil/sti/citations/AD0603981.

Bellman, R. E. (1957): Dynamic Programming. Princeton, NJ, USA: Princeton University Press.

Bendavid, I. und Golany, B. (2008): Setting Gates for Activities in the Stochastic Project Scheduling Problem Through the Cross Entropy Methodology. In: Şerifoğlu, F. und Bilge, Ü. (Hg.): Proceedings of the Eleventh International Workshop on Project Management and Scheduling (PMS 2008). Istanbul, Türkei, 28.-30. April. Istanbul, Türkei: Özbaş Basım ve Tanıtım Hizmetleri, S. 31–34.

Bendavid, I. und Golany, B. (2009): Setting Gates for Activities in the Stochastic Project Scheduling Problem through the Cross Entropy Methodology. In: *Annals of Operations Research* 172 (1), S. 259–276.

Bergstra, J., Bardenet, R., Bengio, Y. und Kégl, B. (2011): Algorithms for Hyper-Parameter Optimization. In: Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., Weinberger und Killian Q. (Hg.): Advances in Neural Information Processing Systems 24. 25th

Conference on Neural Information Processing Systems (NeurIPS 2011). Granada, ESP, 12.-17. Dezember. Red Hook, NY, USA: Curran Associates, Inc. Online verfügbar unter: https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

Bianchi, L., Dorigo, M., Gambardella, L. M. und Gutjahr, W. J. (2009): A Survey on Metaheuristics for Stochastic Combinatorial Optimization. In: *Natural Computing* 8 (2), S. 239–287.

Biswas, S. K., Rauniyar, A. und Muhuri, P. K. (2016): Multi-Objective Bayesian Optimization Algorithm for Real-Time Task Scheduling on Heterogeneous Multiprocessors. In: Proceedings of the 2016 IEEE Congress on Evolutionary Computation (CEC 2016). Vancouver, BC, Kanada, 24.-29. Juli. Piscataway, NJ, USA: IEEE, S. 2844–2851.

Blum, C. und Roli, A. (2003): Metaheuristics in Combinatorial Optimization. Overview and Conceptual Comparison. In: *ACM Computing Surveys* 35 (3), S. 268–308.

Bouazza, W., Sallez, Y., Aissani, N. und Beldjilali, B. (2015): A Model for Manufacturing Scheduling Optimization Through Learning Intelligent Products. In: Borangiu, T., Thomas, A. und Trentesaux, D. (Hg.): Service Orientation in Holonic and Multi-agent Manufacturing, Bd. 594. Cham, CH: Springer International Publishing, S. 233–241.

Bouazza, W., Sallez, Y. und Beldjilali, B. (2017): A Distributed Approach Solving Partially Flexible Job-Shop Scheduling Problem with a Q-learning Effect. In: *IFAC-PapersOnLine* 50 (1), S. 15890–15895.

Bowman, E. H. (1959): The Schedule-Sequencing Problem. In: *Operations Research* 7 (5), S. 621–624.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. und Zaremba, W. (2016): OpenAI Gym. Online verfügbar unter: https://arxiv.org/pdf/1606.01540.

Brucker, P. (2007): Scheduling Algorithms. 5. Auflage. Berlin, Heidelberg: Springer.

Burke, E. K. und Kendall, G. (Hg.) (2014): Search Methodologies. Introductory Tutorials in Optimization and Decision Support Techniques. 2. Auflage. Boston, MA, USA: Springer US.

Campbell, H. G., Dudek, R. A. und Smith, M. L. (1970): A Heuristic Algorithm for the n Job, m Machine Sequencing Problem. In: *Management Science* 16 (10), S. 630–637.

Candelieri, A., Perego, R. und Archetti, F. (2018): Bayesian Optimization of Pump Operations in Water Distribution Systems. In: *Journal of Global Optimization* 71 (1), S. 213–235.

Celebi, M. E. und Aydin, K. (Hg.) (2016): Unsupervised Learning Algorithms. Cham, CH: Springer International Publishing.

Chandra, J. und Talavage, J. (1991): Intelligent Dispatching for Flexible Manufacturing. In: *International Journal of Production Research* 29 (11), S. 2259–2278.

Chang, H. S., Hu, J., Fu, M. C. und Marcus, S. I. (2013): Simulation-based Algorithms for Markov Decision Processes. London, UK: Springer London.

Chaudhry, I. A. und Khan, A. A. (2016): A Research Survey: Review of Flexible Job Shop Scheduling Techniques. In: *International Transactions in Operational Research* 23 (3), S. 551–591.

Chen, J.-C., Wen, C.-K., Jin, S. und Wong, K.-K. (2016): A Low Complexity Pilot Scheduling Algorithm for Massive MIMO. In: *IEEE Wireless Communications Letters* 6 (1), S. 18–21.

Chopard, B. und Tomassini, M. (2018): An Introduction to Metaheuristics for Optimization. Cham, CH: Springer International Publishing.

Coulom, R. (2006): Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search. In: van den Herik, J., Ciancarini, P. und Donkers, H. H. L. M. J. (Hg.): Computers and Games. 5th International Conference on Computers and Games (CG 2006). Turin, Italien, 29.-31. Mai. Berlin, Heidelberg: Springer, S. 72–83.

Csaszar, P., Tirpak, T. M. und Nelson, P. C. (2000): Optimization of a High-Speed Placement Machine using Tabu Search Algorithms. In: *Annals of Operations Research* 96, S. 125–147.

Dahl, O.-J. und Nygaard, K. (1966): SIMULA: An ALGOL-based Simulation Language. In: *Communications of the ACM* 9 (9), S. 671–678.

Dasgupta, D. und McGregor, D. R. (1992): Designing Application-Specific Neural Networks Using the Structured Genetic Algorithm. In: Schaffer, J. D., Whitley, D. und Eshelman, L. J. (Hg.): Proceedings of the 1992 International Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN'92). Baltimore, MD, USA, 06. Juni: IEEE Computer Society Press, S. 87–96.

DeepMind (2019): AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. Online verfügbar unter: https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii.

Dempe, S. und Schreier, H. (2006): Operations Research. Deterministische Modelle und Methoden. Wiesbaden: Teubner.

Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A. et al. (2017): OpenAI Baselines. Online verfügbar unter: https://github.com/openai/baselines.

Doerner, K. F., Gendreau, M., Greistorfer, P., Gutjahr, W., Hartl, R. F. und Reimann, M. (2007): Metaheuristics. Progress in Complex Systems Optimization. New York, NY, USA: Springer Science + Business Media.

Dokeroglu, T., Sevinc, E., Kucukyilmaz, T. und Cosar, A. (2019): A Survey on New Generation Metaheuristic Algorithms. In: *Computers & Industrial Engineering* 137, Aufsatznummer 106040.

Dolatnia, N., Fern, A. und Fern, X. (2016): Bayesian Optimization with Resource Constraints and Productio. In: Coles, A., Coles, A., Edelkamp, S., Magazzeni, D. und Sanner, S. (Hg.): Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling (AAAI 2016). London, UK, 12.-17. Juni. Palo Alto, CA, USA: AAAI Press, S. 115–123.

Domschke, W., Drexl, A., Klein, R. und Scholl, A. (2015): Einführung in Operations Research. 9. Auflage. Berlin, Heidelberg: Springer Gabler.

Dorigo, M., Maniezzo, V. und Colorni, A. (1996): Ant System: Optimization by a Colony of Cooperating Agents. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 26 (1), S. 29–41.

Drexl, A., Fleischmann, B., Günther, H.-O., Stadtler, H. und Tempelmeier, H. (1994): Konzeptionelle Grundlagen kapazitätsorientierter PPS-Systeme. In: *Zeitschrift für betriebswirtschaftliche Forschung* 46 (12), S. 1022–1045.

Du, K.-L. und Swamy, M. N. S. (2016): Search and Optimization by Metaheuristics. Techniques and Algorithms Inspired by Nature. Cham, CH: Birkhäuser.

Du, Y., Wang, T., Xin, B., Wang, L., Chen, Y. und Xing, L. (2020): A Data-Driven Parallel Scheduling Approach for Multiple Agile Earth Observation Satellites. In: *IEEE Transactions on Evolutionary Computation* 24 (4), S. 679–693.

Du, Y., Xing, L., Chen, Y., Chen, Y. und Xiong, J. (2019): An Evolvable Real-time System of Integrated Satellite Scheduling based on Cooperative Neuro Evolution of Augmenting Topologies. In: Proceedings of the 2019 IEEE Congress on Evolutionary Computation (CEC 2019). Wellington, Neuseeland, 10.-13. Juni. Piscataway, NJ, USA: IEEE, S. 1322–1329.

Dunteman, G. H. (1989): Principal Components Analysis. Newbury Park, CA, USA: SAGE Publications.

Dyckhoff, H. und Spengler, T. S. (2007): Produktionswirtschaft. Eine Einführung für Wirtschaftsingenieure. 2. Auflage. Berlin, Heidelberg: Springer.

Eberhart, R. C. und Kennedy, J. (1995): A New Optimizer Using Particle Swarm Theory. In: Proceedings of the Sixth International Symposium on Micro Machine and Human Science (MHS'95). Nagoya, Japan, 04.-06. Oktober. Piscataway, NJ, USA: IEEE, S. 39–43.

Emmons, H. und Vairaktarakis, G. (2013): Flow Shop Scheduling. Theoretical Results, Algorithms, and Applications. New York, NY, USA: Springer Science + Business Media.

Ertel, W. (2016): Grundkurs Künstliche Intelligenz. Eine praxisorientierte Einführung. 4. Auflage. Wiesbaden: Springer Vieweg.

Fleischmann, B. (2008): Begriffliche Grundlagen. In: Arnold, D., Isermann, H., Kuhn, A., Tempelmeier, H. und Furmans, K. (Hg.): Handbuch Logistik. 3. Auflage. Berlin, Heidelberg: Springer, S. 3–12.

Fonseca-Reyna, Y., Martinez, Y., Cabrera, J. M. und Méndez-Hernández, B. (2015): A Reinforcement Learning Approach for Scheduling Problems. In: *Investigacion Operacional* 36, S. 225–231.

François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G. und Pineau, J. (2018): An Introduction to Deep Reinforcement Learning. In: *Foundations and Trends in Machine Learning* 11 (3–4), S. 219–354.

Frank, M. (1999): Modellierung und Simulation – Terminologische Probleme. In: Biethahn, J., Hummeltenberg, W., Schmidt, B., Stähly, P. und Witte, T. (Hg.): Simulation als betriebliche Entscheidungshilfe. State of the Art und neuere Entwicklungen. Heidelberg: Physica-Verlag HD, S. 50–64.

Gabel, T. und Riedmiller, M. (2012): Distributed Policy Search Reinforcement Learning for Job-Shop Scheduling Tasks. In: *International Journal of Production Research* 50 (1), S. 41–61.

Gangwani, T. und Peng, J. (2018): Policy Optimization by Genetic Distillation. In: Bengio, Y. und LeCun, Y. (Hg.): Proceedings of the 6th International Conference on Learning Representations (ICLR 2018). Vancouver, Kanada, 30. April – 03. Mai. Online verfügbar unter: https://openreview.net/pdf?id=ByOnmlWC-.

Gendreau, M. und Potvin, J.-Y. (Hg.) (2010): Handbook of Metaheuristics. 2. Auflage. New York City, NY, USA: Springer Science + Business Media.

Gerpott, F. T., Lang, S., Reggelin, T., Zadek, H., Chaopaisarn, P. und Ramingwong, S. (2022): Integration of the A2C Algorithm for Production Scheduling in a Two-Stage Hybrid Flow Shop Environment. In: *Procedia Computer Science* 200, S. 585–594.

Glaser, H., Geiger, W. und Rohde, V. (1992): PPS Produktionsplanung und -steuerung. Grundlagen – Konzepte – Anwendungen. 2. Auflage. Wiesbaden: Gabler Verlag.

Gomez, F. J., Burger, D. und Miikkulainen, R. (2001): A Neuro-Evolution Method for Dynamic Resource Allocation on a Chip Multiprocessor. In: Proceedings of the International Joint Conference on Neural Networks (IJCNN'01). Washington, DC, USA, 15.-19. Juli. Piscataway, NJ, USA: IEEE, S. 2355–2360.

Gomory, R. E. (1958): An Algorithm for Integer Solutions to Linear Programs. Princeton IBM Mathematics Research Project. Technical Report. Princeton University.

Gondek, V. (2011): Hybrid Flow-Shop Scheduling mit verschiedenen Restriktionen. Heuristische Lösung und LP-basierte untere Schranken. Dissertation. Universität Duisburg-Essen. Fakultät für Mathematik.

Goodfellow, I., Courville, A. und Bengio, Y. (2016): Deep Learning. Cambridge, MA, USA: MIT Press.

Graham, R. L., Lawler, E. L., Lenstra, J. K. und Rinnooy Kan, A. H. G. (1979): Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. In: Hammer, P. L., Johnson, E. L. und Korte, B. H. (Hg.): Discrete Optimization II. Proceedings of the Advanced Research Institute on Discrete Optimization and Systems Applications, of the Systems Science Panel of NATO and of the Discrete Optimization Symposium. Amsterdam: North-Holland Publishing, S. 287–326.

Gräßler, I. (2004): Kundenindividuelle Massenproduktion. Entwicklung, Vorbereitung der Herstellung, Veränderungsmanagement. Berlin, Heidelberg: Springer.

Greschke, P., Schönemann, M., Thiede, S. und Herrmann, C. (2014): Matrix Structures for High Volumes and Flexibility in Production Systems. In: *Procedia CIRP* 17, S. 160–165.

Gromicho, J. A., van Hoorn, J. J., Saldanha-da-Gama, F. und Timmer, G. T. (2012): Solving the Job-Shop Scheduling Problem Optimally by Dynamic Programming. In: *Computers & Operations Research* 39 (12), S. 2968–2977.

Guo, F., Li, Y., Liu, A. und Liu, Z. (2020): A Reinforcement Learning Method to Scheduling Problem of Steel Production Process. In: *Journal of Physics: Conference Series* (1486), Aufsatznummer 072035.

Han, W., Guo, F. und Su, X. (2019): A Reinforcement Learning Method for a Hybrid Flow-Shop Scheduling Problem. In: *Algorithms* 12 (11), Aufsatznummer 222.

Hecht-Nielsen, R. (1987): Kolmogorov's Mapping Neural Network Existence Theorem. In: Grossberg, S., Kohonen, T. und Kosko, B. (Hg.): Proceedings of the IEEE First Annual International Conference on Neural Networks. San Diego, CA, USA, 21.-24. Juni. Piscataway, NJ, USA: IEEE.

Heger, J. und Voss, T. (2020): Dynamically Changing Sequencing Rules with Reinforcement Learning in a Job Shop System With Stochastic Influences. In: Bae, K.-H. G., Feng, B., Kim, S., Lazarova-Molnar, S., Zheng, Z., Roeder, T. und Thiesing, R. (Hg.): Proceedings of the 2020 Winter Simulation Conference (WSC'20). Orlando, FL, USA, 14.-18. Dezember. Piscataway, NJ, USA: IEEE, S. 1608–1618.

Heinrich, L. J. und Ardelt, R. G. (2012): Geschichte der Wirtschaftsinformatik. Entstehung und Entwicklung einer Wissenschaftsdisziplin. 2. Auflage. Berlin, Heidelberg: Springer.

Hevner, A. R., March, S. T., Park, J. und Ram, S. (2004): Design Science in Information Systems Research. In: *MIS Quarterly* 28 (1), S. 75–105.

Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R. et al. (2018): Stable Baselines. Online verfügbar unter: https://github.com/hill-a/stable-baselines.

Holl, A. (1999): Empirische Wirtschaftsinformatik und Erkenntnistheorie. In: Becker, J., König, W., Schütte, R., Wendt, O. und Zelewski, S. (Hg.): Wirtschaftsinformatik und Wissenschaftstheorie. Bestandsaufnahme und Perspektiven. Wiesbaden: Gabler, S. 163–207.

Holland, J. H. (1975): Adaptation in Natural and Artificial Systems. An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. Cambridge, MA, USA: MIT Press.

Hong, J. und Prabhu, V. (2001): Distributed Learning and Control for Manufacturing Systems Scheduling. In: Goos, G., Hartmanis, J., van Leeuwen, J., Monostori, L., Váncza, J. und Ali, M. (Hg.): Engineering of Intelligent Systems. Berlin, Heidelberg: Springer, S. 582–591.

Hong, J. und Prabhu, V. V. (2004): Distributed Reinforcement Learning Control for Batch Sequencing and Sizing in Just-In-Time Manufacturing Systems. In: *Applied Intelligence* 20 (1), S. 71–87.

Hoogeveen, J., Lenstra, J. und Veltman, B. (1992): Minimizing Makespan in a Multiprocessor Flowshop is Strongly NP-hard. Working Paper. Eindhoven University of Technology, Netherlands.

Hromkovič, J. (2014): Theoretische Informatik. Formale Sprachen, Berechenbarkeit, Komplexitätstheorie, Algorithmik, Kommunikation und Kryptographie. 5. Auflage. Wiesbaden: Springer Vieweg.

Hu, Q. und Yue, W. (2008): Markov Decision Processes with Their Applications. New York City, NY, USA: Springer Science + Business Media.

Hu, Y.-Q., Qian, H. und Yu, Y. (2017): Sequential Classification-Based Optimization for Direct Policy Search. In: Singh, S. und Markovitch, S. (Hg.): Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI'17). San Francisco, CA, USA, 04.-09. Februar. Palo Alto, CA, USA: AAAI Press, S. 2029–2035.

Hussain, K., Mohd Salleh, M. N., Cheng, S. und Shi, Y. (2019): Metaheuristic Research: A Comprehensive Survey. In: *Artificial Intelligence Review* 52 (4), S. 2191–2233.

Jaehn, F. und Pesch, E. (2014): Ablaufplanung. Einführung in Scheduling. Berlin, Heidelberg: Springer Gabler.

Jodlbauer, H. (2016): Produktionsoptimierung. Wertschaffende sowie kundenorientierte Planung und Steuerung. 3. Auflage. Wien, AT: Verlag Österreich.

Johnson, S. M. (1954): Optimal Two- and Three-stage Production Schedules with Setup Times Included. In: *Naval Research Logistics* 1 (1), S. 61–68.

Kallrath, J. (2013): Gemischt-ganzzahlige Optimierung. Modellierung in der Praxis. Mit Fallstudien aus Chemie, Energiewirtschaft, Papierindustrie, Metallgewerbe, Produktion und Logistik. 2. Auflage. Wiesbaden: Springer Spektrum.

Kerting, K. und Tresp, V. (2019): Maschinelles und Tiefes Lernen – Treiber der aktuellen KI-Entwicklung. Online verfügbar unter: https://www.wissenschaftsjahr.de/2019/neues-aus-der-wissenschaft/das-sagt-die-wissenschaft/treiber-der-aktuellen-ki-entwicklung/.

Kim, K., Kim, Y. und Park, S. (2020): A Probabilistic Machine Learning Approach to Scheduling Parallel Loops with Bayesian Optimization. In: *IEEE Transactions on Parallel and Distributed Systems* 32 (7), S. 1815–1827.

Kingma, D. P. und Ba, J. (2015): Adam: A Method for Stochastic Optimization. In: Bengio, Y. und LeCun, Y. (Hg.): Proceedings of the 3rd International Conference on Learning Representations (ICLR 2015). San Diego, CA, USA, 07.-09. Mai. Online verfügbar unter: https://arxiv.org/abs/1412.6980.

Kirkpatrick, S., Gelatt, C. D. und Vecchi, M. P. (1983): Optimization by Simulated Annealing. In: *Science* 220 (4598), S. 671–680.

Kis, T. und Kovács, A. (2012): A Cutting Plane Approach for Integrated Planning and Scheduling. In: *Computers & Operations Research* 39 (2), S. 320–327.

Kistner, K.-P. und Steven, M. (2001): Produktionsplanung. 3. Auflage. Heidelberg: Physica-Verlag HD.

Kitano, H. (1990): Designing Neural Networks Using Genetic Algorithms with Graph Generation System. In: *Complex Systems* 4 (4), S. 461–476.

Klug, F. (2017): Das Perlenkettenprinzip der stabilen Auftragsfolge in der Automobillogistik. In: Göpfert, I., Braun, D. und Schulz, M. (Hg.): Automobillogistik. Stand und Zukunftstrends. 3. Auflage. Wiesbaden: Springer Gabler, S. 137–160.

Klug, F. (2018): Logistikmanagement in der Automobilindustrie. Grundlagen der Logistik im Automobilbau. 2. Auflage. Berlin: Springer Vieweg.

Koch, E. (2017): Globalisierung: Wirtschaft und Politik. Chancen – Risiken – Antworten. 2. Auflage. Wiesbaden: Springer Gabler.

Kohonen, T. (1990): The Self-Organizing Map. In: *Proceedings of the IEEE* 78 (9), S. 1464–1480.

Konda, V. R. und Borkar, V. S. (1999): Actor-Critic–Type Learning Algorithms for Markov Decision Processes. In: *SIAM Journal on Control and Optimization* 38 (1), S. 94–123.

Konda, V. R. und Tsitsiklis, J. N. (1999): Actor-Critic Algorithms. In: Solla, S. A., Leen, T. K. und Müller, K.-R. (Hg.): Advances in Neural Information Processing Systems 12. The 12th International Conference on Neural Information Processing Systems (NIPS 1999). Denver, CO, USA, 29. November – 04. Dezember. Cambridge, MA, USA: MIT Press, S. 1008–1014.

Korte, B. H. und Vygen, J. (2018): Kombinatorische Optimierung. Theorie und Algorithmen. 3. Auflage. Berlin, Heidelberg: Springer.

Krizhevsky, A., Sutskever, I. und Hinton, G. E. (2012): ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira, F., Burges, C. J. C., Bottou, L. und Weinberger, K. Q. (Hg.): Advances in Neural Information Processing Systems 25. 26th Conference on Neural Information Processing Systems (NeurIPS 2012). Lake Tahoe, NV, USA, 03.-08. Dezember. Red Hook, NY, USA: Curran Associates, Inc., S. 1097–1105.

Ku, W.-Y. und Beck, J. C. (2016): Mixed Integer Programming Models for Job Shop Scheduling: A Computational Analysis. In: *Computers & Operations Research* 73, S. 165–173.

Kuhn, A. und Rabe, M. (Hg.) (1998): Simulation in Produktion und Logistik. Fallbeispielsammlung. Berlin, Heidelberg: Springer.

Kuhnle, A., Kaiser, J.-P., Theiß, F., Stricker, N. und Lanza, G. (2021): Designing an Adaptive Production Control System Using Reinforcement Learning. In: *Journal of Intelligent Manufacturing* 32 (4), S. 855–876.

Kuhnle, A., Röhrig, N. und Lanza, G. (2019): Autonomous Order Dispatching in the Semiconductor Industry Using Reinforcement Learning. In: *Procedia CIRP* 79, S. 391–396.

Kullback, S. und Leibler, R. A. (1951): On Information and Sufficiency. In: *The Annals of Mathematical Statistics* 22 (1), S. 79–86.

Kurbel, K. (2003): Produktionsplanung und -steuerung. Methodische Grundlagen von PPS-Systemen und Erweiterungen. 5. Auflage. München: Oldenbourg.

Kurbel, K. und Strunz, H. (1990): Wirtschaftsinformatik – eine Einführung. In: Kurbel, K. (Hg.): Handbuch Wirtschaftsinformatik. Stuttgart: Poeschel, S. 1–25.

Kushner, H. J. (1964): A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise. In: *Journal of Basic Engineering* 86 (1), S. 97–106.

Land, A. H. und Doig, A. G. (1960): An Automatic Method of Solving Discrete Programming Problems. In: *Econometrica* 28 (3), S. 497–520.

Lang, S., Kuetgens, M., Reichardt, P. und Reggelin, T. (2021a): Modeling Production Scheduling Problems as Reinforcement Learning Environments based on Discrete-Event Simulation and OpenAI Gym. In: *IFAC-PapersOnLine* 54 (1), S. 793–798.

Lang, S., Lanzerath, N., Reggelin, T., Müller, M. und Behrendt, F. (2020a): Integration of Deep Reinforcement Learning and Discrete-Event Simulation for Real-Time Scheduling of a Flexible Job-Shop Production. In: Bae, K.-H. G., Feng, B., Kim, S., Lazarova-Molnar, S., Zheng, Z., Roeder, T. und Thiesing, R. (Hg.): Proceedings of the 2020 Winter Simulation Conference (WSC'20). Orlando, FL, USA, 14.-18. Dezember. Piscataway, NJ, USA: IEEE.

Lang, S., Reggelin, T., Behrendt, F. und Nahhas, A. (2020b): Evolving Neural Networks to Solve a Two-Stage Hybrid Flow Shop Scheduling Problem with Family Setup Times. In: Proceedings of the 53rd Hawaii International Conference on System Sciences (HICSS 2020). Wailea, HI, USA, 07.-10. Januar. University of Hawaii at Manoa, S. 1298–1307.

Lang, S., Reggelin, T., Schmidt, J., Müller, M. und Nahhas, A. (2021b): NeuroEvolution of Augmenting Topologies for Solving a Two-Stage Hybrid Flow Shop Scheduling Problem: A Comparison of Different Solution Strategies. In: *Expert Systems with Applications* 172, Aufsatznummer 114666.

Law, A. M. (2015): Simulation Modeling and Analysis. 5. Auflage. New York, NY, USA: McGraw-Hill Education.

Lawler, E. L. (1973): Optimal Sequencing of a Single Machine Subject to Precedence Constraints. In: *Management Science* 19 (5), S. 544–546.

LeCun, Y., Bengio, Y. und Hinton, G. (2015): Deep Learning. In: *Nature* 521 (7553), S. 436–444.

Lee, S., Cho, Y. und Lee, Y. H. (2020): Injection Mold Production Sustainable Scheduling Using Deep Reinforcement Learning. In: *Sustainability* 12 (20), S. 8718.

Lew, A. und Mauch, H. (2007): Dynamic Programming. A Computational Tool. Berlin, Heidelberg: Springer.

Li, J. und Aickelin, U. (2003): A Bayesian Optimization Algorithm for the Nurse Scheduling Problem. In: Proceedings of the 2003 Congress on Evolutionary Computation (CEC '03). Canberra, Australien, 08.-12. Dezember. Piscataway, NJ, USA: IEEE, S. 2149–2156.

Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Gonzales, J. et al. (2017): Ray RLlib: A Composable and Scalable Reinforcement Learning Library. In: Abbeel, P., Duan, Y., Houthooft, R., Oh, J., Silver, D. und Singh, S. (Hg.): Deep Reinforcement Learning Symposium (DeepRL @ NeurIPS). 31st Conference on Neural Information Processing Systems (NeurIPS 2017). Long Beach, CA, USA, 04.-09. Dezember. Red Hook, NY, USA: Curran Associates, Inc. Online verfügbar unter: https://royf.org/pub/pdf/Liang2017Ray.pdf.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y. et al. (2015): Continuous control with deep reinforcement learning. Online verfügbar unter: http://arxiv.org/pdf/1509.02971v6.

Lin, C.-C., Deng, D.-J., Chih, Y.-L. und Chiu, H.-T. (2019): Smart Manufacturing Scheduling With Edge Computing Using Multiclass Deep Q Network. In: *IEEE Transactions on Industrial Informatics* 15 (7), S. 4276–4284.

Lindemann, U., Reichwald, R. und Zäh, M. F. (2006): Individualisierte Produkte – Komplexität beherrschen in Entwicklung und Produktion. Berlin, Heidelberg: Springer.

Liu, Z., Yang, Y., Zhou, M.-T. und Li, Z. (2018): A Unified Cross-entropy Based Task Scheduling Algorithm for Heterogeneous Fog Networks. In: Ramachandran, G. S., Krishnamachari, B., Chau, S. C.-K., Yang, Y. und Fung, T. (Hg.): Proceedings of the 1st ACM International Workshop on Smart Cities and Fog Computing. The 16th ACM Conference on Embedded Networked Sensor Systems (SenSys'18). Shenzhen, China, 04. November. New York, NY, USA: ACM, S. 1–6.

Lödding, H. (2016): Verfahren der Fertigungssteuerung. Grundlagen, Beschreibung, Konfiguration. 3. Auflage. Berlin, Heidelberg: Springer Vieweg.

Lorenz, U. (2020): Reinforcement Learning. Berlin, Heidelberg: Springer.

Luczak, H. und Eversheim, W. (Hg.) (1999): Produktionsplanung und -steuerung. Grundlagen, Gestaltung und Konzepte. 2. Auflage. Berlin, Heidelberg: Springer.

Luo, S. (2020): Dynamic Scheduling for Flexible Job Shop with New Job Insertions by Deep Reinforcement Learning. In: *Applied Soft Computing* 91, Aufsatznummer 106208.

Lv, S. und Liu, W. (2016): A Cross-Entropy-based Approach for Joint Process Plan Selection and Scheduling Optimization. In: *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 230 (8), S. 1525–1536.

Mandischer, M. (1993): Representation and Evolution of Neural Networks. In: Albrecht, R. F., Reeves, C. R. und Steele, N. C. (Hg.): Artificial Neural Nets and Genetic Algorithms. Wien, AT: Springer Vienna, S. 643–649.

Manne, A. S. (1960): On the Job-Shop Scheduling Problem. In: *Operations Research* 8 (2), S. 219–223.

Mao, X., ter Mors, A., Roos, N. und Witteveen, C. (2007): Using Neuro-Evolution in Aircraft Deicing Scheduling. In: Tuyls, K., Jong, S. de, Ponsen, M. und Verbeeck, K. (Hg.): Proceedings of the 2007 European Symposium on Adaptive and Learning Agents and Multi-Agent Systems (ALAMAS 2007). Maastricht, NL, 02.-03. April, S. 138–145.

March, S. T. und Smith, G. F. (1995): Design and Natural Science Research on Information Technology. In: *Decision Support Systems* 15 (4), S. 251–266.

Martínez, Y., Nowé, A., Suárez, J. und Bello, R. (2011): A Reinforcement Learning Approach for the Flexible Job Shop Scheduling Problem. In: Coello Coello, C. A. (Hg.): Learning and Intelligent Optimization. 5th International Conference on Learning and Intelligent Optimization (LION 5). Rom, IT, 17.-21. Januar. Berlin, Heidelberg: Springer, S. 253–262.

Matloff, N. (2008): Introduction to Discrete-Event Simulation and the SimPy Language. Online verfügbar unter: http://heather.cs.ucdavis.edu/~matloff/156/PLN/DESimIntro.pdf.

McCulloch, W. S. und Pitts, W. (1943): A Logical Calculus of the Ideas Immanent in Nervous Activity. In: *Bulletin of Mathematical Biophysics* 5 (4), S. 115–133.

McIntyre, A., Kallada, M., Miguel, C. G. und Feher da Silva, C. (2017a): NEAT-Python. Online verfügbar unter: https://github.com/CodeReclaimers/neat-python.

McIntyre, A., Kallada, M., Miguel, C. G. und Feher da Silva, C. (2017b): NEAT-Python Documentation. Configuration File Description. Online verfügbar unter: https://neat-python.readthedocs.io/en/latest/config_file.html.

Mello, R. F. de und Ponti, M. A. (2018): Machine Learning. A Practical Approach on the Statistical Learning Theory. Cham, CH: Springer International Publishing.

Méndez-Hernández, B. M., Rodríguez-Bazan, E. D., Martinez-Jimenez, Y., Libin, P. und Nowé, A. (2019): A Multi-objective Reinforcement Learning Algorithm for JSSP. In: Tetko, I. V., Kůrková, V., Karpov, P. und Theis, F. (Hg.): Artificial Neural Networks and Machine Learning – ICANN 2019: Theoretical Neural Computation. Cham, CH: Springer International Publishing, S. 567–584.

Meudt, T., Wonnemann, A. und Metternich, J. (2017): Produktionsplanung und -steuerung (PPS) – ein Überblick der Literatur der unterschiedlichen Einteilung von PPS-Konzepten. 2. Auflage. Hg. v. Technische Universität Darmstadt. Institut für Produktionsmanagement, Technologie und Werkzeugmaschinen. Darmstadt. Online verfügbar unter: https://tuprints.ulb.tu-darmstadt.de/6654/.

Minguillon, F. E. und Lanza, G. (2019): Coupling of Centralized and Decentralized Scheduling for Robust Production in Agile Production Systems. In: *Procedia CIRP* 79, S. 385–390.

Mitchell, T. M. (1997): Machine Learning. New York, NY, USA: McGraw-Hill.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T. et al. (2016): Asynchronous Methods for Deep Reinforcement Learning. In: Balcan, M. F. und Weinberger, K. Q. (Hg.): Proceedings of the 33rd International Conference on Machine Learning (ICML 2016). New York, NY, USA, 19.-24. Juni, S. 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. und Riedmiller, M. (2013): Playing Atari with Deep Reinforcement Learning (Technical Report, NIPS Deep Learning Workshop 2013). Online verfügbar unter: http://arxiv.org/pdf/1312.5602v1.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G. et al. (2015): Human-Level Control through Deep Reinforcement Learning. In: *Nature* 518 (7540), S. 529–533.

Mockus, J. B. (1975): On Bayesian Methods for Seeking the Extremum. In: Marčuk, G. I. (Hg.): Optimization Techniques. IFIP Technical Conference, Novosibirsk, July 1–7, 1974. Berlin, Heidelberg: Springer, S. 400–404.

Mockus, J. B. (1989): Bayesian Approach to Global Optimization. Theory and Applications. Dordrecht, NL: Springer Netherlands.

Mockus, J. B., Tiesis, V. und Zilinskas, A. (1978): The Application of Bayesian Methods for Seeking the Extremum. In: Dixon, L. C. und Szegö, G. P. (Hg.): Towards Global Optimisation 2. A Collection of Proceedings From Two Workshops Held in Varenna, Italy, in June 1976 and at the University of Bergamo, Italy, in July 1977, and Various Other Research Papers. Amsterdam, NL: North-Holland Publishing, S. 117–129.

Mokhtari, H. (2014): A Two-Stage No-Wait Job Shop Scheduling Problem by using a Neuro-Evolutionary Variable Neighborhood Search. In: *The International Journal of Advanced Manufacturing Technology* 74 (9–12), S. 1595–1610.

Moore, J. M. (1968): An n Job, One Machine Sequencing Algorithm for Minimizing the Number of Late Jobs. In: *Management Science* 15 (1), S. 102–109.

Muhuri, P. K. und Biswas, S. K. (2020): Bayesian Optimization Algorithm for Multi-Objective Scheduling of Time and Precedence Constrained Tasks in Heterogeneous Multiprocessor Systems. In: *Applied Soft Computing* 92, Aufsatznummer 106274.

Nahhas, A., Aurich, P., Reggelin, T. und Tolujevs, J. (2016): Heuristic and Metaheuristic Simulation-based Optimization for Solving a Hybrid Flow Shop Scheduling Problem. In: Bruzzone, A. G., Felice, F. de, Frydman, C., Longo, F., Massei, M. und Solis, A. (Hg.): Proceedings of the 15th International Conference on Modeling and Applied Simulation (MAS 2016). The International Multidisciplinary Modeling and Simulation Multiconference (I3M 2016). Larnaca, CY, 26.-28. September. Genova, IT: Dime Università di Genova, S. 95–103.

Nebl, T. (1997): Einführung in die Produktionswirtschaft. 2. Auflage. München: Oldenbourg.

Nilsson, N. J. (2010): The Quest for Artificial Intelligence. A History of Ideas and Achievements. Cambridge, UK: Cambridge University Press.

Ohno, T., Hof, W. und Rother, M. (2013): Das Toyota-Produktionssystem. 3. Auflage. Frankfurt, New York: Campus.

Osman, I. H. und Laporte, G. (1996): Metaheuristics: A Bibliography. In: *Annals of Operations Research* 63 (5), S. 511–623.

Ou, X., Chang, Q., Arinez, J. und Zou, J. (2018): Gantry Work Cell Scheduling through Reinforcement Learning with Knowledge-guided Reward Setting. In: *IEEE Access* 6, S. 14699–14709.

Ou, X., Chang, Q. und Chakraborty, N. (2019): Simulation Study on Reward Function of Reinforcement Learning in Gantry Work Cell Scheduling. In: *Journal of Manufacturing Systems* 50, S. 1–8.

Page, B. (1991): Diskrete Simulation. Eine Einführung mit Modula-2. Berlin, Heidelberg: Springer.

Palombarini, J. und Martínez, E. (2012): SmartGantt – An Intelligent System for Real Time Rescheduling based on Relational Reinforcement Learning. In: *Expert Systems with Applications* 39 (11), S. 10251–10268.

Palombarini, J. A. und Martinez, E. C. (2018): Automatic Generation of Rescheduling Knowledge in Socio-technical Manufacturing Systems using Deep Reinforcement Learning. In: Proceedings of the 2018 IEEE Biennial Congress of Argentina (ARGENCON 2018). San Miguel de Tucumán, AR, 06.-08. Juni. Piscataway, NJ, USA: IEEE, S. 1–8.

Palombarini, J. A. und Martínez, E. C. (2019): Closed-loop Rescheduling using Deep Reinforcement Learning. In: *IFAC-PapersOnLine* 52 (1), S. 231–236.

Pan, C.-H. (1997): A Study of Integer Programming Formulations for Scheduling Problems. In: *International Journal of Systems Science* 28 (1), S. 33–41.

Pardo, F., Tavakoli, A., Levdik, V. und Kormushev, P. (2018): Time Limits in Reinforcement Learning. In: Dy, J. und Krause, A. (Hg.): Proceedings of the 35th International Conference on Machine Learning (ICML 2018). Stockholm, SWE, 10.-15. Juli, S. 4045–4054.

Park, I.-B., Huh, J., Kim, J. und Park, J. (2020): A Reinforcement Learning Approach to Robust Scheduling of Semiconductor Manufacturing Facilities. In: *IEEE Transactions on Automation Science and Engineering* 17 (3), S. 1420–1431.

Park, J., Chun, J., Kim, S. H., Kim, Y. und Park, J. (2021): Learning to Schedule Job-shop Problems: Representation and Policy Learning using Graph Neural Network and

Reinforcement Learning. In: *International Journal of Production Research* 59 (11), S. 3360–3377.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G. et al. (2019): PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. und Garnett, R. (Hg.): Advances in Neural Information Processing Systems 32. 33rd Conference on Neural Information Processing Systems (NeurIPS 2019). Vancouver, CA, 08.-14. Dezember. Red Hook, NY, USA: Curran Associates, Inc. Online verfügbar unter: https://arxiv.org/pdf/1912.01703.

Paternina-Arboleda, C. D. und Das, T. K. (2005): A Multi-Agent Reinforcement Learning Approach to Obtaining Dynamic Control Policies for Stochastic Lot Scheduling Problem. In: *Simulation Modelling Practice and Theory* 13 (5), S. 389–406.

Pelikan, M., Goldberg, D. E. und Cantú-Paz, E. (1999): BOA: The Bayesian Optimization Algorithm. In: Banzhaf, W. und Daida, J. M. (Hg.): Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation (GECCO'99) – Volumen 1. Orlando, FL, USA, 13.-17. Juli. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc (GECCO'99), S. 525–532.

Pinedo, M. L. (2009): Planning and Scheduling in Manufacturing and Services. 2. Auflage. New York, NY, USA: Springer-Verlag New York.

Pinedo, M. L. (2016): Scheduling. Cham, CHE: Springer International Publishing.

Potts, C. N. und van Wassenhove, L. N. (1985): A Branch and Bound Algorithm for the Total Weighted Tardiness Problem. In: *Operations Research* 33 (2), S. 363–377.

Qian, H. und Yu, Y. (2021): Derivative-Free Reinforcement Learning: A Review. In: *Frontiers of Computer Science* 15 (6), Aufsatznummer 156336.

Qu, S., Chu, T., Wang, J., Leckie, J. und Jian, W. (2015): A Centralized Reinforcement Learning Approach for Proactive Scheduling in Manufacturing. In: Proceedings of the 20th IEEE Conference on Emerging Technologies & Factory Automation (ETFA 2015). Luxemburg, LUX, 08.-11. September. Piscataway, NJ, USA: IEEE, S. 1–8.

Qu, S., Wang, J., Govil, S. und Leckie, J. O. (2016a): Optimized Adaptive Scheduling of a Manufacturing Process System with Multi-skill Workforce and Multiple Machine Types: An Ontology-based, Multi-agent Reinforcement Learning Approach. In: *Procedia CIRP* 57, S. 55–60.

Qu, S., Wang, J. und Jasperneite, J. (2018): Dynamic Scheduling in Large-scale Stochastic Processing Networks for Demand-driven Manufacturing Using Distributed Reinforcement Learning. In: Proceedings of the 23th IEEE Conference on Emerging Technologies & Factory Automation (ETFA 2018). Turin, ITA, 04.-07. September. Piscataway, NJ, USA: IEEE, S. 433–440.

Qu, S., Wang, J. und Shivani, G. (2016b): Learning Adaptive Dispatching Rules for a Manufacturing Process System by using Reinforcement Learning Approach. In: Proceedings of the 21st IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2016). The 21st IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2016). Berlin, 06.-09. September. Piscataway, NJ, USA: IEEE, S. 1–8.

Rabelo, L. C., Jones, A. und Yih, Y. (1994): Development of a real-time learning scheduler using reinforcement learning concepts. In: Proceedings of 1994 9th IEEE International Symposium on Intelligent Control. 1994 9th IEEE International Symposium on Intelligent Control. Columbus, OH, USA, 16.-18. August: IEEE, S. 291–296.

Rajkumar, M., Asokan, P., Page, T. und Arunachalam, S. (2010): A GRASP Algorithm for the Integration of Process Planning and Scheduling in a Flexible Job-Shop. In: *IJMR* 5 (2), S. 230–251.

Ramirez-Hernandez, J. A. und Fernandez, E. (2007): Control of a Re-Entrant Line Manufacturing Model with a Reinforcement Learning Approach. In: Wani, M. A., Kantardzic, M. M., Li, T., Liu, Y., Kurgan, L., Ye, J. et al. (Hg.): Proceedings of the Sixth International Conference on Machine Learning and Applications (ICMLA 2007). The Sixth International Conference on Machine Learning and Applications (ICMLA 2007). Cincinnati, OH, USA, 13.-15. Dezember. Piscataway, NJ, USA: IEEE, S. 330–335.

Rechenberg, I. und Eigen, M. (1973): Evolutionsstrategie. Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Stuttgart-Bad Cannstadt: Frommann-Holzboog.

Reeves, C. R. (2010): Genetic Algorithms. In: Gendreau, M. und Potvin, J.-Y. (Hg.): Handbook of Metaheuristics. 2. Auflage. New York City, NY, USA: Springer Science + Business Media, S. 109–140.

Rich, E. (1983): Artificial intelligence. New York: McGraw-Hill.

Rolf, B., Reggelin, T., Nahhas, A., Lang, S. und Müller, M. (2020a): Assigning dispatching rules using a genetic algorithm to solve a hybrid flow shop scheduling problem. In: *Procedia Manufacturing* 42, S. 442–449.

Rolf, B., Reggelin, T., Nahhas, A., Muller, M. und Lang, S. (2020b): Scheduling Jobs in a Two-Stage Hybrid Flow Shop with a Simulation-Based Genetic Algorithm and Standard Dispatching Rules. In: Bae, K.-H. G., Feng, B., Kim, S., Lazarova-Molnar, S., Zheng, Z., Roeder, T. und Thiesing, R. (Hg.): Proceedings of the 2020 Winter Simulation Conference (WSC'20). Orlando, FL, USA, 14.-18. Dezember. Piscataway, NJ, USA: IEEE, S. 1584–1595.

Roman, I., Ceberio, J., Mendiburu, A. und Lozano, J. A. (2016): Bayesian Optimization for Parameter Tuning in Evolutionary Algorithms. In: Proceedings of the 2016 IEEE Congress on Evolutionary Computation (CEC 2016). Vancouver, BC, Kanada, 24.-29. Juli. Piscataway, NJ, USA: IEEE, S. 4839–4845.

Rubinstein, R. (1999): The Cross-Entropy Method for Combinatorial and Continuous Optimization. In: *Methodology And Computing In Applied Probability* 1 (2), S. 127–190.

Rubinstein, R. Y. (1997): Optimization of Computer Simulation Models with Rare Events. In: *European Journal of Operational Research* 99 (1), S. 89–112.

Ruiz, R. und Vázquez-Rodríguez, J. A. (2010): The hybrid flow shop scheduling problem. In: *European Journal of Operational Research* 205 (1), S. 1–18.

Russell, S. J. und Norvig, P. (2016): Artificial intelligence. A modern approach. Unter Mitarbeit von Ernest Davis und Douglas Edwards. Third edition, Global edition. Boston, Columbus, Indianapolis, New York, San Francisco, Upper Saddle River, Amsterdam, Cape Town, Dubai, London, Madrid, Milan, Munich, Paris, Montreal, Toronto, Delhi, Mexico City, Sao Paulo, Sydney, Hong Kong, Seoul, Singapore, Taipei, Tokyo: Pearson.

Rüttimann, B. G. und Stöckli, M. T. (2016): Going beyond Triviality. The Toyota Production System—Lean Manufacturing beyond Muda and Kaizen. In: *Journal of Service Science and Management* 09 (02), S. 140–149.

Sahni, S. (1979): Preemptive Scheduling with Due Dates. In: *Operations Research* 27 (5), S. 925–934.

Samuel, A. L. (1959): Some Studies in Machine Learning Using the Game of Checkers. In: *IBM Journal of Research and Development* 3 (3), S. 210–229.

Scheer, A.-W. (1995): Wirtschaftsinformatik. Referenzmodelle für industrielle Geschäftsprozesse. Sechste, durchgesehene Auflage. Berlin, Heidelberg, s.l.: Springer Berlin Heidelberg.

Schenk, M., Wirth, S. und Müller, E. (2014): Fabrikplanung und Fabrikbetrieb. Methoden für die wandlungsfähige, vernetzte und ressourceneffiziente Fabrik. 2., vollständig überarbeitete und erweiterte Auflage 2014. Berlin: Springer Vieweg.

Scholz-Reiter, B., Wirth, F., Freitag, M., Dashkovskiy, S., Jagalski, T., Beer, C. de und Rüffer, B. (2007): Mathematical Models of Autonomous Logistic Processes. In: Windt, K. und Hülsmann, M. (Hg.): Understanding autonomous cooperation and control in logistics. The impact of autonomy on management, information, communication and material flow. Berlin, New York: Springer, S. 121–138.

Schotten, M. (1998): Produktionsplanung und -steuerung. Grundlagen, Gestaltung und Konzepte. Hg. v. Holger Luczak und Walter Eversheim. Berlin, Heidelberg: Springer Berlin Heidelberg (VDI-Buch).

Schriber, T. J., Brunner, D. T. und Smith, J. S. (2017): Inside discrete-event simulation software: How it works and why it matters. In: Chan, W. K., D'Ambrogio, A., Zacharewicz, G., Mustafee, N., Wainer, G. und Page, E. H. (Hg.): Proceedings of the 2017 Winter Simulation Conference (WSC'17). Las Vegas, NV, USA, 03.-06. Dezember. Piscataway, NJ, USA: IEEE, S. 735–749.

Schuh, G., Brandenburg, U. und Cuber, S. (2012a): Aufgaben. In: Schuh, G. und Stich, V. (Hg.): Produktionsplanung und -steuerung 1. Grundlagen der PPS. 4. Aufl. Berlin: Springer (SpringerLink Bücher), S. 29–81.

Schuh, G., Schmidt, C. und Helmig, J. (2012b): Prozesse. In: Schuh, G. und Stich, V. (Hg.): Produktionsplanung und -steuerung 1. Grundlagen der PPS. 4. Aufl. Berlin: Springer (SpringerLink Bücher), S. 109–194.

Schuh, G. und Stich, V. (Hg.) (2012a): Produktionsplanung und -steuerung 1. Grundlagen der PPS (SpringerLink Bücher). 4. Aufl. Berlin: Springer.

Schuh, G. und Stich, V. (Hg.) (2012b): Produktionsplanung und -steuerung 2. Evolution der PPS (VDI-Buch). 4., überarb. Aufl. Berlin: Springer.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. und Moritz, P. (2015): Trust Region Policy Optimization. In: Bach, F. und Blei, D. (Hg.): Proceedings of the 32nd International Conference on Machine Learning. The 32nd International Conference on Machine Learning Conference (ICML 2015). Lille, Frankreich, 06.-11. Juli, S. 1889–1897.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. und Klimov, O. (2017): Proximal Policy Optimization Algorithms. Online verfügbar unter: https://arxiv.org/pdf/1707.06347.

Sexton, R. S., Dorsey, R. E. und Johnson, J. D. (1999): Optimization of neural networks: A comparative analysis of the genetic algorithm and simulated annealing. In: *European Journal of Operational Research* 114 (3), S. 589–601.

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P. und Freitas, N. de (2016): Taking the Human Out of the Loop: A Review of Bayesian Optimization. In: *Proceedings of the IEEE* 104 (1), S. 148–175.

Sharda, R., Voß, S., Alba, E. und Martí, R. (Hg.) (2006): Metaheuristic Procedures for Training Neutral Networks (SpringerLink Bücher 36). Boston, MA: Springer US.

Shi, D., Fan, W., Xiao, Y., Lin, T. und Xing, C. (2020): Intelligent Scheduling of Discrete Automated Production Line via Deep Reinforcement Learning. In: *International Journal of Production Research*, S. 1–19.

Shiue, Y.-R., Lee, K.-C. und Su, C.-T. (2018): Real-Time Scheduling for a Smart Factory using a Reinforcement Learning Approach. In: *Computers & Industrial Engineering* 125, S. 604–614.

Shiue, Y.-R., Lee, K.-C. und Su, C.-T. (2020): A Reinforcement Learning Approach to Dynamic Scheduling in a Product-Mix Flexibility Environment. In: *IEEE Access* 8, S. 106542–106553.

Siarry, P. (Hg.) (2016): Metaheuristics. Cham: Springer.

Siepermann, C. (2013): Produktionsplanung und -steuerung. Hg. v. Gabler Wirtschaftslexikon. Wiesbaden. Online verfügbar unter: https://wirtschaftslexikon.gabler.de/definition/produktionsplanung-und-steuerung-51585/version-176072.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G. et al. (2016): Mastering the game of Go with deep neural networks and tree search. In: *Nature* 529 (7587), S. 484–489. Online verfügbar unter https://www.nature.com/articles/nature16961.pdf.

Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D. und Riedmiller, M. (2014): Deterministic Policy Gradient Algorithms. In: Xing, E. P. und Jebara, T. (Hg.): Proceedings of the 31st International Conference on International Conference on Machine Learning. The 31st International Conference on Machine Learning Conference (ICML 2014), 21.-26. Juni. Peking, China, 387–395.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A. et al. (2017): Mastering the game of Go without human knowledge. In: *Nature* 550 (7676), S. 354–359.

Sipser, M. (2006): Introduction to the theory of computation. Boston: Course Technology.

Sivanandam, S. N. und Deepa, S. N. (Hg.) (2008): Introduction to Genetic Algorithms. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg.

Snoek, J., Larochelle, H. und Adams, R. P. (2012): Practical Bayesian Optimization of Machine Learning Algorithms. In: Pereira, F., Burges, C. J. C., Bottou, L. und Weinberger, K. Q. (Hg.): Advances in Neural Information Processing Systems 25. 26th Conference on Neural Information Processing Systems (NeurIPS 2012). Lake Tahoe, NV, USA, 03.-08. Dezember. Red Hook, NY, USA: Curran Associates, Inc., S. 1–9.

Spath, D. (Hg.) (2013): Produktionsarbeit der Zukunft – Industrie 4.0. Studie. Fraunhofer-Institut für Arbeitswirtschaft und Organisation. Stuttgart: Fraunhofer-Verlag.

Stanley, K. O. (2004): Efficient Evolution of Neural Networksthrough Complexification. Dissertation. The University of Texas, Austin, TX. Faculty of the Graduate School.

Stanley, K. O. und Miikkulainen, R. (2002a): Efficient evolution of neural network topologies. In: Proceedings of the 2002 Congress on Evolutionary Computation, CEC'02. Honolulu, HI, USA, 12.-17. Mai. Congress on Evolutionary Computation; IEEE World Congress on Computational Intelligence; Institute of Electrical and Electronics Engineers. Piscataway, NJ, USA: IEEE, S. 1757–1762.

Stanley, K. O. und Miikkulainen, R. (2002b): Efficient Reinforcement Learning through Evolving Neural Network Topologies. In: Langdon, W. B., Cantú-Paz, E. E. und Mathias, K. E. (Hg.): Proceedings of the 4th Annual Conference on Genetic and Evolutionary

Computation (GECCO'02). 4th Annual Conference on Genetic and Evolutionary Computation (GECCO'02). New York City, NY, USA, 09.-13. Juli. San Francisco, CA, USA: Morgan Kaufmann, S. 569–577.

Stanley, K. O. und Miikkulainen, R. (2002c): Evolving neural networks through augmenting topologies. In: *Evolutionary computation* 10 (2), S. 99–127.

Staufen AG (2019): Industrie 4.0 Index. Deutscher Industrie 4.0 Index 2019. Köngen. Online verfügbar unter: https://www.staufen.ag/fileadmin/HQ/02-Company/05-Media/2-Studies/STAUFEN.-Studie-Industrie-4-0-index-2019-de.pdf, zuletzt geprüft am 05.11.2020.

Stecco, G., Cordeau, J.-F. und Moretti, E. (2008): A branch-and-cut algorithm for a production scheduling problem with sequence-dependent and time-dependent setup times. In: *Computers & Operations Research* 35 (8), S. 2635–2655.

Stricker, N., Kuhnle, A., Sturm, R. und Friess, S. (2018): Reinforcement Learning for Adaptive Order Dispatching in the Semiconductor Industry. In: *CIRP Annals* 67 (1), S. 511–514.

Such, F. P., Madhavan, V., Conti, E., Lehman, J., Stanley, K. O. und Clune, J. (2017): Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning. Online verfügbar unter: https://arxiv.org/pdf/1712.06567.

Sun, L., Lin, L., Wang, Y., Gen, M. und Kawakami, H. (2015): A Bayesian Optimization-based Evolutionary Algorithm for Flexible Job Shop Scheduling. In: *Procedia Computer Science* 61, S. 521–526.

Sutton, R. S. und Barto, A. (2018): Reinforcement learning. An introduction. 2th ed. Cambridge, MA, London: The MIT Press.

Sutton, R. S., McAllester, D., Singh, S. und Mansour, Y. (1999): Policy Gradient Methods for Reinforcement Learning with Function Approximation. In: Solla, S. A., Leen, T. K. und Müller, K.-R. (Hg.): Advances in Neural Information Processing Systems 12. The 12th International Conference on Neural Information Processing Systems (NIPS 1999). Denver, CO, USA, 29. November – 04. Dezember. Cambridge, MA, USA: MIT Press, S. 1057–1063.

Swamynathan, M. (2019): Mastering Machine Learning with Python in Six Steps. Berkeley, CA: Apress.

Tan, Q., Tong, Y., Wu, S. und Li, D. (2019): Modeling, planning, and scheduling of shop-floor assembly process with dynamic cyber-physical interactions: a case study for CPS-based smart industrial robot production. In: *The International Journal of Advanced Manufacturing Technology* 105 (9), S. 3979–3989.

Tanaka, Y. und Yoshida, T. (1999): An Application of Reinforcement Learning to Manufacturing Scheduling Problems. In: IEEE SMC'99 Conference Proceedings. The 1999 IEEE International Conference on Systems, Man, and Cybernetics (SMC'99). Tokyo, Japan, 12.-15. Oktober. Piscataway, NJ, USA: IEEE, S. 534–539.

T'kindt, V. und Billaut, J.-C. (2006): Multicriteria scheduling. Theory, models and algorithms. 2nd ed. Berlin, New York: Springer.

Tuncel, E., Zeid, A. und Kamarthi, S. (2014): Solving Large Scale Disassembly Line Balancing Problem with Uncertainty using Reinforcement Learning. In: *Journal of Intelligent Manufacturing* 25 (4), S. 647–659.

van der Ham, R. (2018): Salabim: Discrete Event Simulation and Animation in Python. In: *Journal of Open Source Software* 3 (27), S. 767–768.

van der Zee, D.-J. (2015): Family-based Dispatching with Parallel Machines. In: *International Journal of Production Research* 53 (19), S. 5837–5856.

van Hasselt, H., Guez, A. und Silver, D. (2016): Deep Reinforcement Learning with Double Q-learning. In: Schuurmans, Dale, Wellman, Michael (Hg.): Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI'16). Phoenix, AZ, USA, 12.-17. Februar. Palo Alto, CA, USA: AAAI Press (AAAI'16), S. 2094–2100.

van Wassenhove, L. N., Potts, C. N., Sevast'janov, S. V., Strusevich, V. A. und Zwaneveld, C. M. (1992): The two-stage assembly scheduling problem: complexity and approximation. INSEAD Working paper 92/53/TM.

VDI 3633 Blatt 1, 2014: Simulation von Logistik-, Materialfluß- und Produktionssystemen.

Wagner, H. M. (1959): An integer linear-programming model for machine scheduling. In: *Naval Research Logistics* 6 (2), S. 131–140.

Wang, G., Li, Q. und Wang, L. (2015): An Improved Cross Entropy Algorithm for Steelmaking-Continuous Casting Production Scheduling with Complicated Technological Routes. In: *Journal of Central South University* 22 (8), S. 2998–3007.

Wang, H., Sarker, B. R., Li, J. und Li, J. (2020): Adaptive Scheduling for Assembly Job Shop with Uncertain Assembly Times based on Dual Q-Learning. In: *International Journal of Production Research*, S. 1–17.

Wang, J., Qu, S., Wang, J., Leckie, J. O. und Xu, R. (2017): Real-Time Decision Support with Reinforcement Learning for Dynamic Flowshop Scheduling. In: Smart SysTech 2017. Proceedings of the 2017 European Conference on Smart Objects, Systems and Technologies. The 2017 European Conference on Smart Objects, Systems and Technologies (Smart SysTech 2017(München, Deutschland, 20.-21. Juni. Frankfurt am Main: IEEE Germany Section / VDE-Verlag, S. 1–9.

Wang, Y.-C. und Usher, J. M. (2004): Learning Policies for Single Machine Job Dispatching. In: *Robotics and Computer-Integrated Manufacturing* 20 (6), S. 553–562.

Wang, Y.-C. und Usher, J. M. (2005): Application of Reinforcement Learning for Agent-based Production Scheduling. In: *Engineering Applications of Artificial Intelligence* 18 (1), S. 73–82.

Waschneck, B., Reichstaller, A., Belzner, L., Altenmuller, T., Bauernhansl, T., Knapp, A. und Kyek, A. (2018a): Deep Reinforcement Learning for Semiconductor Production Scheduling. In: Procee-dings of the 29th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC 2018). The 29th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC 2018). Saratoga Springs, NY, USA, 30. April – 03. Mai. Piscataway, NJ, USA: IEEE, S. 301–306.

Waschneck, B., Reichstaller, A., Belzner, L., Altenmüller, T., Bauernhansl, T., Knapp, A. und Kyek, A. (2018b): Optimization of Global Production Scheduling with Deep Reinforcement Learning. In: *Procedia CIRP* 72, S. 1264–1269.

Watkins, C. J. C. (1989): Learning from Delayed Rewards. Ph.D. thesis. University of Cambridge, Cambridge, UK. King's College. Online verfügbar unter: http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf, zuletzt geprüft am 19th April 2020.

Watkins, C. J. C. H. und Dayan, P. (1992): Q-learning. In: *Machine Learning* 8 (3-4), S. 279–292.

Wei, Y.-Z., Jiang, X., Hao, P. und Gu, K. (2009a): Multi-agent Co-evolutionary Scheduling Approach Based on Genetic Reinforcement Learning. In: Proceedings of the 2009 Fifth

International Conference on Natural Computation (ICNC 2009). 2009 Fifth International Conference on Natural Computation (ICNC 2009). Tianjian, China, 14.-16. August. Piscataway, NJ, USA: IEEE, S. 573–577.

Wei, Y.-Z., Jiang, X., Hao, P. und Gu, K. (2009b): Pattern driven dynamic scheduling approach using reinforcement learning. In: Proceedings of the 2009 IEEE International Conference on Automation and Logistics (ICAL 2009). 2009 IEEE International Conference on Automation and Logistics (ICAL 2009). Shenyang, China, 05.-07. August. Piscataway, NJ, USA: IEEE, S. 514–519.

Wei, Y.-Z. und Zhao, M.-Y. (2005): A Reinforcement Learning-based Approach to Dynamic Job-shop Scheduling. In: *Acta Automatica Sinica* 31, S. 765–771.

Weicker, K. (2015): Evolutionäre Algorithmen. 3., überarb. und erw. Aufl. Wiesbaden: Springer Vieweg.

Whiteson, S. und Stone, P. (2006): Evolutionary Function Approximation for Reinforcement Learning. In: *Journal of Machine Learning Research* 7 (31), S. 877–917. Online verfügbar unter http://jmlr.org/papers/v7/whiteson06a.html.

Whitley, D., Starkweather, T. und Bogart, C. (1990): Genetic Algorithms and Neural Networks: Optimizing Connections and Connectivity. In: *Parallel Computing* 14 (3), S. 347–361.

Wiendahl, H.-P. und Wiendahl, H.-H. (2019): Betriebsorganisation für Ingenieure. 9., vollständig überarbeitete Auflage. München: Carl Hanser.

Wilde, T. und Hess, T. (2007): Forschungsmethoden der Wirtschaftsinformatik. In: *WIRTSCHAFTSINFORMATIK* 49 (4), S. 280–287.

Williams, R. J. (1988): Toward a Theory of Reinforcement-learning Connectionist Systems. Technical Report NU-CCS-88–3. Northeastern University, College of Computer Science.

Williams, R. J. (1992): Simple statistical gradient-following algorithms for connectionist reinforcement learning. In: *Machine Learning* 8 (3–4), S. 229–256.

Williams, R. J. und Baird III, L. C. (1993): Analysis of Some Incremental Variants of Policy Iteration: First Steps Toward Understanding Actor-Critic Learning Systems. Technical Report NU-CCS-93–11. Northeastern University, College of Computer Science.

Wirth, N. (1983): Algorithmen und Datenstrukturen. 3., überarbeitete Aufl. Stuttgart: Teubner.

Witten, I. H. (1977): An adaptive optimal controller for discrete-time Markov environments. In: *Information and Control* 34 (4), S. 286–295.

Xie, J., Gao, L., Peng, K., Li, X. und Li, H. (2019): Review on Flexible Job Shop Scheduling. In: *IET Collaborative Intelligent Manufacturing* 1 (3), S. 67–77.

Xue, T., Zeng, P. und Yu, H. (2018): A Reinforcement Learning Method for Multi-AGV Scheduling in Manufacturing. In: Proceedings of the 2018 IEEE International Conference on Industrial Technology (ICIT 2018). The 2018 IEEE International Conference on Industrial Technology (ICIT 2018). Lyon, Frankreich, 20.-22. Februar. Piscataway, NJ, USA: IEEE, S. 1557–1561.

Yang, J., Xu, H., Pan, L., Jia, P., Long, F. und Jie, M. (2011): Task Scheduling using Bayesian Optimization Algorithm for Heterogeneous Computing Environments. In: *Applied Soft Computing* 11 (4), S. 3297–3310.

Yu, T. und Zhu, H. (2020): Hyper-Parameter Optimization: A Review of Algorithms and Applications. Online verfügbar unter: https://arxiv.org/pdf/2003.05689.

Yu, Y., Qian, H. und Hu, Y.-Q. (2016): Derivative-Free Optimization via Classification. In: Schuurmans, Dale, Wellman, Michael (Hg.): Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI'16). Phoenix, AZ, USA, 12.-17. Februar. Palo Alto, CA, USA: AAAI Press (AAAI'16), S. 2286–2292.

Yuan, B., Jiang, Z. und Wang, L. (2016): Dynamic Parallel Machine Scheduling with Random Breakdowns using the Learning Agent. In: *International Journal of Services Operations and Informatics* 8 (2), Aufsatznummer 80083, S. 94.

Zäpfel, G. (1982): Produktionswirtschaft. Operatives Produktions-Management. Berlin: de Gruyter.

Zhang, B.-T. und Mühlenbein, H. (1993): Evolving Optimal Neural Networks Using Genetic Algorithms with Occam's Razor. In: *Complex Systems* 7 (3), S. 199–220.

Zhang, J., Ding, G., Zou, Y., Qin, S. und Fu, J. (2019): Review of Job Shop Scheduling Research and its New Perspectives under Industry 4.0. In: *Journal of Intelligent Manufacturing* 30 (4), S. 1809–1830.

Zhang, W. und Dietterich, T. G. (1995): A Reinforcement Learning Approach to Job-Shop Scheduling. In: Mellish, C. S. (Hg.): Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence. San Mateo, Calif.: Morgan Kaufmann, S. 1114–1120.

Zhang, Z., Wang, W., Zhong, S. und Hu, K. (2013): Flow Shop Scheduling with Reinforcement Learning. In: *Asia-Pacific Journal of Operational Research* 30 (05), S. 1350014.

Zhao, M., Li, X., Gao, L., Wang, L. und Xiao, M. (2018): An Improved Q-learning based Rescheduling Method for Flexible Job-Shops with Machine Failures. In: Proceedings of the 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE 2019). The 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE 2019). Vancouver, BC, Kanada, 22.-26. August. Piscataway, NJ, USA: IEEE, S. 331–337.

Zheng, S., Gupta, C. und Serita, S. (2019): Manufacturing Dispatching Using Reinforcement and Transfer Learning. In: Brefeld, U., Fromont, E., Hotho, A., Knobbe, A., Maathuis, M. und Robardet, C. (Hg.): Machine Learning and Knowledge Discovery in Databases, Bd. 2. The European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Database (ECML PKDD 2019). Würzburg, 16.-20. September. Cham: Springer International Publishing (11908), S. 655–671.

Zhou, L., Zhang, L. und Horn, B. K. (2020): Deep Reinforcement Learning-based Dynamic Scheduling in Smart Manufacturing. In: *Procedia CIRP* 93, S. 383–388.

Zhou, T., Tang, D., Zhu, H. und Wang, L. (2021): Reinforcement Learning With Composite Rewards for Production Scheduling in a Smart Factory. In: *IEEE Access* 9, S. 752–766.

Zimand, M. (2004): Computational complexity. A quantitative perspective. 1st ed. Amsterdam, Boston: Elsevier.

Zimmermann, G. (1988): Produktionsplanung variantenreicher Erzeugnisse mit EDV. Berlin, Heidelberg: Springer.

Zimmermann, H.-J. (2008): Operations Research. Methoden und Modelle ; für Wirtschaftsingenieure, Betriebswirte, Informatiker. 2., aktualisierte Aufl. Wiesbaden: Vieweg.