wbk
Institut für Produktionstechnik

Forschungsberichte aus dem
wbk Institut für Produktionstechnik
Karlsruher Institut für Technologie (KIT)

Andreas Kuhnle

Adaptive Order Dispatching based on Reinforcement Learning

Application in a Complex Job Shop in the Semiconductor Industry

Band 241

Forschungsberichte aus dem
wbk Institut für Produktionstechnik
Karlsruher Institut für Technologie (KIT)

Hrsg.: Prof. Dr.-Ing. Jürgen Fleischer
Prof. Dr.-Ing. Gisela Lanza
Prof. Dr.-Ing. habil. Volker Schulze

Andreas Hermann Kuhnle

Adaptive Order Dispatching based on Reinforcement Learning

Application in a Complex Job Shop in the Semiconductor Industry

Band 241

Adaptive Order Dispatching based on Reinforcement Learning
Application in a Complex Job Shop in the Semiconductor Industry

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften (Dr.-Ing.)

von der KIT-Fakultät für Maschinenbau des

Karlsruher Instituts für Technologie (KIT)

angenommene

Dissertation

von

M.Sc. Andreas Hermann Kuhnle

Tag der mündlichen Prüfung:	08.10.2020
Hauptreferentin:	Prof. Dr.-Ing. Gisela Lanza
Korreferenten:	Prof. Dr.-Ing. Michael Freitag
	Prof. Dr. rer. nat. Stefan Nickel

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the internet at http://dnb.d-nb.de.

Zugl.: Karlsruhe, Karlsruher Institut für Technologie, Diss., 2020

Printed in Germany.

ISBN 978-3-8440-7762-9
ISSN 0724-4967

Shaker Verlag GmbH • Am Langen Graben 15a • 52353 Düren
Phone: 0049/2421/99011-0 • Telefax: 0049/2421/99011-9
Internet: www.shaker.de • e-mail: info@shaker.de

wbk
Institut für Produktionstechnik

Vorwort der Herausgeber

Die schnelle und effiziente Umsetzung innovativer Technologien wird vor dem Hintergrund der Globalisierung der Wirtschaft der entscheidende Wirtschaftsfaktor für produzierende Unternehmen. Universitäten können als “Wertschöpfungspartner” einen wesentlichen Beitrag zur Wettbewerbsfähigkeit der Industrie leisten, in dem sie wissenschaftliche Grundlagen sowie neue Methoden und Technologien erarbeiten und aktiv den Umsetzungsprozess in die praktische Anwendung unterstützen.

Vor diesem Hintergrund soll im Rahmen dieser Schriftenreihe über aktuelle Forschungsergebnisse des Instituts für Produktionstechnik (wbk) des Karlsruher Instituts für Technologie (KIT) berichtet werden. Unsere Forschungsarbeiten beschäftigen sich sowohl mit der Leistungssteigerung von Fertigungsverfahren und zugehörigen Werkzeugmaschinen- und Handhabungstechnologien als auch mit der ganzheitlichen Betrachtung und Optimierung des gesamten Produktionssystems. Hierbei werden jeweils technologische wie auch organisatorische Aspekte betrachtet.

Prof. Dr.-Ing. Jürgen Fleischer
Prof. Dr.-Ing. Gisela Lanza
Prof. Dr.-Ing. habil. Volker Schulze

KIT
Karlsruher Institut für Technologie

Vorwort des Verfassers

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am wbk Institut für Produktionstechnik des Karlsruher Instituts für Technologie (KIT).

Mein besonderer Dank gilt Frau Prof. Dr.-Ing. Gisela Lanza für die Betreuung meiner wissenschaftlichen Arbeit als Hauptreferentin, die persönliche Unterstützung sowie Förderung, das mir entgegengebrachte Vertrauen und die Freiheiten in den vergangenen Jahren, welche weit über diese Arbeit hinausgingen. Weiter danke ich Herrn Prof. Dr.-Ing. Michael Freitag und Herrn Prof. Dr. rer. nat. Stefan Nickel für das Interesse an meiner Arbeit und die Übernahme des Korreferats sowie Herrn Prof. Dr.-Ing. Wolfgang Seemann für den Prüfungsvorsitz.

Dem Karlsruher House of Young Scientists danke ich für die Förderung meines Forschungsaufenthaltes am Massachusetts Institute of Technology. Ebenso gilt mein Dank Prof. Stephen C. Graves für die herzliche Gastfreundschaft und die wissenschaftlichen Diskussionen.

Allen Kolleginnen und Kollegen des wbk in Karlsruhe, insbesondere im Bereich PRO, danke ich für die fortwährende Unterstützung in den vergangenen Jahren. Namentlich seien an dieser Stelle Christoph Liebrecht, Stefan Treber, Niclas Eschner, Constantin Hofmann, Tom Stähr, Robin Kopf, Sebastian Greinacher und Nicole Stricker erwähnt. Unsere gemeinsamen Unternehmungen bleiben mir lange in Erinnerung. Für das Korrekturlesen dieser Arbeit danke ich Carmen Krahe und Marvin May. Darüber hinaus, gilt mein besonderer Dank Andreas Fischer, Johannes Jakubik, Nicole Röhrig, Leonard Overbeck, Henri Felsenstein, Louis Schäfer, Kathrina Traub, Katharina König, Felix Hettich, Jan-Philipp Kaiser, Marvin May, Felix Theiss, Sebastian Behrendt, Patrick Haberkern, Tillmann Stüker, Alexander Valet und Alexander Heilig, welche durch ihre Arbeit sowie zahlreiche Diskussionen einen deutlichen Beitrag zum Gelingen der Arbeit geleistet haben.

Den Kollegen der Infineon Technologies AG aus Regensburg und München möchte ich an dieser Stelle ebenfalls herzlich für die tolle und vertrauensvolle Zusammenarbeit danken.

Abschließend möchte ich mich bei meinen Eltern, Geschwistern und Ann-Kathrin dafür bedanken, dass sie mir zu jeder Zeit den nötigen Rückhalt und stets wertvolle Unterstützung gegeben haben.

Karlsruhe, 20.10.2020

Andreas Hermann Kuhnle

Kurzfassung

Heutige Produktionssysteme tendieren durch die Marktanforderungen getrieben zu immer kleineren Losgrößen, höherer Produktvielfalt und größerer Komplexität der Materialflusssysteme. Diese Entwicklungen stellen bestehende Produktionssteuerungsmethoden in Frage. Im Zuge der Digitalisierung bieten datenbasierte Algorithmen des maschinellen Lernens einen alternativen Ansatz zur Optimierung von Produktionsabläufen. Aktuelle Forschungsergebnisse zeigen eine hohe Leistungsfähigkeit von Verfahren des Reinforcement Learning (RL) in einem breiten Anwendungsspektrum. Im Bereich der Produktionssteuerung haben sich jedoch bisher nur wenige Autoren damit befasst. Eine umfassende Untersuchung verschiedener RL-Ansätze sowie eine Anwendung in der Praxis wurden noch nicht durchgeführt.

Unter den Aufgaben der Produktionsplanung und -steuerung gewährleistet die Auftragssteuerung (*order dispatching*) eine hohe Leistungsfähigkeit und Flexibilität der Produktionsabläufe, um eine hohe Kapazitätsauslastung und kurze Durchlaufzeiten zu erreichen. Motiviert durch komplexe Werkstattfertigungssysteme, wie sie in der Halbleiterindustrie zu finden sind, schließt diese Arbeit die Forschungslücke und befasst sich mit der Anwendung von RL für eine adaptive Auftragssteuerung. Die Einbeziehung realer Systemdaten ermöglicht eine genauere Erfassung des Systemverhaltens als statische Heuristiken oder mathematische Optimierungsverfahren. Zusätzlich wird der manuelle Aufwand reduziert, indem auf die Inferenzfähigkeiten des RL zurückgegriffen wird.

Die vorgestellte Methodik fokussiert die Modellierung und Implementierung von RL-Agenten als Dispatching-Entscheidungseinheit. Bekannte Herausforderungen der RL-Modellierung in Bezug auf Zustand, Aktion und Belohnungsfunktion werden untersucht. Die Modellierungsalternativen werden auf der Grundlage von zwei realen Produktionsszenarien eines Halbleiterherstellers analysiert. Die Ergebnisse zeigen, dass RL-Agenten adaptive Steuerungsstrategien erlernen können und bestehende regelbasierte Benchmarkheuristiken übertreffen. Die Erweiterung der Zustandsrepräsentation verbessert die Leistung deutlich, wenn ein Zusammenhang mit den Belohnungszielen besteht. Die Belohnung kann so gestaltet werden, dass sie die Optimierung mehrerer Zielgrößen ermöglicht. Schließlich erreichen spezifische RL-Agenten-Konfigurationen nicht nur eine hohe Leistung in einem Szenario, sondern weisen eine Robustheit bei sich ändernden Systemeigenschaften auf.

Damit stellt die Forschungsarbeit einen wesentlichen Beitrag in Richtung selbstoptimierender und autonomer Produktionssysteme dar. Produktionsingenieure müssen das Potenzial datenbasierter, lernender Verfahren bewerten, um in Bezug auf Flexibilität wettbewerbsfähig zu bleiben und gleichzeitig den Aufwand für den Entwurf, den Betrieb und die Überwachung von Produktionssteuerungssystemen in einem vernünftigen Gleichgewicht zu halten.

Contents

Abbreviations

Abbreviation	Description
ANN	Artificial Neural Network
ATC	Apparent Tardiness Cost
CI	Confidence Interval
CNN	Convolutional Neural Network
CV	Coefficient of Variation
DQN	Deep-Q Network
DU	Distance Unit (arbitrary)
EDD	Earliest Due Date
FIFO	First In First Out
FLNQ	Fewest Lots in the Next Queue
KPI	Key Performance Indicator
LWT	Longest Waiting Time
MDP	Markov Decision Process
ML	Machine Learning
MLP	Multi-Layer Perceptron
MS	Minimum Slack
MTBF	Mean Time Between Failure
MTOL	Mean Time Off Line
NJF	Nearest Job First
OEE	Overall Equipment Effectiveness
PPC	Production Planning and Control
POMDP	Partially Observable Markov Decision Process
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
SARSA	State, Action, Reward, State, and Action
SPT	Shortest Processing Time
SRPT	Shortest Remaining Processing Time
SST	Shortest Setup Time
TD	Temporal-Difference
TRPO	Trust Region Policy Optimization
TU	Time Unit (arbitrary)
WIP	Work In Process
WSPT	Weighted Shortest Processing Time

Symbol	Description	Unit	Value range
∇	Differential operator	-	-
$\varnothing$	Empty set	-	-
α	Learning rate	-	$[0,1]$
$\alpha\text{-}value$	Alpha value according to the theory of operating curves	-	$\mathbb{R}^{\geq 0}$
$\mathcal{A}$	Set of all possible actions	-	-
$A(s)$	Set of all possible actions in state s	-	$\mathcal{A}$
a, a'	Action, next action	-	$\mathcal{A}$
A_{empty}	Action subset of all actions for moving without dispatching an order	-	$\mathcal{A}$
A_{idle}	Action subset of all actions for waiting and idling for a certain period	-	$\mathcal{A}$
$A_{M \rightarrow S}$	Action subset of all actions for dispatching an order from a machine to a sink resource	-	$\mathcal{A}$
$A_{O \rightarrow D}$	Definition of a dispatching action from resource O to resource D	-	$\mathcal{A}$
$A_{S \rightarrow M}$	Action subset of all actions for dispatching an order from a source to a machine resource	-	$\mathcal{A}$
$A_{valid,t}$	Action subset of all actions that are valid at time t	-	$\mathcal{A}$
a_j	Attribute of order j used for priority index determination	-	$\mathbb{R}$
A_t	Action at time t	-	$\mathcal{A}$
APT_i	Average processing time for machine i	TU	$\mathbb{R}^{\geq 0}$
at_{max}	Highest possible time to perform an action within the system	TU	$\mathbb{R}^{\geq 0}$
at_i	State variable indicating the time it takes to perform a specific action a_i	-	$[0,1]$
ben_i	State variable indicating the relative inbound buffer availability at machine i	-	$[0,1]$
bex_i	State variable indicating the relative outbound buffer availability at machine i	-	$[0,1]$
bpt_i	State variable indicating the total processing time at machine i	-	$\mathbb{R}^{\geq 0}$
γ	MDP discount rate for future rewards	-	$[0,1]$
c_j	Completion time of order j	-	$\mathbb{R}^{\geq 0}$
CAP_i^{EN}	Capacity of inbound buffer at machine i	-	$\mathbb{N}^{\geq 0}$

Symbol	Description	Unit	Value range
CAP_i^{EX}	Capacity of outbound buffer at machine i	-	$\mathbb{N}^{\geq 0}$
CT_j	Cycle time of order j	TU	$\mathbb{R}^{\geq 0}$
$d_{i,j}$	Transportation distance from resource i to resource j	DU	$\mathbb{R}^{\geq 0}$
d_j	Planned completion time of order j	-	$\mathbb{R}^{\geq 0}$
DT_i	Down time of machine i	TU	$\mathbb{R}^{\geq 0}$
ϵ	Probability of ϵ-greedy action selection	-	$[0,1]$
ϵ_{conv}	Threshold for convergence of the RL-agent	-	$\mathbb{R}^{\geq 0}$
$E(\bullet)$	Expected value function, i.e. arithmetic mean	-	$\mathbb{R}$
$f(x)$	Objective function of an optimization problem	-	$\mathbb{R}$
$f(Z)$	Activation function of a perceptron	-	$\mathbb{R}$
f_{aggr}	Aggregation function of the tree-based reward	-	$\mathbb{R}$
G_t	Expected discounted reward at time t	-	$\mathbb{R}$
$h(s,a,\theta)$	Parameterized numerical preferences	-	$\mathbb{R}$
θ	Policy parameter vector	-	$\mathbb{R}^{\|\theta\|}$
i	Multi-purpose index used in several contexts, if just a single index is required	-	-
I	Number of orders, inventory inside the manufacturing system	-	$\mathbb{N}^{\geq 0}$
I_j	Priority rule index of order j	-	$\mathbb{N}^{\geq 0}$
j	Production order index (job, lot, batch), J total number of orders	-	$\{1,...,J\}$
$J(\theta)$	Scalar performance measure of policy parameters θ	-	$\mathbb{R}$
$\widehat{J(\theta)}$	Estimate of the scalar performance measure of policy parameters θ	-	$\mathbb{R}$
k	Operation index, k_j number of all operations of order j	-	$\{1,...,k_j\}$
λ	Rate parameter of the exponential distribution	-	$\mathbb{R}^{>0}$
l_i	State variable indicating whether the dispatching resource's is located at resource i	-	$\{0,1\}$
L_j	Lateness of order j	TU	$\mathbb{R}^{\geq 0}$
μ	Mean time of a stochastic process	TU	$\mathbb{R}^{\geq 0}$
M	Machine resources	-	-
MA	Maintenance resource, n_{MA} number of maintenance workers	-	-

Symbol	Description	Unit	Value range
mf_i	State variable indicating the machine availability at machine i	-	$\{0,1\}$
MT_i	Manufacturing time of machine i	TU	$\mathbb{R}^{\geq 0}$
n	Machine index (production resource), N total number of machines	-	$\{1,...,N\}$
OCC_i^{EN}	Number of occupied inbound buffer slots at machine i	-	$\mathbb{N}^{\geq 0}$
OCC_i^{EX}	Number of occupied outbound buffer slots at machine i	-	$\mathbb{N}^{\geq 0}$
$\pi_t(a \mid s)$	Probability according to policy π_t of $A_t = a$ if $S_t = s$	-	$[0,1]$
$\pi_\theta(a \mid s, \theta)$	Parameterized policy π based on parameter vector θ	-	$\mathbb{R}$
$p(s', r \mid s, a)$	Transition probability function from state s and action a to s' and reward r	-	$[0,1]$
PT_i	Total processing time at machine i, i.e. sum of processing times of orders in the inbound buffer	TU	$\mathbb{R}^{\geq 0}$
PT_j	Processing time of order j	TU	$\mathbb{R}^{\geq 0}$
$\Phi(\bullet)$	Real-value function	-	$\mathbb{R}$
$q_\pi(s,a)$	State-action value function of state s and action a under policy π	-	$\mathbb{R}$
$\mathcal{R}$	Set of all fixed resources (sources, machines, and sinks)	-	-
$r(s,a,s')$	Expected reward function for the transition from state s and action a to s' (r_{sparse} sparse reward function, $r_{modelled}$ modelled reward function)	-	$\mathbb{R}$
R_t	Reward signal at time t	-	$\mathbb{R}$
$r_{const,ep}$	Episodic constant reward function	-	$\mathbb{R}$
r_{const}	Reward function with constant reward ω for each action subset	-	$\mathbb{R}$
r_{NJF^*}	Reward function with constant reward when the NJF*-action a_{NJF^*} is selected	-	$\mathbb{R}$
$r_{res}(\bullet)$	Combined reward function of multiple weighted functions	-	$\mathbb{R}$
r_{tree}	Reward function for the tree-based reward	-	$\mathbb{R}$
r_{util}	Reward function with utilization as reward value	-	$\mathbb{R}$
$r_{uti,ep}$	Episodic utilization reward function	-	$\mathbb{R}$

Symbol	Description	Unit	Value range
$r_{w,util}$	Reward function with utilization time rewarded differently per action subset	-	$\mathbb{R}$
r_{wt}	Reward function with waiting time as reward value	-	$\mathbb{R}$
$r_{w,wt}$	Reward function with waiting time rewarded differently per action subset	-	$\mathbb{R}$
$r_{wt,ep}$	Episodic waiting time reward function	-	$\mathbb{R}$
r_j	Order release time of order j	-	$\mathbb{R}^{\geq 0}$
$\overline{r_{i \to end}}$	Reward at the intersection of the moving mean reward and $\overline{r_{lq}}$	-	$\mathbb{R}$
$\overline{r_{lq}}$	Average reward received in the last quarter of an experiment run	-	$\mathbb{R}$
RPT_i	Remaining processing time at machine i	TU	$\mathbb{R}$
rpt_i	State variable indicating the remaining processing time at machine i	-	$\mathbb{R}$
ρ	Utilization of a process according to the queuing theory	-	$[0, 1]$
σ	Standard deviation of a stochastic process time	-	$\mathbb{R}^{\geq 0}$
SO	Source resources, n_{SO} number of sources	-	-
SI	Sink resources, n_{SI} number of sinks	-	-
$\mathcal{S}$	Set of all possible states	-	-
s, s'	State, next state	-	$\mathcal{S}$
S_t	State representation at time t	-	$\mathcal{S}$
S_T	Terminal state of an episode	-	$\mathcal{S}$
S_{AT}	State element containing variable at_i	-	-
S_{BEN}	State element containing variable ben_i	-	-
S_{BEX}	State element containing variable bex_i	-	-
S_{BPT}	State element containing variable bpt_i	-	-
$S_{j,k}$	Sequence of k operations of order j	-	-
S_L	State element containing variable l_i	-	-
S_{MF}	State element containing variable mf_i	-	-
S_{RPT}	State element containing variable rpt_i	-	-
S_{VA}	State element containing variable va_i	-	-
S_{WT}	State element containing variable wt_i	-	-
ST_j	Setup time of order j	TU	$\mathbb{R}^{\geq 0}$
t	Discrete time step or point in time	-	$\mathbb{R}^{\geq 0}$
T	Time period	TU	$\mathbb{R}^{\geq 0}$

Symbol	Description	Unit	Value range
$t_{\to O}$	Time to move from the current location to the origin O of a dispatching action	TU	$\mathbb{R}^{\geq 0}$
t_{load}	Handling time to load a resource	TU	$\mathbb{R}^{\geq 0}$
$t_{O \to D}$	Time to perform dispatching from origin O to destination D	TU	$\mathbb{R}^{\geq 0}$
t_{unload}	Handling time to unload a resource	TU	$\mathbb{R}^{\geq 0}$
TP	Throughput of a manufacturing system	$\frac{\mathrm{orders}}{\mathrm{TU}}$	$\mathbb{R}^{\geq 0}$
TR	Transportation resources, n_{TR} number of transportation resources	-	-
TT_j	Transport time of order j	TU	$\mathbb{R}^{\geq 0}$
U	Average manufacturing uptime utilization of all machines	-	$[0, 1]$
U_{disp}	Utilization of the order dispatching resource	-	$[0, 1]$
U_i	Manufacturing uptime utilization of machine i	-	$[0, 1]$
v	Speed of the dispatching resource	$\frac{\mathrm{DU}}{\mathrm{TU}}$	$\mathbb{R}^{\geq 0}$
$v_\pi(s)$	State value function of state s under policy π	-	$\mathbb{R}$
va_i	State variable indicating valid actions	-	$\{0, 1\}$
W	Weight vector of a perceptron	-	$\mathbb{R}^{\|W\|}$
w_j	Weight of order j	-	$\mathbb{R}^{\geq 0}$
WT_i^{max}	Longest waiting time of an order at resource i	TU	$\mathbb{R}^{\geq 0}$
WT_i^{mean}	Mean waiting time at resource i	TU	$\mathbb{R}^{\geq 0}$
WT_i^{std}	Standard deviation of the waiting time at resource i	-	$\mathbb{R}^{\geq 0}$
wt_i	State variable indicating the waiting time at source or machine i	-	$\mathbb{R}^{\geq 0}$
WT_j	Waiting time of order j	TU	$\mathbb{R}^{\geq 0}$
$x \in \Omega$	Set of feasible solutions in an optimization problem	-	-
X	Input vector of a perceptron	-	$\mathbb{R}^{\|X\|}$
$x_{j,k,n}$	Assignment variable indicating if operation k of order j is assigned on machine n	-	$\{0, 1\}$
ξ	Initial state distribution	-	-
Y_{dist}	Distance-based evaluation of the tree-based reward	-	$\mathbb{R}^{\geq 0}$
Y_l^i	Evaluation of node i in layer l of the tree-based reward	-	$\mathbb{R}^{\geq 0}$
Y_{util}	Utilization-based evaluation of the tree-based reward	-	$\mathbb{R}^{\geq 0}$
$Z(X, W)$	Perceptron state based on X and W	-	$\mathbb{R}$

1 Introduction

Manufacturing companies are confronted with ever-rising market demand and customer requirements. First, the trend of individualization leads to smaller batch sizes (Koren 2010; Abele & Reinhart 2011). Second, increasing product quality requirements result in specialized and heterogeneous machinery and production operations (Mönch & Fowler et al. 2013). Hence, manufacturing systems as a whole are characterized as highly dynamic systems with a non-linear interplay of stochastic processes (Freitag 2005).

At the same time, the digitization and technological progression in the era of the fourth industrial revolution is observed as significant enabler (Bauernhansl & Hompel et al. 2014; Lasi & Fettke et al. 2014). Regarding production systems, this manifests itself primarily in a comprehensive collection of real-time data that tracks and provides information about the condition of products, machines, and material handling systems. The collected data serves as a basis for an in-depth analysis and optimization of the system (Bauernhansl & Krüger et al. 2016; Henke & Bughin et al. 2016). A broad and more accurate database combined with powerful computing capacity and advanced algorithms are seen as revolutionary driving forces in operations management (Schuh & Reuter et al. 2017; Chui & Manyika et al. 2018).

One frequently discussed approach is data-driven reinforcement learning (RL), which already showed superior performance in popular board games such as Chess or Go (Silver & Hubert et al. 2018). However, board games represent artificial applications with a well-defined system scope and set of rules, describing the system behaviour. These assumptions are hardly given for real-world production operations.

1.1 Motivation

In the manufacturing industry, production control systems are considered as the key enabler for stable and cost-efficient production operations (Lödding 2016). However, integrating advanced data-driven optimization techniques is not trivial and requires considerable effort (Schuh & Reuter et al. 2017) and does, most likely, not overcome all obstacles such as the dilemma of contradicting objectives (Gutenberg 1951).

From a research perspective, there are two directions that incorporate data-driven approaches: On the one hand, further development of planning methods for the design of production systems, towards agile production systems (Greschke 2016; Bochmann 2018) and robust planning under consideration of uncertainty (Pinedo 2016; Echsler Minguillon 2020). On the other hand, the research of production control systems, which is characterized by a decentralized control architecture with autonomous decision agents (Gabel & Riedmiller 2012;

Waschneck & Altenmüller et al. 2018a). The present work focuses on the latter research direction.

If one considers established methods for real-time decision-making, quantitative methods such as mathematical optimization and (meta-) heuristics can only partially meet the challenges outlined above. Large problem instances cannot be solved with mathematical optimization approaches with acceptable computational effort and heuristics, which are static and rule-based, simplify the problem too far (Nickel & Stein et al. 2014). Moreover, both are so-called model-based solution approaches which require extensive domain knowledge and information about the production system characteristics (Waschneck & Altenmüller et al. 2016). Finally, the effort to maintain mathematical models for planning systems is very high, as they are often incorrect and outdated (Lödding 2016).

Autonomous systems, i.e. the implementation of adaptivity and learning ability, resemble the essential strengths of human intelligence (Kagermann & Gaus et al. 2018). They allow the design of advanced adaptive control methods (Russell & Norvig 2016). According to Åström & Wittenmark (2008) and Niehues (2016), ***adaptivity*** refers to the ability of a system to recognize deviations from the planned process at an early stage during operation and to independently derive and implement necessary (counter-) actions (see Figure 1.1).

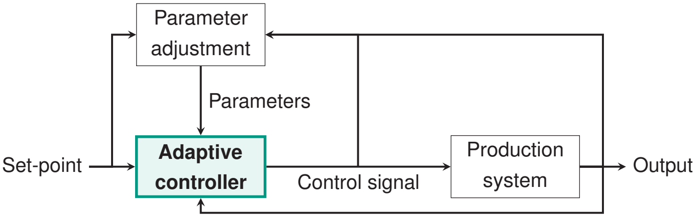


Figure 1.1: Adaptive controller for production systems (adapted from Åström & Wittenmark (2008)).

The learning ability of autonomous systems ensures that the system adapts and reinforces according to changing environmental conditions by combining (initial) knowledge with real-time data (Russell & Norvig 2016). Hence, autonomous and self-optimizing production systems show the same characteristics (Permin & Bertelsmeier et al. 2016). They overcome the manual task of designing a controller, which, in general, is a demanding job, covering in-depth process analysis, mathematical model building, and derivation of control rules (Hafner & Riedmiller 2011). The central discipline in the development of autonomous systems is machine learning, which provides various tools for this purpose, e.g. reinforcement learning (Sutton & Barto 2018). Reinforcement learning overcomes the controller design process by interacting directly with the system in a closed-loop environment. There are still just a few empirical applications for RL-based production control, as the literature review shows in the following. So, the full potential of data-driven approaches in production control is not yet realized.

1.2 Problem statement

Initially, this work is motivated by the challenges and characteristics of the semiconductor industry. However, the scope of applications exceeds this industry and also covers other industries with similar characteristics as elaborated in the following problem statement.

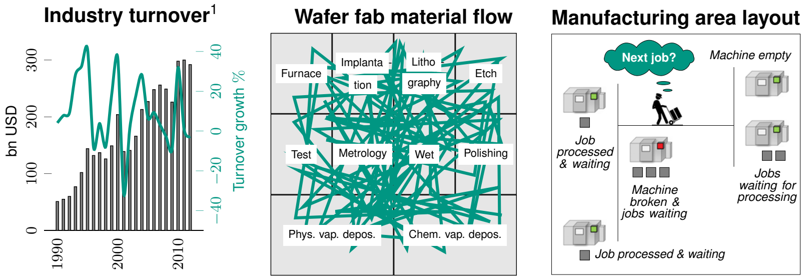


Figure 1.2: Problem statement – complex job shop in the semiconductor industry (adapted from Hilsenbeck (2005), Mönch & Fowler et al. (2013), and Ziarnetzky (2017)).

The current state and, hence, the problem statement addressed by this research is summarized in Figure 1.2 and follows three stages. First, as a basis for a holistic view of the problem, an analysis of the industry environment is necessary. The semiconductor industry has traditionally been subjected to fast innovation cycles and a high level of capital investment, e.g. machine equipment and clean room environment (Mönch & Fowler et al. 2013). Over the last years, the turnover increased considerably, however, not continuously. The percentage growth rate varies from below -30% to over 30% and sometimes the growth rate changes by a margin of 50% from one year to the other (see Figure 1.2 on the left). Moreover, according to Meier (2019), there is a strong bullwhip effect that causes substantial variations in the entire semiconductor supply chain. This presents a particular challenge for companies whenever facing any long-term decision, such as investments in machine capacities.

Second, a semiconductor factory (wafer fab) is operated 24 hours and 365 days a year to amortize the high capital investment. The wafers are combined into lots of 25 and sometimes

[1] Raman Chitkara et al. (2013), *Spotlight on Automotive – PwC Semiconductor Report*. `https://www.pwc.at/de/images/tmt-studie-2.pdf` (accessed on 17.06.2020).

50 (Ziarnetzky 2017). A lot takes up to three months to go through all manufacturing processes, of which around three-quarters are spent on wafer fabrication (*frontend*) and just one-quarter on packaging and logistics (*backend*). A total of up to 800 operations are carried out on different machines with processing times between 15 minutes and 12 hours (Ziarnetzky 2017). Moreover, flexible production systems are required since many product types are manufactured with varying process steps. High cost of capital means that space has to be used economically. Flow line production, which is commonly used in the automotive industry, is therefore not suitable. Instead, the machines are arranged according to the job shop principle. Disadvantages of this layout are potentially long distances and the fact that the material flow becomes complex and hard to control (see a schematic illustration of the material flow for one lot in the middle of Figure 1.2, adapted from Ziarnetzky (2017)).

Third, looking at a specific manufacturing area, the challenges just outlined decompose on the operational level, where the best operational performance is aimed by controlling the material flow and order processing (see Figure 1.2 on the right). Here, most companies are currently using rule-based real-time dispatching systems (Mönch & Fowler et al. 2013). These rules are designed by experienced production engineers and reflect process-specific requirements. However, long-term observations have shown that these systems are sometimes too rigid to consider the dynamics as well as manifold objectives and, therefore, a permanent adjustment effort is required (Waschneck & Altenmüller et al. 2018a). Changes such as layout rearrangement or product mix deviations are not foreseen and, hence, jam situations occur and the operational performance drops.

1.3 Research goal

The problem statement just outlined is addressed by the following overall research goal:

Adaptive order dispatching of complex job shop manufacturing systems based on reinforcement learning.

The research focuses on modelling, implementation, and evaluation of reinforcement learning. A real-world use case by the example of a job shop manufacturing system of a semiconductor company is chosen for evaluation and demonstration. Four research questions are derived from the research goal, which will be considered integrally and answered in the remainder of this research work:

1. **RL-applicability:** Is it possible to obtain an adaptive order dispatching system autonomously based on real-time operational data?

2. **Multi-objective optimization:** Do the modelling design choices of reinforcement learning extensively cover a multi-objective solution space, comparing to heuristic benchmarks?

3. **Transferability:** Does the learning-based approach show a superior performance in terms of changing system characteristics in comparison to existing benchmarks?

4. **User acceptance:** Is the acceptance of practitioners supported by a plausibility analysis of the RL-based control policy?

Additionally, the research is striving for the underlying idea that any production system can be described in a structured way, comprising all decision-relevant information as well as essential system characteristics. Based on this input, a digital representation of the real production system can be created (*digital twin*) to obtain an extensive database for the training of RL-algorithms. After that, the trained algorithm is able to perform the control task in the real-world system. Thereby, as little as necessary human effort is needed to create the control system. Furthermore, the algorithm is not bound to modelled assumptions, such as for mathematical optimization, but directly integrated into the simulated real-world system, which considers all stochastic processes.

1.4 Structure of this work

Having motivated the problem statement and research goal in Chapter 1, this work is structured as follows. The next part starts with the descriptive definition of the problem features. For this purpose, order dispatching is placed in the context of production planning and control (PPC) in Chapter 2. After comparing different quantitative optimization methods, the basics of reinforcement learning are outlined and its suitability is assessed. The fundamentals conclude with research focus areas in Chapter 3. Subsequently, the state-of-the-art literature review derives according to those areas the research deficit that is addressed. The methodology and analytical model for an adaptive order dispatching based on reinforcement learning are developed in four steps in Chapter 4. First, the problem scope is specified and analysed. Next, a production system model is developed (digital twin). Then the adaptive RL-algorithm is described and, finally, the performance evaluation criteria and experiment design are determined. Chapter 5 assesses the suitability of the RL-based order dispatching by outlining two use case scenarios and deductively evaluating the system's performance based on a comparison with conventional heuristic approaches. Moreover, the requirements for applying the developed approach in a real-world setup are outlined. In a final examination in Chapter 6, the results are discussed and an outlook is given to direct future research. Finally, Chapter 7 summarizes the entire work.

2 Fundamentals

This chapter focuses on the fundamentals of production planning and control (PPC, see Figure 2.1). Starting with a characterization of production systems that is relevant for the second section, giving an overview on and categorization of production planning and control tasks (e.g. order dispatching) and methods (e.g. heuristics). As the remainder of this work focuses on order dispatching, the focus lies on the two domains production and logistics and their close interaction. Using the example of semiconductor manufacturing, the properties of modern, complex manufacturing systems are explained. The second part of this chapter outlines alternative quantitative optimization methods that are well-established and applied in industry. Finally, the basics of reinforcement learning are introduced.

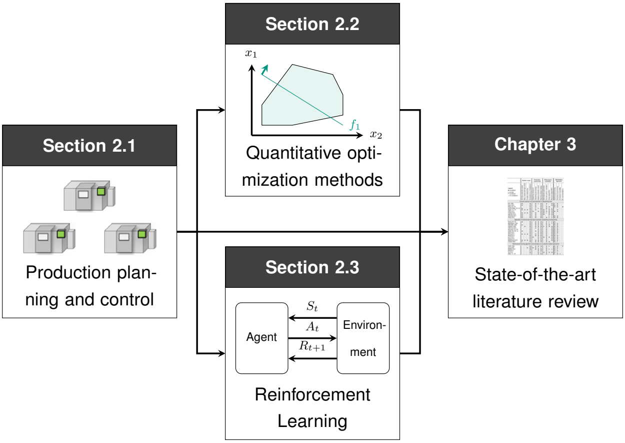


Figure 2.1: Structure of the fundamentals and the state-of-the-art literature review.

2.1 Production planning and control

The term production system is defined as the aggregation of all business, technological, and organizational activities that relate to the processing of material (Eversheim & Schuh 1996). Thereby, the term ensembles various facets that are subdivided into seven *levels* (see Figure 2.2 on the left). Firstly, from a hierarchical point of view, production systems cover the following spectrum (Wiendahl & ElMaraghy et al. 2007): network, site, segment, system, cell, station, and process. The present work sets the scope from production site level down to a system level.

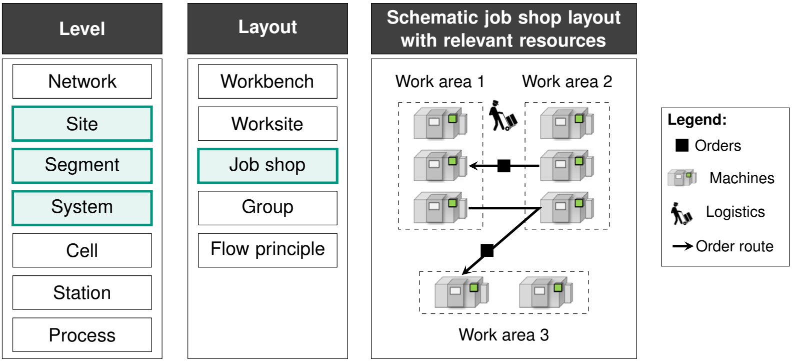


Figure 2.2: Production system level, layout alternatives, and relevant resources that are considered in the job shop system (adapted from Eversheim & Schuh (1996) and Wiendahl (1997)). The scope of the present work is highlighted.

Looking at the system level, there are five principles regarding the *layout* of production resources (Wiendahl 1997): workbench, worksite, job shop, group, and flow principle. Whereas the automotive assembly is mainly organized as highly efficient flow assembly lines, job shops are prevalent in semiconductor manufacturing. Job shops offer higher flexibility due to the degree of freedom within the material flow and tact-freeness, but, at the same time, require additional control effort. According to Heger (2014), semiconductor manufacturing is also categorized as flexible flow shop, when product routes are reasonably similar.

Besides the layout, the batch size and the repetition frequency of the same production order determine the following four production *types* (Aggteleky 1987): individual, small-scale, series, and mass production. In general, the sequence of production layout principles and production types are identical, meaning that a workbench comes along with small batch sizes and individual or small-scale production and mass production is organized as flow principle. However, there are also exceptions in order to meet specific requirements of the market, product, or production processes. One example is semiconductor manufacturing, as outlined in the next section, that operates a mass production in a job shop layout organization because, usually, very large volumes are processed per product variant (Stegherr 2000).

Figure 2.2 also shows on the right a schematic job shop layout example, including relevant resources: firstly, machines that perform the production processes, secondly, logistic resources that connect machines, and, thirdly, the production orders[1] itself.

[1] Hereinafter, product, order, job, lot, or batch are used as synonyms.

2.1.1 Complex manufacturing systems

Nowadays, manufacturing systems are broadly seen as *complex systems* due to manifold unprecedented challenges such as globally spread manufacturing operations (e.g. number of partners, communication), volatile future development (e.g. dynamic changes), and uncertain influencing factors (e.g. limited information) (Wiendahl & Scholtissek 1994; Schuh & Monostori et al. 2008; Koren 2010; ElMaraghy & ElMaraghy et al. 2012). The number of product variants is a major driver of complexity within manufacturing systems, as scarce resources need to be managed efficiently for a widely spread product demand. This also links to the research area of how to handle mass customization (ElMaraghy & Azab et al. 2009; Koren 2010). However, before describing the complexity definition used in the present work, one can generally summarize three types of complexity drivers:

- *Problem complexity:* number of different decisions that need to be made, e.g. order release and order sequencing.
- *Non-linear interdependencies:* number of processes that are potentially stochastic and non-linearly related to each other.
- *Problem dimension:* number of entities, e.g. machines and orders, to be considered.

The technical term complexity was first introduced by Alan Turing in the middle of the 20th century to evaluate algorithmic procedures (Arora & Barak 2009). He subdivided the definition into time complexity, space complexity, and communication complexity. Later, the computational complexity was transferred to categorize optimization problems. This led to the theory of $\mathcal{NP}$-hardness and most problems in operations management are $\mathcal{NP}$-hard optimization problems (Garey & Johnson 1979). $\mathcal{NP}$-hardness means that the problem is not deterministically solvable in polynomial time. According to Garey & Johnson et al. (1976), determining a shortest-length schedule in an m-machine flow shop is $\mathcal{NP}$-complete for $m \geq 3$ and for an m-machine job shop for every $m \geq 2$. However, this technical complexity definition is not appropriate enough to describe the complexity of manufacturing systems.

The broader scientific definition of complexity differentiates the term from *complicatedness* and *chaos* (ElMaraghy & ElMaraghy et al. 2012): Complicated systems consist of many parts or sub-systems, whereas complex systems show uncertainty in addition. In chaotic systems, the internal relations are intractable and slightly changing initial conditions may show an entirely different outcome, whereas in complex systems the internal relations might be non-linear but still trackable.

In order to define the term complex system in the context of the present work, it is referred to the characteristics of semiconductor manufacturing and, in particular, the products itself that are, to a certain extent, unique. Semiconductor products are one of the world-leading high-tech products (Horn 2008). At the heart of the manufacturing processes is the wafer frontend fabrication, next to three further backend manufacturing steps: sort, assembly, and test (Uzsoy & Lee et al. 1992). The frontend fabrication builds the chip's product functions layer by layer onto the silicon wafer and is the most time-consuming and capital-intensive step (Mönch & Fowler et al. 2013). Moreover, the fabrication processes are required to operate close to technological and sometimes even physical limits, which eventually leads to an equally ranked level of complexity of the manufacturing system itself (Horn 2008; Mönch & Fowler et al. 2011; Bauernhansl & Schatz et al. 2014).

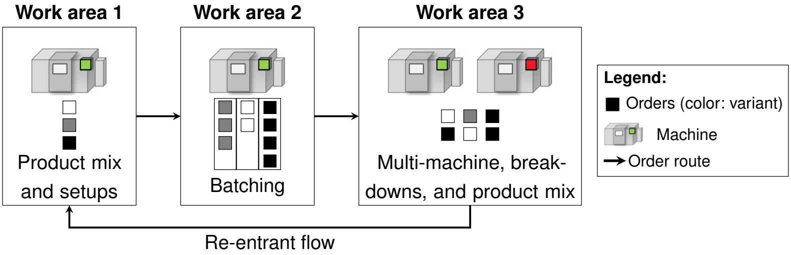


Figure 2.3: Exemplary illustration of characteristics of complex manufacturing system (adapted from Waschneck & Altenmüller et al. (2016)).

All in all, the manufacturing ***system complexity*** is defined according to the following categories taken from Mönch & Fowler et al. (2013) and illustrated in Figure 2.3:

1. Changing mix of product variants.
2. Multiple parallel, costly, and, in some cases, highly unreliable machines that require special maintenance actions.
3. Sequence-dependent setup times that can even exceed the processing time.
4. Tight customer due dates in a market where the product value degrades fast.
5. Various process types, ranging from batch processing (e.g. furnace) to single wafer processes (e.g. lithography).
6. Re-entrant material flow, meaning lots are processed on the same machine several times.

The first four properties are mostly addressed and well-studied by the production scheduling literature, i.e. job shop scheduling optimization problems focusing on tardiness, resulting in optimal setup sequences with the highest possible machine utilization (Graham & Lawler et al. 1979). In contrast, the last two properties mainly affect the material flow complexity and, usually, neglected in most production scheduling approaches.

According to Lödding (2016), the complexity of the material flow system depends on the number of predecessors and successors of a node, e.g. work centre, as well as the number of return flows. Moreover, when various process types, e.g. batching and single processing, are prevalent, both are self-amplifying effects and result in so-called *inventory waves*. A real-world example, taken from a semiconductor company, is depicted in Figure 2.4. Herein, the number of active transports varies with a spread of up to 186%, providing a particular challenge for material flow systems.

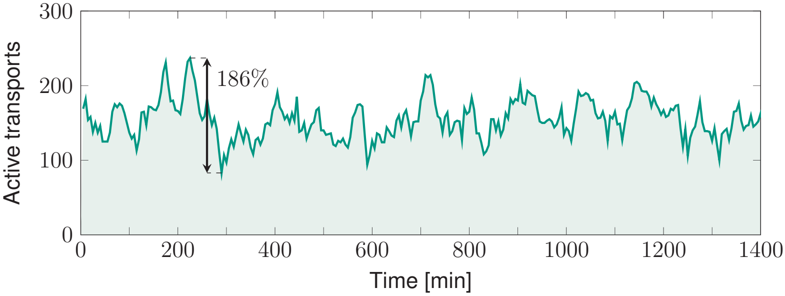


Figure 2.4: Real-world example of the number of active transports in a wafer frontend-fabrication (Lanza & Nyhuis et al. 2018).

Another industry example of production systems that show properties similar to the semiconductor industry are *matrix*-like structured production systems with flexible transportation links, e.g. autonomous guided vehicles (Greschke & Schönemann et al. 2014; Schönemann & Herrmann et al. 2015; Küpper & Sieben et al. 2018). Due to the rising number of product variants in the entire industrial sector and the challenges described above, traditional flow line systems show a poor performance and more and more manufacturers are investigating flexible (agile) production concepts (Echsler Minguillon 2020). Hence, the problem statement of the present research can be extended to any job shop-like system that are increasingly relevant to realize a cost-efficient mass-production of a large product portfolio. As such systems are rather new, adequate production control methods are not yet established.

Digitization in the semiconductor industry Driven by the complexity, semiconductor manufacturers started comparatively early with digitizing and automating initiatives of production resources and processes (Waschneck & Altenmüller et al. 2016). Hence, what was later framed by the term "Industrie 4.0" has already been present a few years earlier. One reason for that is that maximizing the yield of processes and operating close to physical limits requires statistical process control in nearly every machine equipment (Mönch & Fowler et al. 2011). Another reason is the complex material flow, resulting from the given job shop layout (Sturm 2006). Therefore, the entire factory and every production lot are equipped with a localization system that provides location information of any order at any time. This enhances the capabilities of material flow control methods by a more accurate database and supports the application of automated material handling systems (Mönch & Fowler et al. 2013). Hence, automated handling systems are broadly established and responsible for the material flow between work areas and inside a work area between the machines (Sturm 2006). An efficient material handling directly influences overall operational performance indicators such as cycle time and machine utilization (Lin & Wang et al. 2001). Moreover, data-driven production control systems can be applied for the complex job shops (Waschneck & Altenmüller et al. 2018a).

2.1.2 Logistical performance indicators

Performance indicators are used to measure the operational performance of manufacturing systems. The performance measures are, from now on, named *logistical performance indicators* (Lödding 2016). According to Nyhuis (2008), the importance of logistic operations changed through the years. Initially, logistics just focused on transportation, storage, and inventory handling. Due to the rising competition and customer orientation, logistics changed from functional business unit orientation to cross-functional process orientation, also including external supply chain partners (Chopra & Meindl 2019). The increasing integration towards global production networks leverages the importance of logistic operations to a key management focus (Lanza & Ferdows et al. 2019). Nowadays, production and logistics management are closely interlinked to fulfil customer demand on-time (service level) and to ensure a competitive market position (Lödding 2016; Chopra & Meindl 2019).

Wiendahl & Reichardt et al. (2014) describe the overall target of production operations as the least costly implementation of all processes that are relevant for goods' production. Thereby, all operations need to aim at the long-term goal of maximizing profitability. According to Wiendahl (1997), there are four ***logistical performance indicators*** to evaluate the performance: capacity utilization, order cycle time, adherence to due dates, and inventory level. Figure 2.5 illustrates these indicators according to Eversheim & Schuh (1996), and each indicator is defined as follows (Mönch & Fowler et al. 2013; Lödding 2016; VDI 4400 Part 2 2004):

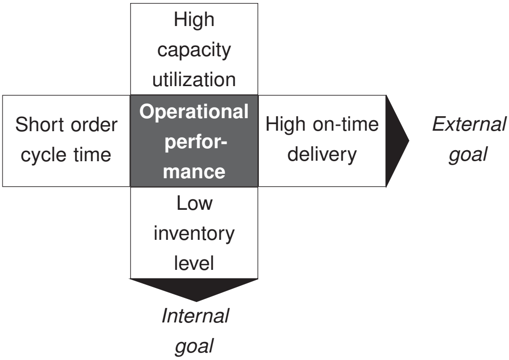


Figure 2.5: Logistical performance indicators (adapted from Eversheim & Schuh (1996)).

- **Cycle time** CT_j: Time period an order j is inside the manufacturing system, i.e. the time difference between completion c_j and order release r_j, or the sum of processing time PT_j, setup time ST_j, waiting time WT_j, and transportation time TT_j. Lödding (2016) emphasizes its importance, in particular for semiconductor manufacturers, as some companies calculate a surcharge of up to 200% for the shortest delivery time.

$$CT_j = c_j - r_j = PT_j + ST_j + WT_j + TT_j \tag{2.1}$$

- **Lateness** L_j: An order delay refers to the actual completion time c_j and the planned completion time d_j.

$$L_j = c_j - d_j \tag{2.2}$$

- **Utilization** U: The average manufacturing uptime utilization of N machines is defined for a period T based on the sum of manufacturing time MT_i without considering the total down time DT_i. Hence, it represents the ratio of used manufacturing time and total available machine capacity.

$$U = \frac{\sum_{i=1}^{N} MT_i}{\sum_{i=1}^{N}(T - DT_i) \cdot N} \tag{2.3}$$

- **Inventory** I: The inventory level, also called *work in process*, counts the number of orders inside the manufacturing system. This is the sum of all orders that are currently processed at a machine, transported, or waiting in a buffer.

The performance indicators can be split into two types: on the one hand, cycle time and lateness are directly related to orders and, on the other hand, utilization and inventory are related to resources (Domschke & Drexl et al. 2015). According to Figure 2.5, the order-related indicators are referred to as external goals, whereas utilization and inventory represent internal goals. Both types will be relevant in the present work.

Dilemma of contradicting objectives When optimizing the operational performance according to the just mentioned indicators, the dilemma of contradicting objectives becomes apparent. Generally speaking, the full realization of one objective can only be achieved at the expense of reaching another (Gutenberg 1951; Lödding 2016). The inventory level minimization of one product, for example, leads to a reduction in the utilization of specific resources. In particular, resource- and order-oriented performance measures are contradicting.

Two formulas describe the relationship between the performance indicators and point out the dilemma. On the one hand, Little (1961) introduced the basic queuing theory formula and, similarly, Bechte (1980) developed the so-called *"Trichtermodell"* to capture, in a stationary system, the relationship between the throughput TP of a system, the average inventory level I, and the average order cycle time CT (alternatively, the throughput can be replaced by the average arrival rate):

$$I = TP \cdot CT \tag{2.4}$$

So, according to *Little's Law*, inventory and cycle time are two indicators that support each other, i.e. they correlate positively. On the other hand, the *Kingman equation* approximates the expected waiting time $E(WT)$ of a $G/G/1$-queue with a given mean and standard deviation of the time between arrivals μ_a, σ_a and processing time μ_p, σ_p, a given utilization $\rho = \frac{\mu_a}{\mu_p}$, and c being the coefficient of variation (Kingman 1961):

$$E(WT) = \frac{\rho}{1-\rho} \cdot \frac{c_a^2 + c_p^2}{2} \cdot \mu_p \tag{2.5}$$

Two factors drive the waiting time: a high utilization (first term) and a high variation (second term) increase the queue length and, hence, the waiting time.

To overcome the dilemma of contradicting objectives, it is therefore essential to consider the interdependencies between different tasks and levels of production planning and control in a comprehensive view (Adam 1996; Nyhuis & Wiendahl 2012). Generally speaking, it is recommended to determine the most important strategic objective first and control the other performance indicators accordingly. This approach is considered in the following section on production planning and control tasks. But first, three further concepts are presented that relate to performance optimization and the dilemma of contradicting objectives. These are: production operating curves, pareto-optimality, and robustness.

Production operating curve Another important concept of PPC is the so-called *Produktionskennlinie* (eng., production operating curve) introduced by Nyhuis & Wiendahl (2012). The curves are used to visualize the course of logistical performance indicators as a function of the influencing variable. As a rule, the inventory level is used as independent influencing variable, since it is the easiest to regulate.

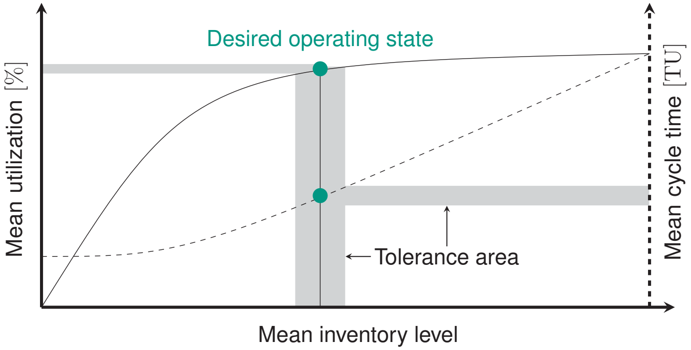


Figure 2.6: Production operating curve according to Nyhuis & Wiendahl (2012). The dashed line refers to the dashed axis.

Representative curves of the average utilization and the average cycle time (arbitrary time unit TU) are shown in Figure 2.6 (Nyhuis & Wiendahl 2012). It can be seen that higher inventory levels lead to higher utilization. At the same time, there are longer order waiting times, since the machine capacities are higher utilized when processing times are kept constant. The utilization decreases for less inventory, because the machines are more likely idling and waiting for orders. Consequently, the curves confirm that it is not possible to pursue both goals, low cycle times and high utilization.

Production operating curves emphasize that, due to the dependencies and dilemma of different objectives and influencing variables, a manufacturing system cannot be operated in any arbitrary state. The course of the curves is the same for any kind of production system. Therefore, the curves show the set of possible system operating states under the same boundary conditions. This approach is also called *logistic positioning* (Nyhuis & Wiendahl 2012).

Eventually, the curves are used to derive optimization measures if better operating states can be identified. With this method, under specification of all non-controllable boundary conditions, the desired operating state is determined by the performance measures inventory, utilization, and cycle time. The boundary conditions are, for example, the existing machines, layout, production orders, or external targets such as customer delivery times.

Pareto-optimality Further considerations of the dilemma of contradicting objectives, e.g. by Kacem & Hammadi et al. (2002), apply the concept of *Pareto-optimality* to find a set of efficient solutions for the multi-criteria optimization problem. A multi-criteria optimization problem is defined as follows:

$$\min_{x \in \Omega} f_1(x), f_2(x) \dots f_n(x) \tag{2.6}$$

Where x is a feasible solution of the optimization problem, Ω the set of all feasible solutions, and $f_i(\bullet)$ represents an objective function for a single optimization criterion. In general, there is no unique solution to such a multi-objective optimization problem (Domschke & Drexl et al. 2015). Therefore, solution x_A is called *Pareto-equivalent* to another solution x_B if they are comparable in terms of all objective functions. If solution x_C optimizes all objectives at least as well as x_A and at least one objective is better than x_A, x_C is called *Pareto-superior* to x_A. If there is no Pareto-superior solution to solution x_D, then x_D is *Pareto-optimal*. So, starting from a Pareto-optimal solution, it is not possible to improve one objective without negatively influencing another. Furthermore, all Pareto-optimal solutions yield a set of operating points that can be named “optimal”.

Robustness Finally, the concept of robustness is important in PPC as it depicts another performance characteristic of production systems. According to Stricker (2016), robustness evaluates the performance of a system under changing conditions and represents the compromise between a high and stable performance level. Due to the high degree of interdependencies between production processes, disturbances accumulate so that even small changes in the system parameters can have significant impact on the overall system performance (Prabhu & Duffie 1999; Freitag 2005).

In the present work, the concept of robustness is considered. Not only high performance in terms of one or multiple performance indicators is envisaged, but a robust performance is likewise relevant.

2.1.3 Production planning and control tasks

Generally speaking, there is a correlation between the complexity of production and material flow systems and the usage of production planning and control methods (Lödding 2016). PPC involves the planning, control, and administration of all processes necessary for the production of goods (Günther & Tempelmeier 2012). Thereby, it is in between and closely connected to the sales and purchasing functions of a company (Wiendahl 1997). It plays a vital role for any manufacturing company in operations management, optimizing its performance and usage of its production factors for the logistical performance indicators outlined in Section 2.1.2 (Schuh & Stich 2012).

According to Arnold & Isermann et al. (2008) and Eversheim (2002), the main tasks of PPC are the time-, quantity-, and capacity-related planning and control of all manufacturing operations. Hackstein (1984) presented an early reference model for PPC, and the most well-known model is called “Aachener PPS” introduced by Schuh & Stich (2012). The “Aachener PPS” categorizes all tasks into tasks related to the production network (e.g. network configuration, network sales planning, and network demand planning), the key tasks (e.g. production program planning, production demand planning, and control of external sourcing as well as internal sourcing), and cross-section tasks (e.g. controlling, order management, and inventory management). Concerning these categories, this work focuses on the control of internal sourcing.

However, more suitable for the remainder of this work are hierarchically structured models of PPC, such as the ones introduced by Switalski (1989), Mönch & Fowler et al. (2013), Wiendahl & Reichardt et al. (2014), and Kellner & Lienland et al. (2018). Herein, PPC decisions and activities are structured according to the time horizon as follows (see also Figure 2.7):

1. The result of the first *planning phase* is a production program that covers a period from months to a year. It contains the type, amount, and due date of products. The program is based on the available resource capacity as well as customer demand. The demand might also be a forecast for the planning period.

2. The production program is translated into production orders that are released based on an *order release policy*. When an order is released, the production operations start and a planned due date is defined for the respective product order.

3. *Material and capacity planning* is performed on a weekly or bi-weekly frequency. Detailed bill of material lists, current inventory levels, and operator work plans are used as input to determine order sequences and update due dates. These activities are also covered by the term *scheduling* (Graham & Lawler et al. 1979; Brucker & Knust 2012). A schedule

allocates (scarce) resources, such as machines or operators, over a period of time. It can range from a single resource to an entire system with multiple resources.

4. The *dispatching* phase starts after production orders are released and scheduling is performed. Hence, it mainly focuses on the control, implementation, and monitoring of operations with respect to the plan determined in the previous phase. A rescheduling might be performed in case of deviations from the initial plan (Vieira & Herrmann et al. 2003). Generally speaking, any dispatching decision needs to be taken in near real-time, i.e. seconds or minutes without disturbing the production process (Sturm 2006).

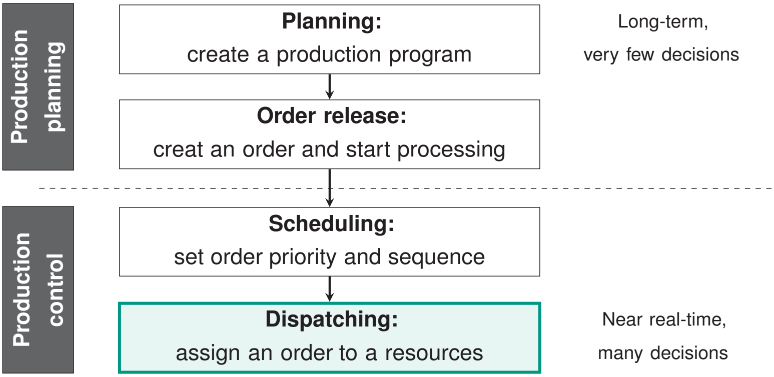


Figure 2.7: Production planning and control hierarchy (adapted from Mönch & Fowler et al. (2013)). The scope of the present work is highlighted.

Production planning, the first two phases, incorporates manifold decisions that are synchronized and covers a more extended period than production control. Production control, the last two phases, is of particular importance due to its direct influence on the performance indicators as any decision that is made will be implemented as such (Lödding 2016). Moreover, a wide range of activities is required in order to react to disturbances such as a machine breakdown, operator illness, or shortage of delivery (Nyhuis 2008).

Depending on the focal point where most decisions are made, it is either referred to as *central* (planning-focused) or *decentral* (control-focused) PPC organization (Scholz-Reiter & Freitag et al. 2005; Meissner & Ilsen et al. 2017).

Figure 2.8 gives an overview and classification of computational approaches to support the decision-making process in production control that can be found in academic literature according to Uzsoy & Lee et al. (1992) and Uzsoy & Lee et al. (1994). It differentiates between

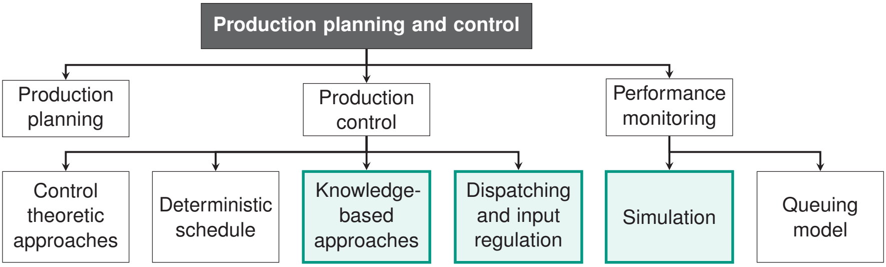

Production planning and control
Production planning
Production control
Performance monitoring
Control theoretic approaches
Deterministic schedule
Knowledge-based approaches
Dispatching and input regulation
Simulation
Queuing model

Figure 2.8: Computational approaches to support decision-making in production control (adapted from Uzsoy & Lee et al. (1992) and Uzsoy & Lee et al. (1994)). The focus of the present work is highlighted.

production planning, see above, production control, and performance monitoring that is used to understand the system behaviour better. This classification is supported by the summary of Nyhuis (2008). Further details on control theoretic approaches are found in Duffie (1996). The present work focuses on knowledge-based, dispatching, and simulation methods.

2.1.4 Order dispatching

This last sub-section focuses on order dispatching as the central task on the lowest level of the production control hierarchy (see Figure 2.7). If there is a planned schedule available, dispatching might just cover the implementation of the schedule. However, in many cases, there is no schedule or it is not up-to-date due to unforeseen disturbances. Then dispatching is of crucial importance to control the entire order process flow (Gudehus 2010). Moreover, Waschneck & Altenmüller et al. (2018b) show based on a real-world case study the relevance and importance of order dispatching as the compliance of shopfloor operations with predetermined schedules can be as low as 50%. As compliance reveals a gap between the planned and as-is operational performance, it poses a key issue in operations management.

Because there are multiple and sometimes contradicting definitions of order dispatching, the present work refers to the following definition, according to Lin & Wang et al. (2001) and Mönch & Fowler et al. (2013), which covers two tasks:

- *Dispatching of processing*: selection of the next order to be processed, i.e. assignment of an order to a processing resource.
- *Dispatching of transport*: selection of the next order to be transported, i.e. assignment of an order to a transportation resource.

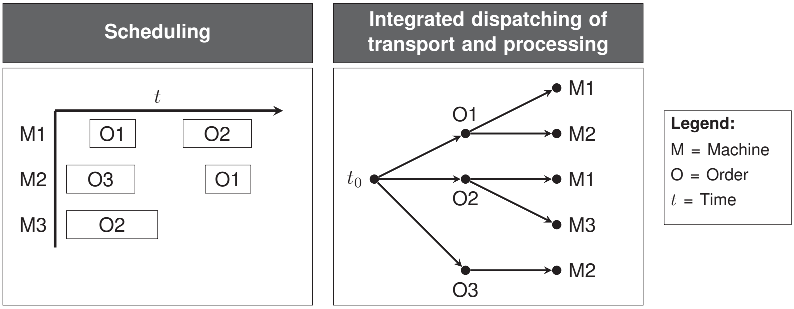


Figure 2.9: Difference between scheduling for a time period t (left) and integrated order dispatching at a point in time t_0 of transport and processing (right) according to a simplified example with three machines and three orders.

Dispatching decisions are made on the shopfloor either by machines or material handling systems, e.g. manual forklifts or automated guided vehicles. In contrast to the broad scheduling literature, e.g. by Graham & Lawler et al. (1979), order dispatching in the present work considers not only machines but also transportation resources. The research in the domain of integrated production and transportation optimization is rather new and not as well investigated (Knust 2000; Fazlollahtabar & Saidi-Mehrabad 2013).

Summarizing the just described scheduling and dispatching tasks, Figure 2.9 depicts the difference between both and describes the integrated order dispatching of transport and processing as it is used in the remainder of this work. For illustration, a simplified example with three machines and three orders is used. *Scheduling* considers a time horizon t (in the future), whereas *dispatching* is a decision at a specific point in time t_0 (Mönch & Fowler et al. 2013). Hence, a schedule assigns the three orders to the three machines according to their work plan (Figure 2.9 on the left). A dispatching decision refers to a single action out of a set of all feasible assignments of available orders and machines that can be illustrated as a tree-structure (Figure 2.9 on the right).

2.1.5 Summary: Solution approach requirements

This summary shall conclude the fundamentals and concepts described so far, which are relevant for the remainder of this work. Moreover, it forms the requirements for the next section that introduces various quantitative optimization methods that are applicable as solution approaches.

1. A job shop manufacturing system needs to consider different machines, flexible transportation resources, and orders. It relates to an entire production site or just a sub-system of it.

2. Modern manufacturing systems exhibit characteristics of complex systems, as multiple stochastic processes overlap each other and the range of variation is large.

3. The objective is to optimize logistical performance indicators in terms of resource utilization and order cycle time. Furthermore, a robust performance is envisaged.

4. An integrated order dispatching, covering transportation and processing, is the PPC task this research is based on. The dispatching system needs to be near real-time capable and able to perform sufficiently well under changing conditions (adaptivity).

2.2 Quantitative optimization methods

Economic behaviour and decision-making are described as rational and objective-driven processes (Adam 1996; Zimmermann 2008). Operations Research focuses on real-world decision problems (Nickel & Stein et al. 2014) and its application in form of decision-making support systems supposes to improve the quality of decisions (Domschke & Drexl et al. 2015). Support systems cover any analytical and computational method that is applicable to find the best solution alternative under a given set of constraints and objectives. This section defines the basic terms and categorizes the most appropriate quantitative optimization methods.

In the following, decision and optimization problems are seen as synonyms. The solution method can be either an *exact* method that guarantees to solve the problem to optimality or methods that do not necessarily provide the optimal solution. Any method covers the following *problem solving phases* (Domschke & Drexl et al. 2015):

1. Analysis of the problem

2. Determine the objective and solution alternatives

3. Mathematical modelling (descriptive model)

4. Data gathering

5. Solution algorithm

6. Evaluation and implementation of the solution

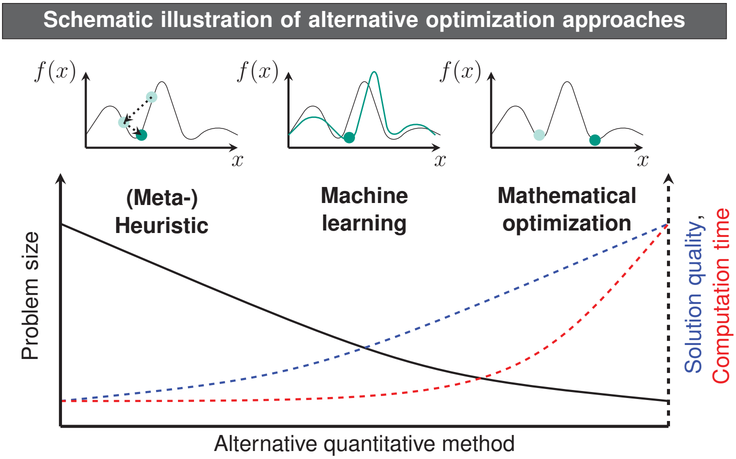

Schematic illustration of alternative optimization approaches
$f(x)$
x
$f(x)$
x
$f(x)$
x
(Meta-) Heuristic
Machine learning
Mathematical optimization
Problem size
Solution quality,
Computation time
Alternative quantitative method

Figure 2.10: Comparison of quantitative methods with respect to applicable problem size, feasible solution quality, and required computation time. The dashed, coloured lines refer to the dashed, coloured axis.

Chapter 4 and Chapter 5 follow this procedure, as the problem of order dispatching represents all characteristics of a decision problem (Mönch & Fowler et al. 2013; Domschke & Drexl et al. 2015).

In the following sub-sections, mathematical optimization, heuristics, and machine learning will be characterized and application examples named. Figure 2.10 depicts and summarizes the essential differences between the quantitative methods with respect to the applicable problem size, the solution quality, and the computational effort. Moreover, the upper part of Figure 2.10 sketches the different approaches of the quantitative methods. Generally speaking, heuristics follow a step-wise procedure, machine learning determines the optimal solution based on an approximation model, and mathematical optimization exactly determines local and global optima, for instance, based on the first and second derivative. Reinforcement learning is one representative of the machine learning category, as explained in the following. Additionally, simulation will be introduced as another quantitative approach that is, however, by itself not classified as an optimization method.

2.2.1 Mathematical optimization

Mathematical optimization problems consist of three elements: an objective function f that is optimized, i.e. maximized or minimized, constraints that need to be ensured, and a valid value range for the decision variables (Nickel & Stein et al. 2014). An example is given by the basic scheduling problem as introduced in Brucker & Knust (2012): A set of J orders has to be processed on a set of N machines. Each order j consists of k_j operations that have to be executed in a defined sequence $S_{j,k}$. Each operation must be performed on an assigned machine $x_{j,k,n} \in \{0,1\}$ with a processing time of $PT_{j,k,n} > 0$. The objective function $f(x) : X \to [0, +\infty]$ represents, for instance, the total processing time.

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & \sum_{n \in N} x_{j,k,n} = 1, \quad \forall j \in \{1, ..., J\}, k \in \{1, ..., k_j\} \\ & x_{j,k,n} \in [0,1], \quad \forall j \in \{1, ..., J\}, k \in \{1, ..., k_j\}, n \in \{1, ..., N\} \end{aligned}$$

Established mathematical optimization techniques are the following: linear, mixed-integer, non-linear, dynamic, and constrained optimization. Those as well as the concept of online optimization are explained in Appendix A1.

Many authors in the field of mathematical programming point out that the way of modelling and mathematically formulating the real-world problem has a significant influence on the computational effort (Zimmermann 2008; Nickel & Stein et al. 2014; Domschke & Drexl et al. 2015). “Good” models can be solved efficiently with the help of modern solution algorithms, although they might be $\mathcal{NP}$-hard. The following authors give an overview of efficient modelling approaches for common real-world problems in operations management: Zimmermann (2008), Klein & Scholl (2011), Kallrath (2013), and Domschke & Drexl et al. (2015).

2.2.2 Heuristics and metaheuristics

A heuristic represents an algorithm which tries to approximate the exact solution of an optimization problem. The solution approach is based on a procedure that includes empirical knowledge on how to find a “good” solution and mostly foregoes any kind of mathematical programming technique (Zimmermann 2008). In comparison to mathematical optimization, heuristics do not guarantee to find the optimum and do not provide any information on the solution quality with respect to the optimal solution, such as upper and lower bounds (Domschke & Drexl et al. 2015). However, they show for most $\mathcal{NP}$-hard optimization problems a “good” compromise between solution quality and computational effort (Zimmermann 2008).

Heuristics are commonly divided into procedures that provide feasible initial solutions and procedures that take a feasible solution and optimize it further in the direction of the objectives (Nickel & Stein et al. 2014). Important additional characteristics of heuristics are the following: firstly, feasibility especially for integer variables is always guaranteed, as naturally only feasible operations that handle integer variables as integers are allowed and no relaxation is applied. Secondly, the trade-off between a myopic algorithm with fast convergence and an approach that avoids deadlocks and local optimal solutions is known as the interplay of *exploration* and *exploitation*. This trade-off is also relevant for reinforcement learning and explained in the next sub-section in more details.

Generally speaking, simple heuristics follow a greedy approach, which always selects the best decision alternative, related to the optimization objective (Zimmermann 2008). On the contrary, algorithms that fit a broader range of problems are denoted as *metaheuristics* (Gendreau & Potvin 2005). Metaheuristics outperform heuristics, if the latter is based on a greedy-strategy that only obtains an inferior solution. This is, because the procedure of a metaheuristic implements concepts to avoid myopia, for example, by allowing worse than best (greedy) operations in some iterations or by adding randomness to the algorithm, which is also referred to as exploration (Gendreau & Potvin 2005). A more detailed explanation of heuristics and metaheuristics is outlined in Appendix A1.

Heuristics applied in PPC Simple heuristics, such as priority rules, are the prevailing industrial practice in production control, particularly in semiconductor manufacturing (Sarin & Varadarajan et al. 2011; Mönch & Fowler et al. 2013; Waschneck & Altenmüller et al. 2016). They are advantageous in terms of computational efficiency, simplicity, and real-time capability. In a semiconductor company, production control heuristics are implemented and executed by a so-called real-time dispatcher system, which allows easy programming and implementation of multi-level priority rules (Waschneck & Altenmüller et al. 2016). The system determines, for example, which job is processed next. Herein, a priority index $I_j(a_j)$ is assigned to each job j waiting in a queue based on its attributes a_j, with either the highest or lowest index being selected. Individual priority rules, thus, differ by choice of the attributes. The most important and popular priority rules are listed in the following and based on the summaries of Panwalkar & Iskander (1977), Blackstone & Phillips et al. (1982), Haupt (1989), Klemmt (2012), and Mönch & Fowler et al. (2013):

- **FIFO**: First In First Out selects the next job based on the sequence of arrival. It has the advantage to adhere to the initially planned, incoming sequence.

- **EDD**: Earliest Due Date chooses the job with the closest due date to ensure on-time delivery, given that the system load is not too high so that not all due dates can be achieved. Not always “real” customer due dates exist. If this is not the case, artificial due dates can be defined, e.g. based on a targeted flow factor (Baker 1984).

- **LWT**: Longest Waiting Time chooses the job with the longest waiting time in order to minimize order cycle times and, in particular and in contrast to FIFO, orders remaining and lingering for an extremely long time within the system.

- **NJF**: Nearest Job First selects the closest available job, leading to high utilization of the dispatching unit. The opposite of this rule, Farthest Job First (**FJF**), can improve NJF as circulation between two resources can be prevented.

- **SST**: Shortest Setup Time determines the next job based on the required setup time and is common in order sequencing in front of a machine if setup times are present.

- **SPT**: Shortest Processing Time always selects the job with the shortest processing time to keep the number of waiting jobs as small as possible and reduce the mean cycle time. Vice versa, Longest Processing Time (**LPT**) selects the longest processing time.

- **WSPT**: Weighted Shortest Processing Time selects the job with the shortest processing time that is multiplied with a weight factor, e.g. to separate regular and urgent jobs.

- **SRPT**: Shortest Remaining Processing Time selects the job with the shortest total processing time of the left operations.

- **MS**: Minimum Slack chooses the job with the least remaining slack time, i.e. the job with the least lateness. In general, this leads to a smoothing of lateness fluctuations. As high variations increase the chance of non-utilized capacities, this rule is often used in composite rules. Possible further extensions consider the variance of cycle times as variance measure (referring to Kingman's Equation 2.5).

- **FLNQ**: Fewest Lots in the Next Queue focuses on a high machine utilization as well as a balanced workload in case of multiple machines. It selects the job that has the least number of jobs in front of the next machine. Usually, this rule does not guarantee good performance in terms of on-time delivery.

- **ATC**: Apparent Tardiness Cost, according to Vepsalainen & Morton (1987), uses an index combining WSPT and MS. It is adjustable with a look-ahead parameter k (job j, job weight w_j, processing time PT_j, due date d_j, current time t, and average processing time of waiting jobs $\overline{PT}$):

$$ATC_j = \frac{w_j}{PT_j} \cdot e^{-\frac{\max\{d_j - PT_j - t, 0\}}{k \cdot \overline{PT}}} \qquad 2.7$$

The first term represents the WSPT and the second term the MS rule. However, composite rules with parameters can potentially lead to additional effort to compute the best parameter setting for the considered problem.

- An advanced **composite rule** example for order dispatching is given by Jeong & Randhawa (2001), who summarize the distance to a job (NJF), the destination queue fill level (FLNQ), and the left space at the current location of the job.

These rules can be classified into two categories according to the information and order attributes based on which the decision is made (Mönch & Fowler et al. 2013): *local* rules just use the information related to one resource, e.g. the waiting queue, and *global* rules consider future jobs or other down- or up-stream resource information.

Furthermore, additional heuristic rules can be designed by applying the following: *truncation*, i.e. limiting a particular rule to an exceptional condition such as jobs waiting too long, *conditioning*, i.e. adapting a rule to the current system state, or *multi-level* methods that add hierarchical conditions or are required in case of a tie (Mönch & Fowler et al. 2013). In practice, resolving a tie or deadlock and preventing starvation is of essential importance to ensure running production operations. For instance, FIFO is more robust than EDD, which can lead to unstable operations (Hilsenbeck 2005). Hence, in hardly any real-world application just a single heuristic rule is enough.

So far, the research of priority rules has not found any heuristic rule that dominates all other rules for any setup, system state, or objective as pointed out by Haupt (1989) and Geiger & Uzsoy et al. (2006). This fact refers to the so-called *no free lunch theorem*, according to which there is no such rule that surpasses random guessing over all possible applications (Wolpert & Macready 1997). Recent approaches that are presented in Chapter 3 design multi-level priority rules, for instance, with the help of evolutionary algorithms (Freitag & Hildebrandt 2016). However, these rules are neither obvious to understand nor simple to validate (Heger 2014). Moreover, they are still not generic enough to show a robust performance for different systems and changing conditions. Finally, multi-level rules pose the risk of contradictions within the rule-hierarchy, especially when applied in complex environments (Mönch & Fowler et al. 2013).

2.2.3 Machine learning

Machine learning covers computer programs that generate knowledge by using empirical data (experience) and optimize a specific performance criterion (Mitchell 1997). Thereby, the key characteristic of machine learning lies in the ability to draw conclusions from a database about the underlying system and its behaviour. This process is also known as *inference* (Russell & Norvig 2016).

The recent increasing importance of machine learning results mainly from the fact that for some tasks no suitable algorithm exists or the development and constant adaptation would be too costly (Alpaydin 2010). At the same time, an ever-increasing database emerges that is not adequately considered in conventional algorithms. Inference can be used to build knowledge that allows decisions to be made without having to program an explicit algorithm. This plays an important role, especially in environments with a vast solution space, because the effort to construct a tailored algorithm is inadequately high. Another factor that increases the relevance of machine learning is the increasing importance of adaptive control systems that can (re-) act in any situation autonomously (Nyhuis 2008; Stricker & Kuhnle et al. 2018).

Generally speaking, there are two ways of how *learning* is incorporated in algorithms: adjusting model parameters, such as the connection weights of an artificial neural network, or varying the model structure by changing the size of an artificial neural network.

Machine learning is commonly divided into three categories (Mitchell 1997): *unsupervised learning*, *supervised learning*, and *reinforcement learning*. All three categories generate knowledge, but based on a different range of experience and application of built-in knowledge. Unsupervised learning focuses on finding patterns to separate an unknown database, whereas supervised learning aims at pre-determined and labelled data structures. Reinforcement learning uses self-initiated actions to change its environment and receive a reward feedback to learn an optimal behaviour policy. Thus, reinforcement learning actively influences its learning data as opposed to unsupervised or supervised learning. A more detailed description and evaluation concerning the applicability in the context of adaptive order dispatching of unsupervised and supervised learning are provided in Appendix A1, according to Mitchell (1997), Russell & Norvig (2016), and Ertel (2017). This section will continue with and focus on the category of reinforcement learning.

Reinforcement learning Reinforcement learning, having its roots in psychology and the investigation of animal intelligence (Thorndike 1911), addresses the question of how an autonomous, intelligent program, which is called *agent*, observes and acts in its environment and learns to choose optimal actions in order to achieve a goal formulated at the beginning of the learning process. The term ***agent*** is, according to Russell & Norvig (2016), understood as an instance that receives information about its environment, e.g. via sensors, and can influence its surrounding, e.g. by means of actuators. Every agent's action is rewarded or punished by a scalar value as an indicator of its desirability. The goal of the agent is to maximize the cumulative reward.

The reward signal is, in some sense, a kind of supervision, which is why one can understand reinforcement learning as an intermediate between supervised and unsupervised learning. However, unlike supervised learning, reinforcement learning does not have independent data instances, because past actions and states influence the evaluation of the current situation and future states and actions. So there is a time-related dependency. Moreover, reinforcement learning is used if a priori no relation between input and output data is available, but every input-output-combination can be evaluated easily. These properties make reinforcement learning attractive for the optimization of production operations.

2.2.4 Simulation

Simulation is a powerful quantitative method that, however, is not an optimization technique itself. Nevertheless, there are several applications in which simulations show significant advantages when combined with one of the previously outlined approaches. First, when it is not possible, too dangerous, too costly, or too time-consuming to observe the process in real-world and perform real-world experiments (Domschke & Drexl et al. 2015; VDI 3633 Part 1 2014). Second, when analytic models do not exist so far, include too many assumptions, or are too complex to represent the real-world problem (Law 2014). So, the first and foremost aim of simulation applied in production planning and control is the representation of systems that do not exist, have stochastic properties, or consist of multiple components that interact with each other, as in these cases analytical models are infeasible or even impossible to build (Mönch & Fowler et al. 2013; Domschke & Drexl et al. 2015).

There are two types of simulations depending on the time progress within the simulation (Law 2014; Gutenschwager & Rabe et al. 2017): *discrete-event* and *continuous* simulations. The former type processes one event after another. After the simulation start, events are generated, timed, and removed when their end is reached. An event handler keeps track of all events and executes events based on their due time. Nearly all applications in logistics and production within the discrete industry fall into this category. In the latter case, the time progresses continuously, meaning in infinitesimal small time steps. This is appropriate for process industry applications and, hence, not further considered in the present work.

Simulations can be combined with mathematical optimization methods or, more recently, with machine learning algorithms. In the former case, the simulation is used to evaluate solution alternatives, e.g. a schedule as output of a constraint optimization model (März & Krug 2011; Gutenschwager & Rabe et al. 2017). That is beneficial when the system's dynamics do not allow an analytic solution evaluation. However, due to the computational effort of running a simulation, this approach is limited to small problems (Klemmt 2012).

For combined simulation and machine learning applications, the advantage of the simulation is its ability to generate “cheap” training data. In most recent applications of reinforcement learning, this is a common approach, such as the example of the board game Go (Silver & Schrittwieser et al. 2017), the computer game StarCraft[1], or a robotic hand solving Rubik’s cube[2]. This approach will be further described in the next sections as it is also a central element of the present research.

2.2.5 Summary: Quantitative methods for an adaptive order dispatching

Summarizing the presented quantitative methods, the two major influencing factors that affect the selection of the appropriate method are the *degree of centrality* and the duration of the *planning horizon* (Nyhuis 2008). Figure 2.11 gives a structured overview of the approaches that are applicable for PPC.

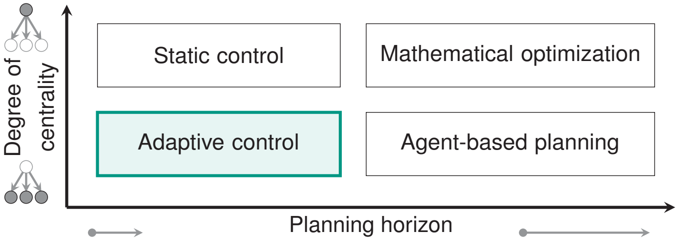


Figure 2.11: Overview of appropriate quantitative methods for production planning and control (adapted from Nyhuis (2008)). The focus of the present work is highlighted.

If the considered planning horizon is very long and the degree of centrality high, i.e. there is only a single decision-making unit, mathematical optimization is predominant, whereas for decentral systems agent-based planning is more appropriate (Domschke & Drexl et al. 2015). The two most short-sighted approaches are static and adaptive control systems, which are extensively discussed in the previous section.

[1] Vinyals et al. (2019), *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II* `https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii` (accessed on 17.06.2020).

[2] Akkaya et al. (2019), *Solving Rubik’s Cube with a Robot Hand* `https://openai.com/blog/solving-rubiks-cube` (accessed on 17.06.2020).

In addition to that, the summary of this section is given as follows:

1. The requirements listed in Section 2.1.5 can be fulfilled by an adaptive order dispatching system.
2. Heuristic priority rules are commonly applied in industrial applications and thoroughly studied in academia.
3. The quality of an optimization method is largely determined by the level of knowledge about the problem characteristics. For instance, the process of creating decision rules requires deep understanding of the underlying system.
4. Reinforcement learning fulfils the requirements of an adaptive order dispatching.
5. Discrete-event simulation is a powerful tool to evaluate complex systems and to create a sufficient database for a reinforcement learning application.

2.3 Reinforcement Learning

Reinforcement learning optimizes the behaviour of an autonomous agent within a defined environment (Kulkarni 2012). The agent learns based on a trial-and-error approach, wherein learning data is generated, iteratively based on observations that result in actions and a reward value that is provided as feedback, i.e. a positive or negative scalar. Collecting experience by actively interacting with the environment is referred to as *sampling* (Sutton & Barto 2018).

One reason for recent advances in reinforcement learning is that, nowadays, still many control optimization applications ask for specific algorithms that do not exist yet. Either it is too complex to develop or the system constantly changes and requires a continuous adjustment of the control algorithm (Alpaydin 2010). Reinforcement learning offers a promising approach to address these challenges.

2.3.1 Agent and environment

Since the definitions as well as related terms of agent and environment are ambiguous in literature, the introduction by Russell & Norvig (2016) is considered from now on. The term *agent* describes a computer system that is embedded in an environment and makes decisions. According to these authors, there are the following agent types:

- *Simple reflex* agent: Behaving based on reflex-like condition-action rules, such as “if A then B”.

- *Model-based reflex* agent: In addition to the simple reflex agent, this agent keeps track of the environment state via an internal model of the environment, e.g. a history of past actions, but chooses the action in the same way.

- *Goal-based* agent: The behaviour is not driven by condition-action rules but directed by certain goals that need to be achieved. Thereby, an internal model not only recalls the history but also alternative actions are assessed.

- *Utility-based* agent: This agent replaces fixed goals by a utility function that determines the preference among different future environment states. Usually, the expected utility is maximized considering probabilities for each decision alternative.

- *General learning* agent: That concept is based on four major elements. A performance element that decides on actions, a critic element that evaluates the current agent's performance in its environment and how to improve it, a learning element that changes and adjusts the performance element, being the main optimization element, and, finally, a problem generator, generating new learning experience.

Concerning the internal model of the agent, there are two options established (Kaelbling & Littman et al. 1996): *model-free* means that there is no specific model given to the agent and the agent itself learns a model directly from the environment based on a generic approach, e.g. an artificial neural network. In contrast, *model-based* agents do have a constructed internal model to start with and continuously adjust it to meet the current environment state. An example of a model-based approach is, for instance, the usage of the set of rules in a board game or the laws of Physics in a physical real-world application.

Reinforcement learning falls into the category of *general learning* agents. The interaction between the agent and the environment is the central part of the learning process. It is illustrated in Figure 2.12 and is linked to the concepts of control theory and adaptive controllers introduced in Chapter 1 (see Figure 1.1). The agent represents the controller, the environment the controlled system, the action the control signal, and the set-point is related to the reward.

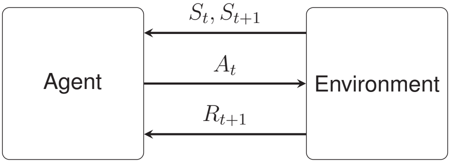


Figure 2.12: Key elements and basic interaction mechanism of reinforcement learning (adapted from Sutton & Barto (2018)).

The following introduction of reinforcement learning is based on the terminology of Sutton & Barto (2018). The interaction between agent and environment happens in a series of discrete time steps $t = 0, 1, 2, \ldots$. The reinforcement learning agent (short RL-agent) receives in t the information about the current state of the environment as state representation S_t whereupon action A_t is selected. In the next time step $t + 1$ it receives the evaluation of choosing A_t in S_t as reward R_{t+1}. Afterwards, a new state S_{t+1} results based on the action and the environment dynamics. This information is used by the agent to optimize the action selection policy. The policy $\pi_t(a \mid s)$ represents the conditional probability of $A_t = a$ if $S_t = s$. An exemplary interaction sequence looks as follows:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, ... \qquad 2.8$$

Note that the learning process does not directly require manual supervision, once all elements, i.e. state, action, and reward function, are defined. Chapter 4 will focus on the modelling of reinforcement learning in details. Thus, at this point it is enough to conclude that the function that determines the reward, the relevant state information, the set of possible actions, and the optimization algorithm are the design elements that need to be adjusted for any application of reinforcement learning. The remainder of this section explains the basic optimization mechanisms as theoretical background of reinforcement learning.

Multi-agent reinforcement learning Systems that are subjected to numerous stochastic influences and for which decisions are made in multiple instances show a high degree of complexity (see Section 2.1.1). In such systems, agent-based decentral control architectures are particularly suitable (Monostori & Váncza et al. 2006; Gabel 2009). Multiple agents can pursue different goals and make autonomous decisions independently, which reduces the complexity comparing to an integrated control architecture. Moreover, multi-agent systems are promoted in the context of Industry 4.0 (Lasi & Fettke et al. 2014), cyber-physical production systems (Monostori & Kádár et al. 2016), and self-optimizing production systems (Permin & Bertelsmeier et al. 2016). However, multi-agent systems are not explicitly considered in the present work, though some presented concepts are transferable and can be extended to multiple agents. For literature on multi-agent systems and reinforcement learning, the reader is referred to the work of Caridi & Cavalieri (2004), Mönch (2006), Monostori & Váncza et al. (2006), Gabel (2009), Busoniu & Babuska et al. (2008), and Gupta & Egorov et al. (2017).

2.3.2 Markov Decision Process

The theoretical foundations of reinforcement learning are ascribed to the theory of dynamic programming. In particular, Markov Decision Processes (MDP) build the theoretical basis for sequential decision problems, i.e. optimization tasks in stochastic and dynamic environments in which the optimal decision-making for a set of feasible actions is asked for (Puterman 1994; Bertsekas 2005).

A *finite MDP* is defined by the following elements (Sutton & Barto 2018): finite set of environment states $\mathcal{S}$, finite set of actions $\mathcal{A}$, transition probabilities p, reward function r, and discount rate γ. Herein, the transition probabilities $p(s', r \mid s, a)$ completely define the dynamics and transition behaviour of the environment, i.e. what is the probability $\mathbb{P}$ of the next state $s' \in \mathcal{S}$ based on the prior state $s \in \mathcal{S}$ and a given action $a \in \mathcal{A}$. The four-argument function $p(s', r \mid s, a)$ stands for the entire environment dynamics and the three-argument function $p(s' \mid s, a)$ for the state-transition probabilities. The reward function r describes the evaluation according to the optimization objectives:

$$r(s, a, s') = \mathbb{E}\left[R_t \mid S_{t-1} = s, A_{t-1} = a, S_t = s'\right] \tag{2.9}$$

MDPs are divided into processes of *episodic* and *continuous* nature. If the interaction of the agent with its environment is terminated at a certain time or event, there is an episodic MDP and otherwise a continuous MDP. Another important differentiation characteristic of MDPs is the observability of the state, i.e. whether it is *completely observable* (MDP) or just *partially observable* (POMDP). Most real-world problems fall into the category of POMDP. Generally speaking, the reinforcement learning methods discussed in this work are not limited to MDPs, but also applicable for POMDPs as well as for episodic and continuous processes. Hence, from hereupon just the term MDP is used.

Markov property The key concept of an MDP is the *Markov property*. It refers mainly to the state representation, which needs to fulfil the following property (Sutton & Barto 2018):

$$p(s', r \mid s, a) \overset{!}{=} \mathbb{P}(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a) \tag{2.10}$$

The property is fulfilled, if the transition function p, i.e. the transition to state S_t under observation of reward R_t, depends only on the preceding state S_{t-1} and action A_{t-1}.

When applying reinforcement learning, the agent receives the state information S_t in the form of a vector in each time step. According to the Markov property, the state representation must, thus, contain all task- and objective-related information. However, as stated by Sutton & Barto (2018) and demonstrated by other authors too (see Chapter 3), it is not necessarily required that the property is strictly fulfilled and reinforcement learning is still applicable.

Visualization MDPs are usually visualized in the form of transition graphs. States and actions are depicted as nodes and possible state transitions by weighted edges. The weight of an edge represents its probability. An episodic process with three states and two actions is shown in Figure 2.13. The state s_e is the terminal state and β indicates the probability of the state transition if action a_1 is selected in state s_1. All other transition probabilities are equal to 1, as there is only one possible next state.

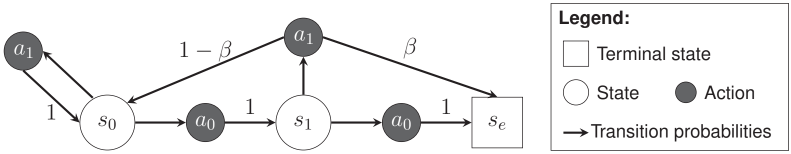


Figure 2.13: Exemplary Markov decision processes visualization as transition graph.

2.3.3 Basic concepts of Reinforcement Learning

Taking the mathematical formulation as an MDP, the basic concepts of RL-methods are the following: the agent's policy, the reward function, the value function, the optimal value function as well as policy, and the exploration-exploitation trade-off. These form the basis for a broad range of RL-methods and are described in more detail in this section, again according to the terminology of Sutton & Barto (2018).

Policy The agent's policy $\pi(a \mid s)$ describes the agent's action selection behaviour[1]. For each combination of states $s \in \mathcal{S}$ and actions $a \in \mathcal{A}$, it indicates the probability of selecting a particular action while observing a specific state:

$$\pi(a \mid s) = \mathbb{P}(A_t = a \mid S_t = s) \tag{2.11}$$

In reinforcement learning, the policy is a dynamic element that is iteratively adjusted during the learning phase to optimize the agent's behaviour. The initial probabilities are set random in most implementations.

[1] For reasons of simplicity, the subscript t is omitted for the agent's policy π. However, as the policy is adapted over the learning process, it is not independent of t.

Reward function The reward function $r(s,a)$ is the numerical representation of the learning objective. It determines the feedback and evaluation value $R_{t+1} \in \mathbb{R}$ as a consequence of action $a \in \mathcal{A}$ and state $s \in \mathcal{S}$. In line with the MDP definition, the agent optimizes the long-term goal fulfilment, i.e. not only the next reward but the sequence of future rewards $R_{t+1}, R_{t+2}, ...$. The expected discounted reward G_t is computed via:

$$G_t = R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot R_{t+3} + ... = \sum_{k=0}^{T} \gamma^k \cdot R_{t+k+1} \tag{2.12}$$

The reward values per time step are weighted with a discount factor $\gamma \in [0,1]$, which determines the weight of future and current valuations. Hence, the parameter γ is of central importance to the reward function. If the factor is rather small, the impact of future rewards is low and, hence, the agent aims at short-term reward maximization, while a high value implies a far-sighted decision-making behaviour. In an episodic MDP, the sequence is cut off when a terminal state is reached, and T is set to infinity in a continuous MDP.

Conclusively, the reward depicts *what* to achieve and not *how* to achieve it, which is relevant when interpreting the computational results in Chapter 5.

Value function The value function $v_\pi(s)$ determines the desirability of a state. Thereby, it joins the policy π and the expected reward G_t, and forms the basis for any optimization algorithm. It considers, based on state S_t, the expected future rewards under the current policy π. In addition to that, the value can be determined on the basis of state-action combinations in the form of the so-called Q-function $q_\pi(s,a)$ (also called state-action value). Both are defined as follows:

$$v_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k \cdot R_{t+k+1} \mid S_t = s \right] \tag{2.13}$$

$$q_\pi(s,a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k \cdot R_{t+k+1} \mid S_t = s, A_t = a \right] \tag{2.14}$$

The value function can be also defined recursively, according to Bellman's Equation 2.20. This form of representation makes it possible to express the value of a state over the values of its successor states (Sutton & Barto 2018):

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi \left[G_t \mid S_t = s \right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma \cdot G_{t+1} \mid S_t = s \right] \\ &= \sum_a \pi(a \mid s) \cdot \sum_{s',r} p(s', r \mid s, a) \cdot \left[r + \gamma \cdot \mathbb{E}_\pi \left[G_{t+1} \mid S_{t+1} = s' \right] \right] \end{aligned}$$

$$= \sum_{a} \pi(a \mid s) \cdot \sum_{s',r} p(s', r \mid s, a) \cdot [r + \gamma \cdot v_{\pi}(s')] \tag{2.15}$$

This is of central importance for any RL-algorithm and reconsidered in Section 2.3.4.

Optimal value function and optimal policy The recursively defined form of the value function in Equation 2.15 allows, according to Bellman, the solution of an MDP. The optimality principle of Bellman (1966) states that the solution, i.e. the optimal policy, of a decision problem can be aggregated from a sequence of optimal solutions of sub-problems, i.e. sub-policies.

The agent's goal is to maximize the expected reward G_t and determine an optimal policy π^* according to that. A policy is called optimal if and only if G_t is greater or equal to any state $s \in \mathcal{S}$ and any other possible policy π (Sutton & Barto 2018):

$$v_{\pi^*}(s) \geq v_{\pi}(s) \quad \Rightarrow \quad \pi^* \geq \pi, \quad \forall s \in \mathcal{S} \tag{2.16}$$

Accordingly, determining the optimal policy is equivalent to the optimal value function. For this, the maximum of the value function (Equation 2.13) or the Q-function (Equation 2.14) are:

$$v^*(s) = \max_{\pi} v_{\pi}(s), \quad \forall s \in \mathcal{S} \tag{2.17}$$

$$q^*(s, a) = \max_{\pi} q_{\pi}(s, a), \quad \forall s \in \mathcal{S}, a \in A(s) \tag{2.18}$$

The optimal value function $v^*(s)$ can be re-formulated by exploiting the fact that the value of a state under an optimal policy π^* equals the expected total reward for the selection of the best action of the optimal Q-function $q^*(s, a)$. This formulation is particularly useful, as it avoids the reference to a specific policy.

$$\begin{aligned}
v^*(s) &= \max_{a \in A(s)} q^*(s, a) \\
&= \max_{a \in A(s)} \mathbb{E}_{\pi^*} \left[R_{t+1} + \gamma \cdot G_{t+1} \mid S_t = s, A_t = a\right] \\
&= \max_{a \in A(s)} \mathbb{E} \left[R_{t+1} + \gamma \cdot v^*(S_{t+1}) \mid S_t = s, A_t = a\right] && 2.19 \\
&= \max_{a \in A(s)} \sum_{s',r} p(s', r \mid s, a) \cdot [r + \gamma \cdot v^*(s')] && 2.20
\end{aligned}$$

Equation 2.19 and Equation 2.20 are two different forms of *Bellman's optimality* equation that can be formulated analogously for the Q-function:

$$q^*(s,a) = \max_{a' \in A(s)} \mathbb{E}\left[R_{t+1} + \gamma \cdot q^*(S_{t+1}, a') \mid S_t = s, A_t = a\right] \tag{2.21}$$

$$= \max_{a' \in A(s)} \sum_{s',r} p(s', r \mid s, a) \cdot \left[r + \gamma \cdot q^*(s', a')\right] \tag{2.22}$$

Solving these equations and determining the optimal value function are the core of *value-based* RL-methods, which are outline in the next section. Bellman's equation translates into a system of equations and, hence, the optimal value function is equivalent to the solution of a system of equitations with one equation for each state (Sutton & Barto 2018).

So, the optimal value function and optimal policy can be determined by mathematical optimization and explicitly formulated. However, in real-world applications, this is computationally hard due to the large amount of state-action pairs that need to be evaluated and stored. Generally speaking, reinforcement learning is an approximation method, that approximates the optimal policy, e.g. by reducing states with a low probability to a random action selection policy and, thus, putting the focus just on frequently occurring states (Sutton & Barto 2018). The underlying optimization approach follows the optimality principles just explained.

Exploration and exploitation In reinforcement learning, the agent learns a policy through the experience gathered by interacting with the environment. This raises the question to what extent the agent should explore the environment as it represents the solution space (referring also to the problem generator feature of general learning agents in Section 2.3.1). Moreover, exploring might result temporarily in a lower reward in order to receive a higher reward in the long-run. To answer this question, there are two opposing concepts that are already introduced in Section 2.2.2 on (meta-) heuristics (Witten 1976; Glover & Kochenberger 2003; Sutton & Barto 2018): *exploration* and *exploitation*. In exploitation, known information is used *greedily* to maximize the reward. In contrast, the purpose of exploration is to explore the environment to gather new information and, potentially, discover an even better policy. It is broadly agreed on, that only by combining exploration and exploitation the optimal policy can be determined (Kaelbling & Littman et al. 1996).

One common approach in reinforcement learning to enforce exploration is the ϵ-greedy action selection (Sutton & Barto 2018). Herein, at any time step a random action is selected with a probability of $\epsilon \in [0, 1]$. Usually, ϵ is set to a high value in the beginning and decreases over a linear amount of iterations to zero or a small number above zero.

2.3.4 Model-free methods

In most real-world applications, the transition probability function p, i.e. the dynamics of the environment, is unknown. In such cases, *model-free* reinforcement learning methods are applied, in contrast to *model-based* methods, such as dynamic programming, that are applicable when the entire MDP is known (Kaelbling & Littman et al. 1996). Generally speaking, using a model results in a more efficient learning process with fewer interactions with the environment and, if the model is accurate, the RL-policy also matches more accurately to the real-world than model-free methods. As stated before in Section 2.3.1, the regarded application of this work does only allow for model-free methods, as a complete environment model is not available. This sub-section explains which basic RL-methods exist that are used to optimize and solve the underlying MDP.

All model-free RL-methods have in common that they first act according to $\arg\max_{a \in A(s)} q_\pi(s, a)$, i.e. the action with the highest q-value is selected, and use the experience to adjust and update the policy or the value function of the current policy. Collecting experience through interaction with the environment is called sampling, as explained before. Computing the value function for a given policy is called *policy evaluation* and improving the policy based on an adjusted value function is named *policy improvement*. When combining both steps, the agent's policy eventually converges to the optimal policy π^* (Sutton & Barto 2018).

According to Sutton & Barto (2018), three types of methods are introduced that are applicable for model-free reinforcement learning (see also Figure 2.14 – the fourth method, Trust Region Policy Optimization, will be presented in Section 4.4.1.1 as it is the method applied in the present work): two *value-based* methods, Monte Carlo and Temporal-Difference, and *policy-based* methods. The presented methods are generic procedures that enclose a range of algorithms that implement the methods' principles.

Figure A1.1 in Appendix A1 sketches the different approach of value- and policy-based methods for a simplified and schematic example. Value-based methods iteratively compute state values and derive the policy based on these. In contrast to that, policy-based methods start with an initial policy and improve the entire policy in every iteration.

Monte Carlo First, Monte Carlo methods learn the optimal value function from an average sample of an entire episode. Episodes terminate in a fixed manner and, at the end of an episode, the policy is updated accordingly. So, this type of method is only applicable to MDPs, where the environment can be defined in episodes.

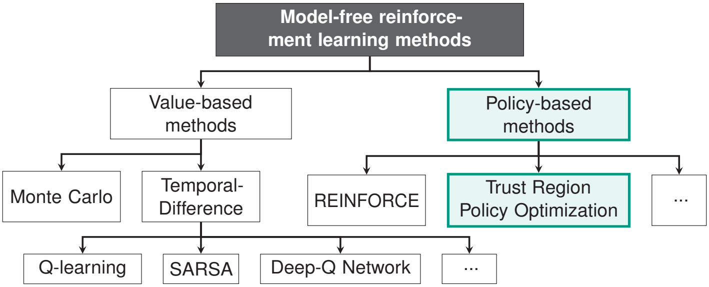


Figure 2.14: Classification of model-free reinforcement learning methods. The focus of the present work is highlighted.

For a given episode sample of policy π: $S_0, A_0, R_1, S_1, A_1, R_2, ..., S_T$, where S_T denotes the terminal state, the return in each time step is calculated by $G_t = R_{t+1} + ... + \gamma^{T-1} \cdot R_T$. Monte Carlo methods evaluate policy π by averaging the obtained returns for each state and action via Equation 2.14, thus learning the value function over the empirical mean.

Temporal-Difference Second, temporal-difference (TD) methods do not require complete episodes as evaluation experience to improve the value function of a policy, as it is required for Monte Carlo methods. It improves the estimation of a value function after every time step. To do this, the simplest algorithms use Equation 2.13 or Equation 2.14 directly. For a given transition, i.e. action A_t, state S_t, policy π, reward R_{t+1}, and new state S_{t+1}, the estimation of state value $v_\pi(S_t)$ can be improved by using the estimate $v_\pi(S_{t+1})$ of the subsequent state value and the obtained reward R_{t+1}:

$$v_\pi(S_t) \longleftarrow v_\pi(S_t) + \alpha \cdot (R_{t+1} + \gamma \cdot v_\pi(S_{t+1}) - v_\pi(S_t)) \qquad 2.23$$

The learning rate α is, next to the discount rate γ, another important parameter that needs to be defined upfront and determines to what extent new experience updates the old evaluation[1]. Due to this, TD-methods can be used in non-terminating environments to update the value function directly after the execution of a single action.

If the value function is approximated by means of a TD-method and not just based on one state transition but n state transitions, Equation 2.23 is extended to:

$$v_\pi(S_t) \longleftarrow v_\pi(S_t) + \alpha \cdot \left(R_{t+1} + ... + \gamma^{n-1} \cdot R_{t+n} + \gamma^n \cdot v_\pi(S_{t+n}) - v_\pi(S_t)\right) \qquad 2.24$$

[1] The improvement of an estimation based on an estimate is also called *bootstrapping*.

For $n = \infty$, in the presence of an episodic environment, this equation converges to G_t and, thus, the TD-method equals to a Monte Carlo method. Note that Equation 2.23 and Equation 2.24 are transferable to the Q-function.

Q-learning and SARSA are two representative algorithms that fall into the category of TD (see Figure 2.14). Both are explained in the following for the Q-function. Q-learning, developed by Watkins & Dayan (1992), applies the following update rule:

$$q_\pi(S_t, A_t) \longleftarrow q_\pi(S_t, A_t) + \alpha \cdot (R_{t+1} + \gamma \cdot \max_{a \in A(S_t)} q_\pi(S_{t+1}, a) - q_\pi(S_t, A_t)) \tag{2.25}$$

The advantage of using the **Q-learning** algorithm, which also caused its breakthrough and broad application (see Chapter 3), is that this update rule is independent of the actual selected action A_t. In any step, the optimal value $\max\limits_{a \in A(S_t)} q_\pi(S_{t+1}, a)$ is used for the update and improvement step, which directly follows Bellman's principle of optimality. This facilitates the algorithm's proof of convergence to an optimal policy.

SARSA considers for every update step the state, action, reward, next state, and next selected action (hence, the quintuple sequence of s̲tate, a̲ction, and r̲eward explains the abbreviation SARSA) (Rummery & Niranjan 1994):

$$q_\pi(S_t, A_t) \longleftarrow q_\pi(S_t, A_t) + \alpha \cdot (R_{t+1} + \gamma \cdot q_\pi(S_{t+1}, A_{t+1}) - q_\pi(S_t, A_t)) \tag{2.26}$$

The SARSA algorithm is called *on-policy*, as the update follows policy π, whereas the Q-learning update in Equation 2.25 is *off-policy*, because the terms of the update formula are independent of the current policy π.

Policy-based methods Finally, while the previous methods learn the value function and select the next action based on value estimates, a policy-based method learns the policy directly without relying on a value function for action selection. Usually, a vector θ is introduced that parametrizes the policy as $\pi_\theta(a \mid s, \theta)$. The optimal parametrized policy is given by the expected value $v_\pi(\xi) = \mathbb{E}_{s_0 \sim \xi}\left[v_\pi(s_0)\right]$[1] for a starting state distribution ξ and starting state s_0 taken from ξ according to:

$$\max_\pi v_\pi(\xi) \approx \max_\theta v_{\pi_\theta}(\xi) \tag{2.27}$$

For determining the optimal parameter vector θ^*, i.e. the optimal policy, optimization methods are used that maximize the performance of the policy measured by a scalar function $J(\theta)$

[1] The tilde symbol $s_0 \sim \xi$ means that "s_0 is distributed as ξ".

(Sutton & Barto 2018). The most frequently used method is the gradient descent method, which determines the updated parameters θ_{t+1} on basis of stochastic estimates $\widehat{\nabla J(\theta_t)}$, whose expectation approximates the gradient of the performance measure $\nabla J(\theta_t)$ via (Sutton & Barto 2018):

$$\theta_{t+1} = \theta_t + \alpha \cdot \widehat{\nabla J(\theta_t)} \tag{2.28}$$

Again the learning rate α is used for the update. The action selection, for a discrete action space, is commonly expressed by a numerical preference function $h(s, a, \theta)$ and the selection probability is calculated according to the so-called *soft-max* distribution, which is defined as follows (Sutton & Barto 2018):

$$\pi(a \mid s, \theta) = \frac{e^{h(s,a,\theta)}}{\sum_{a' \in A(s)} e^{h(s,a',\theta)}} \tag{2.29}$$

This procedure reveals three advantages of policy-based methods (Sutton & Barto 2018). First, they are feasible to converge to a deterministic policy and, at the same time, preserve exploration properties. Second, the result can be a stochastic policy, as the policy naturally represents a probability function. Stochastic policies are, for instance, advantageous in environments with imperfect information. Finally, the insertion of an initial policy that is based on prior knowledge is applicable.

One policy-based method is the **REINFORCE** algorithm by Watkins & Dayan (1992) (see Figure 2.14). It takes the basic stochastic gradient update from Equation 2.28 and derives the following update rule:

$$\theta_{t+1} = \theta_t + \alpha \cdot G_t \cdot \frac{\nabla \pi_\theta(A_t \mid S_t, \theta_t)}{\pi_\theta(A_t \mid S_t, \theta_t)} \tag{2.30}$$

The update of the parameters θ is proportional to the total return of an episode G_t, as for Monte Carlo, and the fraction of the action probability gradient and the action probability itself. The second term directs the parameter update into the direction where the probability of selecting action A_t in state S_t again is the highest and controls that a high action probability does not self-enforce.

Hybrid policy- and value-based methods As stated before, the Q-function is not learned in policy methods, but estimated by state transitions. However, without going into details, there are hybrid algorithms called *actor-critic methods* that integrate value- and policy-based methods. Generally speaking, the variance of an estimation is high when the number of samples is low (Grondman & Busoniu et al. 2012). So, by additionally learning the Q-function in a policy-based method, improved performance is expected. The actor optimizes the parameters θ in the direction suggested by the critic that learns the Q-function, for example, using Monte Carlo or TD methods, in order to reduce the variance. For further details, it is referred to the survey of Grondman & Busoniu et al. (2012).

2.3.5 Value function approximation

For MDPs with a manageable number of states and actions, the evaluation of each possible state v or state-action pair q can be stored in a look-up table. However, this is not possible for MDPs with a large or infinite number of states and state-action pairs, since, on the one hand, there is insufficient memory available and, on the other hand, learning every individual state or state-action pair is extensively time-consuming (Sutton & Barto 2018).

Therefore, function approximators are applied in practice to approximate the value function. Widely used approximators are artificial neural networks (ANN). ANNs are usually applied in supervised learning. In reinforcement learning ANNs can be applied, too, and its weight parameters are adjusted during the learning process to approximate the value function or the parametrized policy (Sutton & Barto 2018). Another advantage of ANNs lies in the ability to generalize and infer unseen states by using data samples (Russell & Norvig 2016). Furthermore, ANNs have the strength, comparing to other approximation methods, that they are able to capture non-linear dependencies.

A generic ANN consists of several connected layers, each consisting of multiple neurons. A weight vector represents the connections from one layer to the following. Every neuron passes all input values to an output value through a predefined mathematical operation.

Although there are several types of ANNs, it is here enough to understand a multi-layer perceptron (MLP) as illustrated in Figure 2.15. It consists of a variable number of *perceptron* neurons. Each perceptron is connected to each perceptron of the previous layer. The state Z of a perceptron with input vector X and weight vector W is calculated via (Ertel 2017):

$$Z(X,W) = x_0 \cdot w_0 + \sum_{i=1}^{|X|} x_i \cdot w_i \qquad 2.31$$

Note that the input x_0 has a special meaning, as it always equals 1 and does not depend on the output of a neuron of a previous layer. Together with the associated weight w_0, this pair is called *bias* and it can be shown that it increases the generalizability of the network (Ertel 2017). The *activation function* $y = f(Z)$ is used to calculate the output of the perceptron,

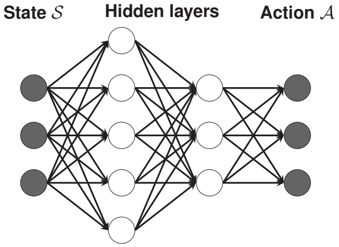


Figure 2.15: Schematic illustration of an MLP network resembling an RL-policy with three input neurons (states), two hidden layers, and three output neurons (actions).

which in turn represents an input for the perceptrons of the following layer. Common activation functions are, for example, the rectified linear unit (ReLU) or the hyperbolic tangent:

$$f_{ReLu}(Z) = \max\{0, Z\}\,, \quad f_{tanh}(Z) = \tanh Z \qquad 2.32$$

The update rule of ANNs in reinforcement learning is equivalent to supervised learning applications. The aim is to calculate the ANN's output (prediction) for each training input data and to minimize the error between the prediction and the given output. The squared error is commonly applied as error measure (Ertel 2017). The update of weights is determined by the backpropagation algorithm, which adjusts the weights depending on their influence on the prediction error. Further details can be found in Ertel (2017).

Deep reinforcement learning Deep reinforcement learning refers to RL-algorithms that apply "deep" ANNs, i.e. ANNs with several hidden layers and many neurons in each layer (Mnih & Kavukcuoglu et al. 2013; Schmidhuber 2015). The most prominent applications are the board games Chess and Go by Silver & Hubert et al. (2018), which led to a rise in many other fields of application (see Chapter 3). Another application of deep reinforcement learning is given by the just introduced policy-based methods that rely on a parametrized policy. Herein, θ can be represented by all weights of an ANN. This idea is considered in the algorithm applied in the present research.

Generally speaking, deep ANNs and deep reinforcement learning are regarded as *black-box* approaches (Guidotti & Monreale et al. 2018). They are captured by inputs and outputs, however, the internal structure is unknown because of the multifold stochastic and non-linear dependencies between the neurons. On the contrary, the structure of *white-box* models, e.g. linear regression, can be described explicitly. This fact will be considered in the evaluation of the computational results in Chapter 5.

2.3.6 Summary: Adaptive order dispatching based on Reinforcement Learning

Reinforcement learning algorithms are able to act as adaptive controller. Figure 2.16 illustrates the key differences between learning-based and heuristic approaches. The figure depicts the difference upon the extent and way of integrating data, as data is regarded as the main driver of future operational performance improvements, not limited to PPC.

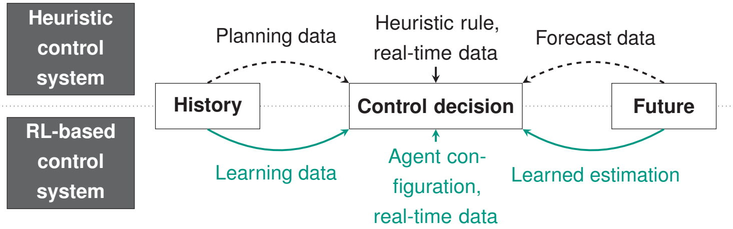


Figure 2.16: Schematic comparison of heuristic and RL-based control systems by emphasizing the different ways of integrating historic and future information.

Heuristics use real-time data and a predefined rule that is in most cases static. The rule compromises the optimization objective as well as the domain knowledge about how best to achieve high performance. Past and future data are hardly considered in terms of planning data or forecasts. In contrast, reinforcement learning is just directed by a reward signal that does only set the goal *what* to achieve by evaluating state-action pairs and not *how* to achieve it. Moreover, the RL-algorithm records the history as learning data and information on the future is implicitly incorporated in the decision by learned estimations.

Summarizing the key concepts of reinforcement learning leads to the following conclusion:

1. Reinforcement learning assumes an agent that directly interacts with its environment to determine the optimal solution for a sequential decision problem.
2. The theoretical foundations of reinforcement learning are MDPs that also back up the optimality of RL-algorithms and perfectly fit to the adaptivity requirement.
3. State, action, reward, and agent configuration form the basic modelling choices that need to be defined upfront applying reinforcement learning.
4. This work focuses on a model-free, policy-based optimization method, optimizing the likelihood of actions in contrast to value-based methods that optimize value estimations.
5. Most real-world problems come with large state and action spaces that require approximation methods, such as ANNs, and are also able to generalize.

3 State-of-the-art literature review

The idea of learning the behaviour of systems autonomously goes back to the 1950s (Sutton & Barto 2018). To date, numerous approaches have been developed for the implementation of learning-based systems that can be applied to optimization problems. Among these, reinforcement learning offers great potential for production control tasks. Section 3.1 first defines the research focus areas of this work and is based on the introduction and fundamentals presented so far as well as the outcome of the literature review. Afterwards, the latest research in three fields is introduced: Section 3.1.1 on advanced rule-based dispatching approaches, Section 3.1.2 on RL-approaches in adaptive PPC, and Section 3.1.3 on advanced RL-applications that are not related to production applications. Finally, Section 3.2 concludes with the research deficit that is addressed in the present work.

3.1 Literature review of research focus areas

The research focus areas of the present work are outlined in the following paragraphs. These are the dimensions that are relevant to this research and are used to describe the research deficit. Table 3.1 summarizes the review and is based on these dimensions, too.

System scope: Semiconductor manufacturing and, in particular, wafer frontend fabrication is categorized as complex *job shop* (see Section 2.1.1), whereby the production program is characterized by a *high volume* of a *high number of variants*. Moreover, the complexity is based on numerous overlapping *stochastic processes*, e.g. machine availability, processing times, and a *complex material flow*. Hence, semiconductor manufacturing differs from other industries (Mönch & Fowler et al. 2013).

Production control task: As production control task, *order dispatching* related to *transportation* and *processing* is considered. The control strategy needs to be *adaptive*, i.e. it is characterized in such a way that control decisions are always made based on the current state of the system. Ultimately, only a limited amount of computation time is available for solving the control decision problem, i.e. a decision is required in near *real-time*, since the execution cannot delay the production process.

Performance objective: The performance target system is *multi-objective*, i.e. the optimality of the dispatching is defined by several goals. It can consist of both *order-related*, e.g. due date or cycle time, and *resource-related*, e.g. utilization or uptime, performance indicators. According to the adaptivity, control decisions need to be *dynamically* evaluated based on the current system state, comparing to an evaluation at the end of a period, e.g. day or shift.

Optimization approach: Rule-based heuristics meet the near real-time requirement. Additionally, data-driven *reinforcement learning* algorithms can be applied. The required learning data is generated in a *simulation-based* approach. A simulation allows the generation of an unlimited and consistent training database as well as the comparison of alternative production system configurations. Besides, simulations enable a direct benchmarking with alternative control methods, as they can be run in parallel. Moreover, an investigation of the *plausibility* of the implemented RL-agents is feasible. Plausibility, in that case, does not mean the full decision-making traceability of the black-box models, but to find an approach to capture some kind of reasoning behind it. Finally, the results of the present work should be comprehensive and evaluate *RL-modelling alternatives* to allow for recommendations for similar production control applications.

3.1.1 Dynamic selection and parametrization of dispatching rules

In contrast to the adaptive selection of single dispatching decisions as evaluated in this work, the authors covered in this sub-section show examples of optimization methods that select or parametrize dispatching rules dynamically.

Kim & Lee (1998) design a genetic algorithm combined with RL-features for a fictional machine scheduling problem. They propose an iterative list scheduling method that generates and refines schedules iteratively. In order to handle the vast amount of possible states and actions, they use the principle of genetic algorithms and policies are encoded as chromosomes. A population represents a set of policies. Thereby, they reach for most computer experiments a performance similar to specific heuristics.

Aydin & Öztemel (2000) investigate an RL-agent control architecture for dynamic scheduling. The Q-learning agent, called Q-III, interacts with a simulation environment and selects the most appropriate priority rule based on the queue size of the machine and mean slack time of the queue. The reward signal is based on literature findings such as the following: the SPT rule is most appropriate when the system is overloaded. Hence, a positive reward is given when selecting the SPT rule in an overloaded system. As a finding from a simple prescinded scenario, better results for the mean tardiness than each static decision rule on its own can be achieved in most test cases.

Wang & Usher (2005) apply Q-learning to a single machine setup. The algorithm solves the dispatching rule problem, and they confirm that the algorithm is able to learn the most appropriate rule, i.e. EDD, SPT, or FIFO, according to three different objectives: minimize the maximum lateness, the number of tardy jobs, and the mean lateness. If the correct rule, meaning the best according to the current state, is selected, the agent receives positive feedback and negative if not. In their previous work, Wang & Usher (2004) investigate the influence of the parameters one can manipulate when applying Q-learning, such as the number of states, reward definition, learning rate, discount rate, exploration-exploitation trade-off, and initial Q-values. Based on extensive computation experiments, recommendations are provided. A similar approach is followed in the present work.

The research of Scholz-Reiter & Hamann (2008) on decentralized production control deals with the task of inventory control at each workstation in a customer order-oriented job shop with non-directional material flows. The goal is an adaptive control system that adjusts the inventory target levels according to external and internal production environment conditions. A simulated annealing metaheuristic is used to calculate the target stock levels and supplemented by pre-trained neural networks, which regulate the material flow at each workstation by choosing the most appropriate next workstation. However, next to the target stock level, no further objectives are considered. The evaluation is performed in a system with 82 workstations and two months of simulation time. This work is part of the German Collaborative Research Centre 637 “Autonomous Cooperating Logistic Processes: A Paradigm Shift and its Limitations”. More work on autonomously controlled production systems and, in particular, the level of autonomy is presented in Scholz-Reiter & Görges et al. (2009). They can show that a high level of autonomous control, defined by the number of decentral decision-making units, increases the performance, although coming at the expense of an increased level of complexity.

Heger (2014), Heger & Branke et al. (2016), as well as earlier results published in Scholz-Reiter & Harjes et al. (2010) investigate the dynamic rule selection for the scheduling problem of complex semiconductor production systems. The motivation is similar to previous authors and addresses the problem that no static heuristic rule outperforms other rules in any case and with respect to any objective (no free lunch theorem). They implement a Gaussian process model to obtain a regression model, outperforming other methods, e.g. neural networks, in previous studies for the dynamic selection of the most appropriate rule and rule-related parameters. As validation example, they make use of Intel’s MiniFab[1] test case (Tsakalis & Flores-Godoy et al. 1997) and, additionally, consider a dynamic flow shop scenario with sequence-dependent setup times. The Gaussian process is pre-trained from various simulation runs and predicts the best-performing rule and, thereby, reduces the mean tardiness. Furthermore, they are able to show that a reduced number of simulation runs is enough for pre-training as no further accuracy improvement of the model performance is achieved.

The work of Chen & Xia et al. (2015) studies the application of a Q-learning algorithm for the load carrier scheduling problem. The carriers supply an assembly line with material and parts and are rewarded for the line throughput as well as their transportation distance,

[1] Kempf, K., *Intel Five-Machine Six Step Mini-Fab Description: Dr. Karl Kempf*. `http://aar.faculty.asu.edu/research/intel/papers/fabspec.html` (accessed on 17.06.2020).

hence, minimizing material handling costs. Improved performance can be achieved by adding forecast information, e.g. product sequence or a list of required parts, to the state vector of the decision agent, comparing to just current buffer fill levels. Their algorithm selects the most appropriate pre-defined heuristic and, eventually, outperforms existing approaches. In particular, the look-ahead feature is promising and investigated in the present work, too.

Freitag & Hildebrandt (2016) further advance the investigation on combined multi-level dispatching heuristics for semiconductor job shops. They develop a multi-criteria optimization approach, which determines an optimized dispatch rule. A genetic algorithm optimizes, according to the two performance targets average lead time and standard deviation of the lead time, the dispatching rule based on a population of possible control rules. The population is created via operators, such as arithmetic operations, constant parameters, and further attributes. The used application scenario is the FAB6 model (Fowler & Robinson 1995) that is available publicly and represents a benchmark test data set of a real-world semiconductor factory with complex material flow characteristics. As a result, they achieve significantly improved performance in terms of just-in-time production.

Next, Niehues (2016) deals with the adaptive control of a job shop. The system is permanently subjected to disruption events and has to be highly adaptive in order to minimize order-related costs. The developed control system includes the identification of disruptive event classes, which aggregate multiple root causes. Furthermore, the disruptions are assessed concerning their effect on the logistical performance. A disruption-class-dependent catalogue of (counter-) measures is created to preserve the initial plan. A re-optimization of the entire order sequence is the last measure, if all other measures are not sufficient to repair the initial plan. The test case is taken from an arbitrary flexible production system for which they are able to improve the profitability, mean tardiness, and predictability of delivery dates.

Shahrabi & Adibi et al. (2017) apply reinforcement learning to the parameter estimation of a variable neighbourhood search that determines a schedule for a dynamic job shop with stochastic job arrivals and machine breakdowns. A Q-learning algorithm is implemented that incorporates a method that evaluates the quality of selected parameters for the neighbourhood search to compute the reward. In each rescheduling event, the algorithm determines a new parameter set according to the actual shopfloor state. After a training period, the performance outreaches standard dispatching rules, such as SPT and the simple variable neighbourhood search without reinforcement learning. However, the flexibility of the algorithm is seen as the most crucial advantage. This result is similar to the objectives of the present research, as showing superior performance in just a single scenario is not satisfying in practice.

Finally, Shiue & Lee et al. (2018) choose an RL-based system for a real-time scheduling approach. The algorithm is based on a Q-learning module that learns to select the best rule out of a set of multiple, pre-defined dispatching rules. They are able to show a better performance than previously implemented single or combined dispatching rule systems.

3.1.2 Applications of Reinforcement Learning to adaptive scheduling and dispatching

The research of the authors considered in this sub-section have in common that they all apply RL-algorithms directly to production scheduling or dispatching, i.e. they differ from the RL-applications in the previous section in such a way that not just a decision rule is determined but the scheduling or dispatching decision itself.

Dietterich & Zhang (1995) and Zhang & Dietterich (1995) present, probably, the first research on the application of reinforcement learning to job shop scheduling. The “NASA space shuttle payload processing” problem, i.e. scheduling different tasks prior to a space shuttle launch while minimizing the total duration of all operations, is investigated, which can be attributed to the class of scheduling problems. Their approach takes an initially generated schedule that, however, violates some constraints and repairs this schedule iteratively. A reward is given for actions that repair the schedule and to the number of iterations needed. They implement a time delay TD algorithm and further enhance it with two features: experience replay and ϵ-greediness. Shortly before their research, the TD-gammon algorithm of Tesauro (1992) showed promising results for the board game backgammon. The same results, i.e. high performance with a lower effort on feature engineering, are shown in the scheduling use case. The TD algorithm is, for instance, 1.8-times faster than a simulated annealing method.

Mahadevan & Theocharous (1998) demonstrate a simulation-based RL-algorithm, called SMART, for the inventory control of a single-product transfer line with three machines. A positive reward is given for satisfied demand and a negative reward for machine repairs in case of breakdowns. Though output and inventory are optimized. Comparing to a Kanban control heuristic, they show a lower inventory level. Additionally, the SMART algorithm allows for the consideration of maintenance measures, because next to the production control policy it also learns a maintenance policy. This is a new result, as it shows the generalization property of learning algorithms. Paternina-Arboleda & Das (2001) continue this work and consider a four machine case with different demand rates. Moreover, they implement ConWIP, basestock, and two boundary control policies as benchmarks, in addition to the Kanban heuristic. The RL-policy outperforms the benchmarks, in particular, for a Poisson demand case.

Another exemplary job shop production system is described by Riedmiller & Riedmiller (1999). There is a single RL-agent that optimizes the summed tardiness of all jobs. They analyse the effect of varying state vectors and show that the Q-learning algorithm can optimize the local dispatching strategy and outperform simple due date, relative slack, and waiting time heuristics. The multi-layered neural network for representing the value function shows the generalization capability, as the trained network could be applied to another setup successfully.

Stegherr (2000) is one of the first to investigate a multi-agent RL-system for the scheduling of a variant series production, using the example of a semiconductor backend production. The research follows a multi-criteria objective function that considers inventories, throughput times, capacity utilization, and on-time delivery. First, state and reward definitions of individual agents are derived for every single objective and then validated in a simplified production system. Afterwards, they are successfully applied in the semiconductor use case. The multi-objective consideration of the present work is based on this comprehensive work.

Csáji & Monostori et al. (2006) translate a market-based production control system into the application context of cyber-physical production systems. They show an example of a successful implementation and testing of the decentralized agent architecture presented by (Monostori & Csáji et al. 2004). The investigations by Csáji & Monostori et al. (2006) reveal a three-layered control architecture. It incorporates simulated annealing on the highest level to control the exploration rate. Next, there are several RL-algorithms for so-called order and resource agents to optimize the schedule for the maximum completion time. On the third layer, the value function is approximated with a neural network.

Gabel & Riedmiller (2008), Gabel (2009), and Gabel & Riedmiller (2012) extend the research on scheduling and multi-agent systems. Each agent represents a machine and decides sequentially which order to process next. In this simple setup, they conclude that learning methods surpass simple heuristics and reach even a near-optimal performance in some test instances. In their further research, they examine both policy- and value-based RL-algorithms and a setup in which agents can communicate with each other to achieve the best possible schedule. The investigated application is, in all cases, fictitious or based on established scheduling benchmarks. Their insights on RL-algorithms are considered in this work, though the scheduling problem setup differs from the order dispatching task.

Shah & Gosavi et al. (2010) describe a Q-learning algorithm for the application to a remanufacturing production system. The decision whether to use remanufactured or as-new components in the production process is considered as complex decision that is hard to solve for large-size real-world problems. However, their work results in the trivial conclusion that there is a cost advantage over non-use of remanufactured materials. Hence, their research rather focuses on a new application area instead of an in-depth algorithm analysis.

Zhang & Zheng et al. (2011) propose a SARSA(λ, k)-algorithm for the scheduling of final quality tests of a semiconductor manufacturer. SARSA(λ, k) uses λ as decay and k as cutoff parameter. The objective is to optimize the throughput of the testing work centre. They achieve superior performance in comparison to industrial benchmarks and simple heuristics.

Wang & Li et al. (2012) address the stochastic economic lot scheduling problem for a single machine and three products case. They implement an adaptive control based on a Q-learning algorithm that optimizes the long-term cost. The agent decides what to produce and thereby influences the setup, holding, and backorder cost. After the learning phase, the agent surpasses two benchmark policies in all test cases. The benchmarks are the so-called Brownian and the FIFO policy.

Arviv & Stern et al. (2016) present a collaborative reinforcement learning approach for the scheduling of the job transfer on fixed parallel tracks of identical jobs in a multi-machine flow shop. Two agents are considered. Moreover, the level of collaboration and information sharing are investigated for four levels between the two extremes of full collaborative learning and information sharing to no exchange and full independence. They develop a dual Q-learning algorithm that optimizes the makespan and assigns the reward to the agents. This is just one example of a much broader literature on collaborative and multi-agent systems that is, however, not thoroughly considered in the present work.

Wang & Wang et al. (2016) investigate a resource-constrained flow line system with two machines and a finite buffer in between, and consider the maintenance scheduling problem. The problem is modelled as MDP and solved by a multi-agent RL-approach. They consider a distributed RL-algorithm taken from the work of Das & Gosavi et al. (1999) and Mahadevan & Marchalleck et al. (1997). The objective and reward are based on the cost related to maintenance actions and the number of defects. They show that the cost-sharing approach for both decision agents reveals better overall results than independent RL-agents.

The latest challenges and characteristics of job shop manufacturing in the semiconductor industry are comprehensively summarized in Waschneck & Altenmüller et al. (2016). Subsequently, Waschneck & Reichstaller et al. (2018) and Waschneck & Altenmüller et al. (2018b) train multiple DQN-agents in a semiconductor factory simulation. Each agent represents a different work centre, such as lithography, implant, or furnace, that have individual characteristics and, therefore, require an individual scheduling policy. The scheduling considers the decision of the next processing order, on the events of arrival and moveout of an order. The reward is derived from the uptime utilization, and penalties are given for unfeasible actions. However, after a training phase of two days, the heuristic benchmark cannot be surpassed in all test cases. This work is close to the present work and motivates a more in-depth investigation into the capabilities of advanced RL-algorithms.

Echsler Minguillon (2020) investigates the trade-off between central and decentral PPC with respect to the robustness in an agile production system setup taken from the automotive industry. On a central level, an initial schedule is determined that contains a defined slack to cover an expected range of disturbances. A decentral control system is implemented that is based on the RL-algorithm of Gabel & Riedmiller (2008) and comes into play when a disturbance happens that requires rescheduling. The algorithm learns which actions to perform to repair the schedule. For an improved performance of the decentral rescheduling after a certain training period, the slack on the central level can be reduced.

3.1.3 Advanced applications of Reinforcement Learning not related to manufacturing

One objective of the present work is to make use of transferable results from RL-application domains that are not related to production scheduling or dispatching. Most RL-applications are currently found in the area of robotics and strategic games. Therefore, the last section of the literature review elaborates on the most relevant applications of reinforcement learning that are not related to the context of industrial production.

The dispatching of parallel elevators, according to Crites & Barto (1995) and Crites & Barto (1998), is an early but still well-known example that gives a vivid illustration of reinforcement learning. Manually designed heuristic decision rules usually perform the dispatching of elevators. In their work, they apply Q-learning to a real-world problem setting with four elevators. Because of the large number of possible states, i.e. more than 10^{22} states, they chose a neural network to approximate the value function. Moreover, they compare different modelling approaches, such as training one neural network for all elevators or a separate network for each elevator, as well as the modelling choice, which state information to provide to the decision agent. These basic considerations are similarly relevant in the application of the present work.

Other scheduling examples are the scheduling of multiple cranes in a container terminal by Zeng & Yang et al. (2009) and the project management problem investigated by Wauters & Verbeeck et al. (2011). Qu & Wang et al. (2016) develop a multi-agent Q-learning algorithm for the adaptive workforce management.

Additionally, Mnih & Kavukcuoglu et al. (2013) introduce an advanced DQN algorithm and apply it to computer games. The algorithm is able to learn policies from a high-dimensional state input, i.e. a screen pixel representation. Moreover, the algorithm shows superior performance for several games without changing the algorithm significantly.

The latest research by Silver & Schrittwieser et al. (2017) and Silver & Hubert et al. (2018) shows the enormous potential of RL-algorithms to reach an expert level of human performance in board games, without providing any prior domain knowledge as training data. Their “self-learning” approach is based on a simulated real-world application and motivates the research of the present work to a large extend.

3.2 Research deficit

Table 3.1 highlights the just presented research that feeds into the present work and categorizes them according to the focus areas presented in the first part of Section 3.1. From this overview, it can be seen that there are both: prior work dealing with the control of job shops and those in which reinforcement learning has been successfully implemented. All in all, the research deficit can be summarized as follows:

- Basic heuristics provide control systems that require much manual adaption effort in case of external or internal changes.
- Most existing RL-based PPC approaches implement a state-dependent rule selection or parametrization, which means that adaptive decision-making on a dispatching decision-level is not sufficiently realized.
- A real-world-sized production application, especially considering the integrated order dispatching for transportation and processing and multiple stochastic processes, has not been considered for reinforcement learning. Moreover, multiple objectives pose an additional challenge, hardly addressed so far.
- The analysis of the plausibility of the agent’s behaviour as well as the comparison of alternative RL-agent designs and its direct comparison with benchmark algorithms have not yet been carried out.
- A methodical approach to the application of reinforcement learning in the context of PPC applications does not exist and, often, just single application scenarios are presented.

Table 3.1: Overview of relevant research for the adaptive order dispatching by means of reinforcement learning.

	System scope				**Production control task**				**Performance objective**				**Optimization approach**			
Legend: ● considered ◐ partially considered ○ not considered	Job shop production system	Many variants with high volume	Complex material flow	Stochastics (e.g. breakdowns)	Dispatching (transport)	Dispatching (processing)	Adaptive production control	Real-time decision-making	Multi-objective optimization	Dynamic perform. evaluation	Order-related KPIs	Resource-related KPIs	Reinforcement learning	Simulation-based training	Plausibility of RL-behaviour	RL-modelling alternatives
Dynamic selection and parametrization of dispatching rules																
Kim 1998	●	○	○	●	○	●	◐	●	○	●	○	●	◐	●	○	○
Aydin 2000	●	○	○	●	○	●	○	●	○	◐	○	●	●	●	○	○
Wang ’04, ’05	○	○	○	●	○	●	○	●	○	●	●	○	●	●	○	●
Scholz-Reiter ’08, ’09	●	○	○	●	○	◐	◐	●	○	●	◐	◐	○	○	○	○
Heger ’14, ’16	●	●	●	●	○	●	○	●	●	◐	●	●	○	●	○	○
Chen 2015	○	○	○	●	●	○	○	●	●	●	●	●	●	●	○	○
Freitag 2016	●	●	●	●	○	●	○	●	●	◐	●	●	○	○	○	○
Niehues 2016	●	○	○	●	○	●	○	●	●	○	●	●	○	○	○	○
Shahrabi 2017	●	○	○	●	○	●	○	●	○	○	◐	◐	●	●	○	○
Shiue 2018	●	○	○	●	○	●	○	●	○	●	◐	◐	●	●	○	○
Applications of Reinforcement Learning to adaptive scheduling and dispatching																
Zhang 1995	●	○	○	●	○	●	●	●	○	●	◐	◐	●	○	○	○
Mahadevan 1998	○	○	○	●	○	●	●	●	○	●	◐	◐	●	●	○	○
Riedmiller 1999	●	○	○	●	○	●	●	●	○	●	●	○	●	●	○	○
Stegherr 2000	○	●	○	●	○	●	●	●	◐	●	●	●	●	●	○	○
Paternina 2001	○	○	○	●	○	●	●	●	○	●	○	●	●	●	○	○
Monostori ’04, ’06	○	○	○	●	○	○	●	●	○	●	○	●	●	●	○	○
Gabel ’08, ’09, ’12	●	○	○	●	○	●	●	●	○	●	○	●	●	●	○	●
Shah 2010	○	○	○	●	○	○	●	●	○	●	○	●	●	●	○	○
Zhang 2011	●	●	●	●	○	●	●	●	○	●	○	●	●	◐	○	○
Wang 2012	○	○	○	●	○	●	●	●	○	●	○	●	●	●	○	○
Arviv 2016	○	○	○	●	●	○	●	●	●	●	●	●	●	●	○	○
Wang 2016	○	○	○	●	○	○	●	●	○	●	○	●	●	●	○	○
Waschneck 2018	●	●	●	●	○	●	●	●	○	●	○	●	●	●	○	○
Echsler 2020	◐	○	◐	●	○	●	●	●	○	●	●	○	●	◐	○	○
Advanced applications of Reinforcement Learning not related to manufacturing																
Crites ’96, ’98	○	○	○	●	○	●	●	●	○	●	◐	○	●	●	○	◐
Zeng 2009	○	○	●	●	●	●	◐	◐	○	●	○	○	●	●	○	○
Wauters 2011	○	○	○	●	○	●	●	●	○	●	◐	◐	●	◐	○	○
Mnih 2013	○	○	○	◐	○	○	●	●	○	●	○	○	●	●	○	○
Qu 2016	○	○	○	●	○	○	●	◐	●	●	●	●	●	◐	○	○
Silver ’17, ’18	○	○	○	◐	○	○	●	●	○	●	○	○	◐	●	◐	○

4 Adaptive order dispatching based on Reinforcement Learning

This chapter describes the methodological approach developed in this work to achieve the objectives set in Section 1.3 and to address the research deficit identified in Section 3.2. The overall aim of this research is to develop a method for adaptive order dispatching. This is investigated by implementing a reinforcement learning algorithm and, in doing so, it focuses on the modelling and in-depth evaluation of it to increase the applicability of this rather new type of machine learning algorithm (see Figure 4.1). Eventually, the outcome of the method is a data-driven, adaptive order dispatching system, which enables the real-time control of a job shop manufacturing system. The data-driven approach enforces the vision of developing high-performing control systems autonomously with as less manual effort and domain expertise as possible. So, the transferability to further applications is an essential requirement that motivates this research, too.

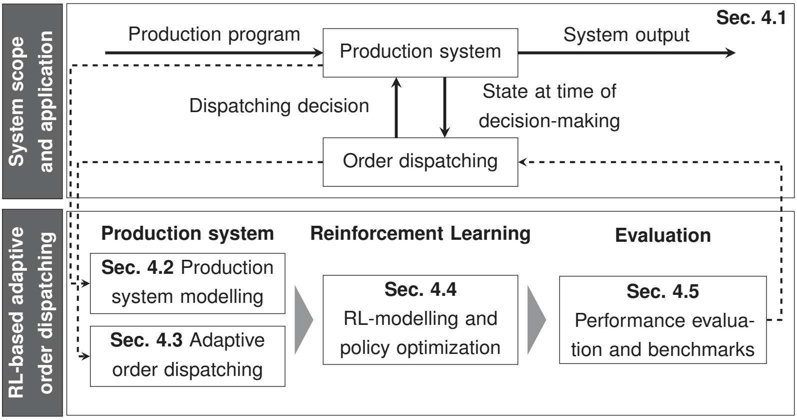

System scope and application
Sec. 4.1
Production program
Production system
System output
Dispatching decision
State at time of decision-making
Order dispatching
RL-based adaptive order dispatching
Production system
Reinforcement Learning
Evaluation
Sec. 4.2 Production system modelling
Sec. 4.3 Adaptive order dispatching
Sec. 4.4 RL-modelling and policy optimization
Sec. 4.5 Performance evaluation and benchmarks

Figure 4.1: Methodological approach.

According to the problem solving approach of Domschke & Drexl et al. (2015) introduced in Section 2.2, Section 4.1 frames the problem scope and most important assumptions. Next, the description as formalized model is separated into two phases. First, the production system is modelled to be implementable as a virtual training environment (digital twin) in form of a discrete-event simulation (see Section 4.2). Second, the overall adaptive production control architecture is described (see Section 4.3). After that, the RL-algorithm is presented in Section 4.4 by describing the model-free, policy-based optimization algorithm, the state and action representation, the reward function, and the relevant hyper-parameters. Finally, the evaluation approach, analysis of plausibility, and benchmarking are covered in Section 4.5. The application of the herein outlined method is part of Chapter 5.

The method is based on the research of the author. Supporting work is based on the following student theses that were written under guidance of the author: (A_Schäfer 2018), (A_Behrendt 2019), (A_Hettich 2019), (A_Kaiser 2019), (A_Theiß 2019), and (A_May 2019). In addition, the problem setup and adaptive control architecture are published in the paper by Stricker & Kuhnle et al. (2018). Recommendations for the modelling and design of reinforcement learning are considered in Kuhnle & Röhrig et al. (2019) and Kuhnle & Schäfer et al. (2019). Moreover, the detailed investigation of RL-modelling alternatives is summarized by Kuhnle & Kaiser et al. (2020).

4.1 Scope and assumptions

The scope of this work was, to a certain extent, already narrowed down in Chapter 2. The following assumptions further specify the scope of the approach:

1. An existing job shop manufacturing system with a fixed layout is the research object. Inside the system, orders are processed and process data and performance measures are continuously monitored. Processes outside the system are not considered, except the generation of production orders that are fed into the system. This is performed based on a predefined probability distribution (e.g. production program).

2. The order dispatching is the focused PPC-task. The number of dispatching agents is fixed to a single agent type. However, multiple agents of the same type are in principle applicable.

3. A discrete-event simulation represents the training environment and is automatically created and parametrized based on a structured list of input parameters. The production process logic is implemented in the simulation as described in this section. Hence, the simulation model is generic to a certain extent.

4. The learning process starts directly on instantiation of the simulation model and is uncoupled from the real production system. The RL-agent's initial behaviour is completely random. The current production system state is known at any time. These assumptions are reconsidered in Chapter 5 and Chapter 6.

5. Learning is completed when a state of convergence is reached. Afterwards, the trained agent can be applied and deployed as adaptive controller. However, as the real-world application is not part of this work, no validation is provided for the deployment.

6. Without actually applying it in real-world, it is assumed for the deployment that whenever a trained and converged RL-agent is not valid or not accurate enough any more, e.g. due to structural changes of the real-world system, a new simulation-based training phase starts from scratch, considering the latest system parameters (see assumptions 3 and 4 above).

4.2 Production system modelling and virtual training environment

This section provides the foundation for all subsequent steps, such as training and evaluation of the adaptive control system. A discrete-event simulation framework is described that is required because of three reasons: First, generating sufficient training data in terms of quantity would require roughly two years of data collection in a real-world production system. Next, pure data-driven training methods need to try out invalid actions or actions with negative consequences, as their consequence is only learned based on the reward signal. Usually, negative consequences impede a real-world application. Third, all stochastic influencing factors need to be controllable for an accurate comparison and benchmarking of different control algorithms, which is not possible in reality. Moreover, following Section 2.2.4, simulation-based training and evaluation environments are predominant in various reinforcement learning applications.

The section starts with a description of the scope, most relevant elements (see Section 4.2.1), and control structure (see Section 4.2.2) that make up the simulation model. This follows the standard procedure of simulation in production and logistics according to VDI 3633 Part 1 (2014). The simulation is capable of representing a job shop with complex characteristics (see Section 4.2.3). However, no claim is made to the completeness of the modelling. Finally, a validation of the simulation model is essential and covered in the next chapter.

Purpose of system modelling The system scope covers all production resources, such as machines and transport resources, and the process control structure. The control structure is manifested in decision-making units, also called *agents*, that are linked to resources and determine actions. So, the following three properties characterize the simulation framework:

- *Time-invariant properties*: production layout (e.g. job shop, flow-line), number of resources (e.g. machines, buffers), and fixed properties of the resources (e.g. capacity).
- *Time-variant processes*: machine breakdowns, repair process, varying processing times, and mix of product variants.

- *Control and decision-making units*: decision alternatives and responsible areas of decision-making (e.g. single machine, machine group) as well as decision-making algorithm (e.g. heuristic, reinforcement learning).

The system model covers the entire material and information flow within the borders that are defined by the input and output of production orders. Incoming orders contain information, e.g. regarding the processing machines and lot size, and trigger production processes or transportation processes. After processing, the orders leave the system and order-related information, e.g. waiting and processing times, are recorded.

In order to be able to extend the scope of a single production system to multiple sub-systems (see scope highlighted in Figure 2.2), being part of a factory plant or even a production network, one either widens the scope of the system model to that level, i.e. order dispatching covering the material flow within a production network, or defines separate models for each sub-system as long as no interaction needs to be considered between those. Building a simulation based on this framework, shall enable a broad application for various production system setups, different types of decision-making algorithms, and even different production control tasks.

4.2.1 Organizational structure

The organizational structure of the simulation model fulfils the requirement of a generic approach, as just stated. Therefore, the model is designed in a way to ensure the transferability to different types and structures of production systems as well as the parametrization according to changing real-world system parameters.

Next to production orders, four types of resources are part of the organizational structure of the simulation framework (see Figure 4.2): processing resource or machines, transport resources, sources, and sinks. Orders are not defined as resources, because they represent physical and informational objects that do not take actions or change the state of other resources. The resources are described in the following, along with the relevant decisions.

Machine resources perform the value-adding production processes that are required for each order. A machine contains a single processing unit. Parallel processing is not considered. Batching is also not part of the present work. The processing time is not fixed and, in fact, the production order specification (“recipe”) determines the processing time. It is exponentially distributed and the distribution is truncated by a minimum and maximum boundary value. The exponential distribution is chosen to represent the large processing variance, as an order with just a single wafer and a simple processing operation takes far less time than an order with over 20 wafers and a long processing operation (Waschneck & Altenmüller et al. 2016).

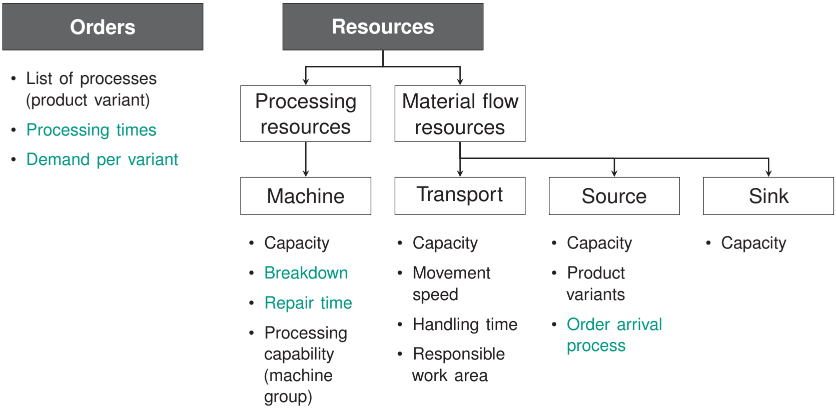


Figure 4.2: Categorization of production system components, i.e. resources and orders, and relevant parameters. Stochastic parameters are highlighted.

Furthermore, machines contain an inbound (pre-processing) and outbound (post-processing) buffer with parametrizable capacities (see Figure A2.1 of Appendix A2). A decision-making unit selects the next order out of the inbound buffer, i.e. performs the order sequencing. Here, the FIFO heuristic is assumed as no setup times or order-specific due date are explicitly considered. Moreover, the FIFO sequencing heuristic supports the learning process of the order dispatching agent because the sequence given by the order dispatching is processed in precisely that way. Thereby, "noise" in the training data, e.g. by rearranging the order sequence, is reduced.

Decision-making of machine resources: In which sequence should the available orders be processed?

The availability of machines is not granted and machines require maintenance from time to time or break down. The required maintenance activities and breakdowns are considered together and follow a single exponential distribution with an occurrence rate of $\lambda = \frac{1}{MTBF}$ (MTBF, mean time between failures). The required time to repair and setup the machine is called mean time off line (MTOL) and follows an exponential distribution, too, with a rate of $\lambda = \frac{1}{MTOL}$. Again, the distribution assumption is backed by the standard procedures of semiconductor operations described by Mönch & Fowler et al. (2013) and SEMI E10-0304 (2004). It is given that machines break down after an operation has finished and not within, as machines recognize failures based on process and product tolerances that are evaluated after every operation.

Furthermore, machines are sub-organized into *machine groups*. Machines in the same group are technically capable of performing the same processing operation. That is also why machine setups do not need to be considered explicitly. The machine group definition is an essential characteristic of the present work and will be discussed in Chapter 5.

The product quality and scrap rates are not considered. Given the fact that the primary application is semiconductor manufacturing with sophisticated in-process quality control and complete end-of-line inspection, scrap is handled separately and not in the scope of order dispatching within a work centre.

Transport resources can represent either a fixed, e.g. conveyor belt, or a loose transportation resource type, e.g. automated guided vehicle or forklift. In both cases, the responsible work area, i.e. the resources that can be reached by the transport resource, is defined by a two-dimensional connectivity matrix and, analogously, a distance matrix determines the distance. By dividing the distance by the movement speed the transport time is calculated. Moreover, handling times need to be considered for picking up and dropping orders. The capacity, i.e. the number of orders that can be transported at the same time, is limited and set to one. The availability is not restricted, as it is assumed that enough vehicles or forklift operators are available also during breaks and in between shifts. To determine which order is transported next, each transport resource has a decision-making unit assigned to it. In the present work, the transport resource is considered as the key unit for the adaptive order dispatching.

Decision-making of transport resources: Which order should be transported next, i.e. from where to transport an order, and to which machine should the order be transported?

Source resources create and release new production orders based on an arrival process and feed them into the system. A set of product variants is assigned to each source that defines which variants arrive at this respective source. Machines can be assigned to multiple sources, and the assignment may not necessarily be the closest source. The product variant demand is based on an empirical distribution represented by “orders per day”-probabilities. The term product variant, as used in the present work, is defined below.

Similar to a machine, a source contains a limited outbound buffer, in which the released orders are placed. The arrival time of new orders is exponentially distributed with a mean arrival rate of $\lambda = \frac{1}{\text{mean arrival time}}$. However, the rate can also be not restricted and then orders arrive whenever a free buffer slot is available. The exponential arrival process is assumed as the input of orders from external suppliers or previous work centres varies due to changing product mixes and batch processes that cause *inventory-waves*. Moreover, it is in line with the assumptions in Mönch & Fowler et al. (2013). As there is a predefined order release interval and the released variants are randomly chosen, no further decision needs to be made at the source.

Sink resources store orders that finished processing and are defined by an inbound buffer capacity. In the following, the sink is modelled with an unlimited capacity. No decision needs to be made at this resource type.

Production orders are physical and information-containing objects that define their process flow and specify which machine types and operations are required for processing (“recipe”). The sequence of machines is defined as determination criteria for the *product variant*, and the customer demand is defined per product variant, as described above. Hence, process times are defined on an order-level. Any process auxiliaries are neglected in the present work. Production orders itself do not make any decision.

Note that buffer resources are not explicitly considered, because the definition of machine, source, and sink resources already include internal buffers.

Parametrization of fixed and transportation resources All not-moving resources, i.e. machines M, sources SO, and sinks SI, define the set of fixed resources $\mathcal{R}$ that represent an array:

$\mathcal{R}$:	SO_1	$\ldots$	SO_{nSO}	M_1	$\ldots$	M_n	SI_1	$\ldots$	SI_{nSI}

The relevant parameters and value ranges are summarized in Table A2.1 of Appendix A2. The list of parameters are resource-specific features and the number of resources can be adjusted by adding, for instance, a machine to the set of machines and extending the array.

Loose transportation resources are defined as two-dimensional, symmetric distance matrix based on the length of the resource array $\mathcal{R}$:

$d_{i,j}$	SO_1	$\dots$	$SO_{n_{SO}}$	M_1	$\dots$	M_n	SI_1	$\dots$	$SI_{n_{SI}}$
SO_1	0	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
$\dots$	$\dots$	0	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
$SO_{n_{SO}}$	$\dots$	$\dots$	0	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
M_1	$\dots$	$\dots$	$\dots$	0	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	0	$\dots$	$\dots$	$\dots$	$\dots$
M_n	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	0	$\dots$	$\dots$	$\dots$
SI_1	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	0	$\dots$	$\dots$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	0	$\dots$
$SI_{n_{SI}}$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	0

The layout of the considered production system, the possible transportation routes, and restricted work areas can be modelled comprehensively by this two-dimensional matrix. It defines the distance $d_{i,j}$ and infeasible transportation links by the value of -1. Further characteristics of the transportation resource have already been introduced before and the list of parameters is stated in Table A2.1 of Appendix A2.

The modular simulation framework is primarily focused on the application to job shop manufacturing systems. However, the framework is able to represent a flow line system, too. Furthermore, multiple transport resources are implementable with the restriction that every resource performs the same dispatching task.

Definition of operation modes Finally, Figure 4.3 defines and summarizes the most relevant operation modes and operation times that are covered in the production system model and that are related to the resources described so far. Note that the time axis and width do not match real figures.

The definitions are ajar to the concept of overall equipment effectiveness (OEE), which is established in operations management (Wiendahl 1997). Moreover, semiconductor operation standards define these, too (SEMI E10-0304 2004). The figure links different operations at production orders, transport resources, and processing resources. For example, the processing time of the machine is equivalent to the period in which the order is in the mode "processing". Loading, transporting, and unloading operations together form the total transport operation. The performance indicators such as the uptime utilization of machines, the order cycle time, and transport resource utilization, which are introduced in Section 4.5.2, are based on Figure 4.3, too.

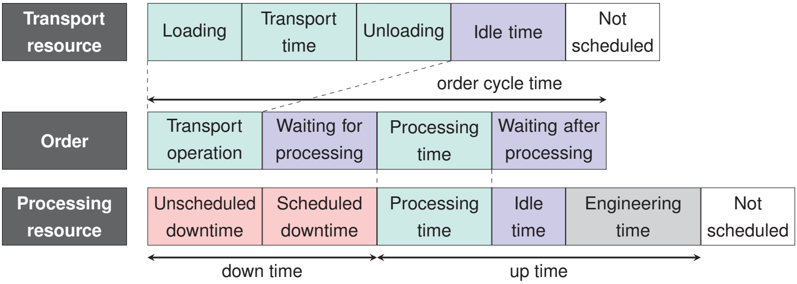


Figure 4.3: Definition and relationship of operation modes and times of production orders, transport resources, and processing resources (adapted from Wiendahl (1997) and SEMI E10-0304 (2004)).

4.2.2 Process control structure

So far, the organizational structure of the production framework was outlined. This sub-section focuses on the process structure and how the process flow is controlled. It thereby takes the production system and its elements as given, i.e. changing the aforementioned organizational structure is not considered.

Figure 4.4 highlights the interlinkage of relevant PPC-tasks, taken from the fundamentals in Section 2.1, and depicts the order process flow as a process flow chart. As already mentioned, order dispatching is the decision of interest in the present work.

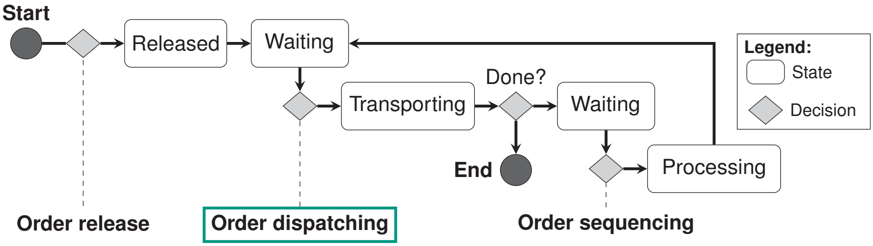


Figure 4.4: Order process flow chart with interfaces between order flow and the PPC-decisions order release, dispatching, and sequencing.

When an order is released, it is assigned to a source buffer and waiting for transportation or processing. At that moment, the order is already appointed to a target machine for the next processing operation. The assignment is modelled by a stochastic demand distribution in the order generation process, as described above. However, in real-world it can be also based on a schedule that assigns orders to machines and thereby manages the material flow between work centres and minimizes machine setups and idle times.

When a transport resource is available, the order is moved to the next processing machine and waits there for processing. The order sequencing is based on a simple decision rule, which selects the next order out of the available orders in the inbound buffer. As stated before, the decision rule is a deterministic FIFO heuristic in order to prevent side effects that might influence the investigations of the RL-based order dispatching negatively. After processing, the order is placed in the outbound buffer and is again waiting to be dispatched. If no further processing is required, the final dispatching brings the order to a sink.

Note that the next machine assignment can be adjusted by the order dispatching to another machine that is in the same machine group as the initially planned machine, as stated in the previous section. This might be reasonable due to the levelling of processing capacities or temporarily unavailable machines. As a consequence, the dispatching decision has a twofold character: first, which order to dispatch next and, second, where to dispatch it. Hence, it takes a focal role within the process control framework with a significant impact on the performance.

Order dispatching decision subsets For the order dispatching decision, distinctive decision subsets are conceivable given the already defined organizational structure and order process flow. These are the following (see also Figure 4.5 and Figure A2.1 of Appendix A2):

- $Source \rightarrow Machine$: Order dispatching from a source to a machine buffer.
- $Machine \rightarrow Machine$: Order dispatching from a machine's outbound buffer to another machine's inbound buffer.
- $Machine \rightarrow Sink$: Order dispatching from a machine to a sink buffer.
- $Empty$: Moving empty-handed, i.e. transporting no order, to any resource location.
- $Idle$: Idling, i.e. not dispatching an order and not changing the current location.

These five decision sets reduce the total number of possible decisions to just the relevant ones. For instance, dispatching from a source to a sink is not reasonable in the given setting. However, note that the last two sets are not necessarily required to fulfil the dispatching task but both broaden the scope and degree of freedom for the dispatching agent, what is particularly analysed in the computational experiments of Chapter 5.

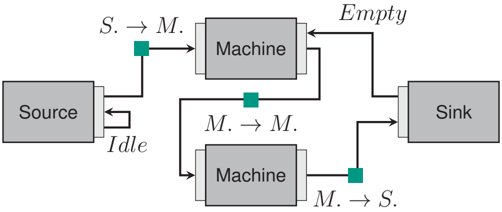


Figure 4.5: Schematic illustration of five order dispatching decision subsets.

4.2.3 Categorization of complexity drivers

The order dispatching decision problem faces two decisions simultaneously, i.e. which order should be transported next and to which machine should the order be transported. Moreover, multiple objectives are considered, as resource- and order-related objectives are relevant. Hence, the decision problem is harder to solve than simple order sequencing or order release problems.

Furthermore, the production processes are subjected to multiple interdependencies and stochastic influences. These are the following:

- *Processing resources*: Unknown breakdowns and repair operations reduce the availability of resources on a considerable scale.
- *Production orders*: The production program varies due to a stochastic product mix, stochastic processing times, and the uncertainty concerning the processing machine within the machine group. These result in an order list with widely distributed processing times and, potentially, unequally utilized resources.

Hence, order dispatching in such a highly dynamic environment has to take into account these complexity dimensions adequately. The present work uses an adaptive and RL-based algorithm to find a control policy that is able to capture the complex system characteristics. The necessary procedure is outlined in the following section.

4.3 Adaptive order dispatching architecture

After explaining the concepts behind all system elements, the system architecture as a whole is described in this section. The conceptional model is translated into an architecture implemented in a software environment and considers two concepts: modularity with standardized interfaces and sequential event-based processing. The latter ensures the applicability and transferability of the underlying MDP concept.

4.3.1 Formal control architecture

Figure 1.1 illustrated the control-theoretic concept behind the overall approach of this work. Figure 4.6 gives an aggregated and schematic overview of the event-based control architecture that ensures a continuous production process. Initially, the production framework is running (Step 1), which means that, according to the process control structure explained in Section 4.2.2, orders are released, dispatched, sequenced, and processed and machines break down from time to time. For every event it is evaluated whether it causes a state change that is relevant for the decision-making unit, e.g. the RL-agent making dispatching decisions (Step 2). Whenever the state changes, all resources are reactivated and asked for a new action (Step 3).

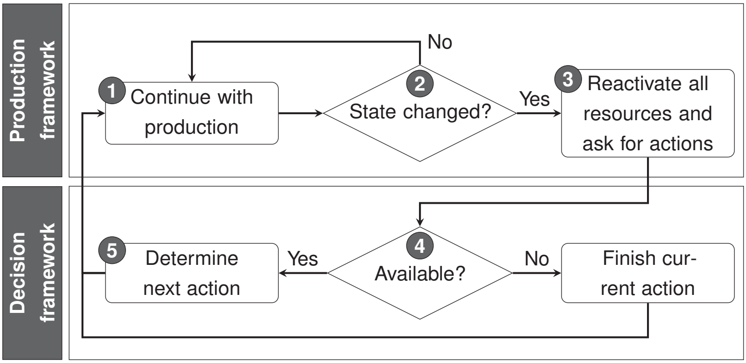


Figure 4.6: Schematic control flow chart of the production and decision-making framework to ensure a continuous production process.

The framework in the lower part of Figure 4.6 represents the decision units of the resources, such as the transport or machine resources. If they are available at the moment, i.e. not performing another action, they are reactivated and asked for a decision (Step 4). Afterwards, they determine their next action (Step 5), e.g. transport an order to a machine. These actions are transferred to the production framework and directly executed (Step 1). Eventually, the state of the production system is changed and whenever an action is finished all other waiting and newly available resources are reactivated again, as new actions might be requested. When no feasible action is available, the respective resource waits until the next reactivation. Actions once taken, are not aborted or reconsidered.

The decision framework is requested in any case a decision needs to be taken by a resource. Either a heuristic decision algorithm or an RL-algorithm can be applied to determine the decision. The procedure for both is similar, as the single difference is that the heuristic does not undergo a learning process. Hence, the interface towards the decision algorithm is standardized and requires the following three elements:

- **State**: An adaptive decision-making derives the decision upon the latest state. Thus, the state information includes, for instance, the buffer fill level, order waiting time, remaining processing time, machine availability, or available dispatching orders. This information is provided as binary or numerical values.
- **Action**: The output of the decision agent is a decision action that is also encoded as a numeric value. The numerical value encodes the action and is looked-up in a pre-defined table that defines how the action value needs to be interpreted.
- **Reward**: At least for RL-agents, a reward is required for any action, as the agent maximizes its reward in order to learn the optimal policy. The reward is, again, a numerical function value and calculated based on the action and state.

4.3.2 Order dispatching decision-making based on Reinforcement Learning

The RL-based decision agent of the transport resource determines the order to be dispatched to a destination. The entire procedure is illustrated in Figure 4.7. By means of the above-described framework, first state S_t is derived in the state module. The agent selects, based on its policy and S_t, an action A_t by calling the method $act(S_t)$. Action A_t encodes two information values: the $order$ to be transported and the $destination$ resource. An *action-mapping*-procedure in the action module is applied to translate the numerical action value into the order and destination information, which can be interpreted by the production framework. The reward R_{t+1} is calculated in the reward module for S_t, A_t, and S_{t+1} and transferred to the RL-agent module for optimizing the policy by calling the method $observe(R_{t+1})$. This method updates policy π according to the optimization algorithm, which will be explained in Section 4.4.

Whenever an action, i.e. a pair of order and destination, is invalid, a recursion is performed, i.e. the agent is called recursively and a new action is requested for the same state. This is repeated until the agent selects a valid action or the maximum number of recursions is reached. The recursion limit is a customizable parameter. If the recursion limit is reached, the resource performs an *idle*-action with a fixed, pre-determined idle time. The purpose of this is to have enough time passed by for a production state change to happen and, so, the RL-agent will decide the next time based on a new state.

It can be deduced from computational results that the learning speed of an RL-agent can be increased significantly by introducing this recursion procedure in case of invalid actions

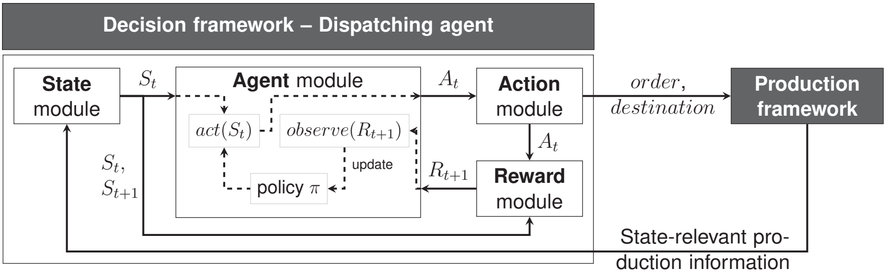


Figure 4.7: Decision-making process flow of the dispatching agent to select the next action to be performed.

(see Section 5.2). However, a too long forced idle time in case of the maximum recursion limit results in both: a significantly longer learning process and a lower average performance. Furthermore, the recursion limit and forced idle time parameter prevent the agent from getting trapped in a loop of choosing invalid actions over and over again, which is a known issue for ill-designed RL-agents (Sutton & Barto 2018).

Relation between MDP and RL-policy update Figure 4.8 depicts schematically the interface and interplay between the MDP, which is implemented as simulated production environment, and the update of the RL-policy during the training process. The policy is represented as ANN, and the weights of the network are adjusted continuously according to the state, action, and reward values derived from the interaction with the environment. The update is the actual optimization of RL-algorithms.

Furthermore, one can check whether the herein considered architecture fulfils the MDP definition (see Section 2.3.2). First, the sequential event-based production and decision framework supports the applicability of the basic MDP concepts. Second, to fulfil the Markov property, all information of the stochastic processes, such as breakdowns of machines and processing times, need to be represented in the state representation to allow an estimation of the transition probability function p and a prediction of the subsequent state and reward. However, this is hardly practicable in real-world applications due to the multitude of influences. Therefore, Sutton & Barto (2018) call that case an approximation of the state representation to the Markovian state representation. Still, they stress that reinforcement learning can be successfully applied. Moreover, Chapter 3 showed state-of-the-art applications that are similar to the application of the present work. So, it is assumed in the following that without strictly proving the Markov property, reinforcement learning can be applied to the presented system modelling.

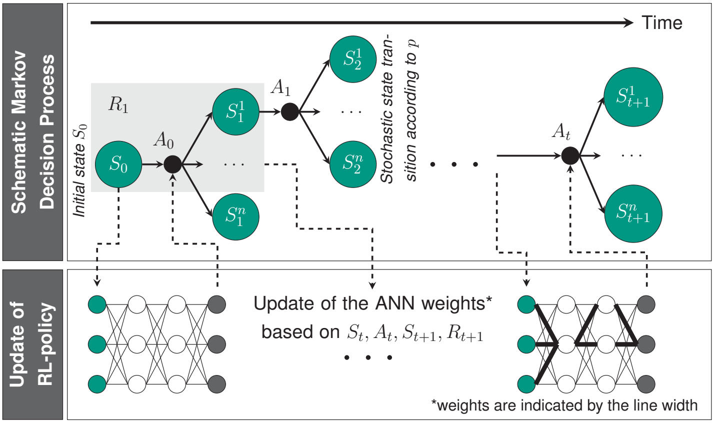

Schematic Markov Decision Process
Time
Initial state S_0
R_1
S_0
A_0
S_1^1
S_1^n
A_1
S_2^1
S_2^n
Stochastic state transition according to p
A_t
S_{t+1}^1
S_{t+1}^n
Update of RL-policy
Update of the ANN weights* based on $S_t, A_t, S_{t+1}, R_{t+1}$
*weights are indicated by the line width

Figure 4.8: Relation between a Markov Decision Process and the updated RL-policy over the training process.

4.3.3 Learning and operation phase

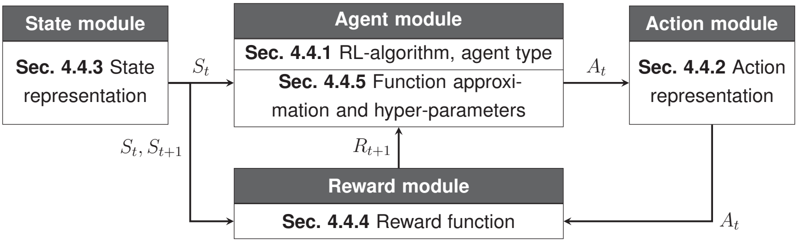
Two phases are separated when deploying the RL-based adaptive order dispatching in practice (see also Figure 4.1). First, in the *learning* phase, the RL-agent is trained in the simulation environment that represents the real production system (digital twin). Second, after the training when the performance is on a level that is acceptable for application, the trained agent can be applied in real-world *operation*. Without loss of generality, both phases are feasible with the above presented adaptive control architecture. Hence, no further restrictions apply for a deployment of the framework.

In addition to that, there are two alternative deployment modes: the training, i.e. policy update, is either continued based on the data generated in reality or the training stops and the policy is just applied. The former pursues the adaptive nature of the algorithm and continues to adjust the performance according to the dynamics of the production system. This requires that the reward module continues to determine a feedback that is observed by the RL-agent and the learning rate α is still positive and not set to zero. Another reason for a continued training after deployment is that in case the simulation does not entirely represent and match to the reality the offset is levelled over time.

If training is turned off, i.e. the learning rate α is set to zero, it is recommended to have a supervision of the system performance in place in order to capture system changes that require a re-training. Again, both alternatives are applicable in the presented control architecture. Note that re-training of the agent from scratch with a new system configuration is fast enough for most real-world applications, i.e. the frequency of significant changes happening in real-world is far below the time to train a new RL-agent. Moreover, the time-intense training can be outsourced and performed in parallel to real-world operations.

4.4 Reinforcement Learning modelling

This section introduces the RL-modelling that is the basis for finding an appropriate RL-agent that fulfils the requirements of an adaptive order dispatching for a complex job shop manufacturing system. Figure 4.9 summarizes the relevant features and outlines the structure of this section. It is based on the process flow displayed in Figure 4.7. A more detailed summary of the entire step-by-step approach is presented in Figure A3.1 in Appendix A3.


State module

Sec. 4.4.3 State representation

Agent module

Sec. 4.4.1 RL-algorithm, agent type

Sec. 4.4.5 Function approximation and hyper-parameters

Action module

Sec. 4.4.2 Action representation

Reward module

Sec. 4.4.4 Reward function

Figure 4.9: Methodological approach of modelling the RL-based adaptive order dispatching introduced in Figure 4.7.

First and foremost, the model-free RL-algorithm is determined in Section 4.4.1, what is hereinafter also called *agent type*. As already stated before, any agent type is based on the sequential decision-making and, therefore, the key modelling elements of an MDP and RL-agent are similar. The set of actions $\mathcal{A}$ is fixed and introduced in Section 4.4.2. All possible states $\mathcal{S}$ are described as state representation in Section 4.4.3. The current state S_t is calculated in the simulation environment as described in the previous Section 4.2. Whenever there is a state transition, the reward function r determines the reward feedback (see Section 4.4.4). The discount rate γ and learning rate α are two agent type-specific hyper-parameters that are discussed in Section 4.4.5.

4.4.1 Agent type – Optimization algorithm

The application determines to a large extent which RL-algorithm is suitable. The RL-algorithms described in Section 2.3.4, such as Q-learning or SARSA, use a tabular representation of the value function. Therefore, their area of application is limited to problems with comparatively small state and action spaces, which is not given for most production control applications in practice. Thus, this sub-section outlines, based on the theoretic concepts introduced in Section 2.3.3, advanced methods in the form of two algorithms, which both have been successfully applied in production control applications. As a summary, Table 4.1 gives an overview of their advantages and disadvantages. Afterwards, episode design alternatives are presented that are required due to the selected algorithm.

4.4.1.1 Advanced Reinforcement Learning methods

Deep-Q network (DQN), as introduced by Mnih & Kavukcuoglu et al. (2013), is an extension of the value-based Q-learning algorithm described in Section 2.3.4 that uses a neural network as function approximator. However, there are several drawbacks, e.g. unstable learning, when combining Q-learning and ANNs, which are addressed by the authors via the following additional enhancements of the basic Q-learning algorithm.

Experience replay is used to regularly update the network with a batch update method, integrating past transition experiences, i.e. state-reward-action tuples, that are stored in a replay memory. This increases the learning speed and the update robustness, as the temporal correlation between experiences is reduced. Moreover, once learned experience is never completely forgotten.

Next, in order to further improve and stabilize the training process, two ANNs are used. The first network represents the target Q-function and is only updated, for instance, every thousandth iteration to keep the target constant over that period. The second network includes all update iterations and updates the first network after a thousand iterations.

Finally, all positive reward values are normalized to the interval $[0, 1]$ and all negative values to the limits of $[-1, 0]$. That shows a more stable learning process as noise in the update data is further reduced.

However, the DQN algorithm requires additional parameters, e.g. for the experience replay, the two networks, as well as the network update frequency, which limit its universal applicability. Moreover, it shows a more evident sensitivity to hyper-parameter variations. For further computational results, related to the application of this algorithm, the reader may be referred to the work of (A_Schäfer 2018), (A_Kaiser 2019), and (A_May 2019).

Trust Region Policy Optimization (TRPO), developed by Schulman & Levine et al. (2015), is a policy-based algorithm and mainly used in the optimization of large non-linear policies. The specific goal is to find robust and sample-efficient gradients, as bad samples do affect not only the output performance but also the subsequent input values and can, thereby, lead to a vicious circle of deteriorating performance. Moreover, the learning rate α profoundly influences the step size of a gradient update, which can lead to a quickly declining performance. Finally, looking at the mathematical details, changes between the actual policy space and the parameter space of the parametrized policy could not be mapped efficiently before. The TRPO algorithm addresses these obstacles.

In general, gradient methods optimize a policy by following the gradient of an objective function in the optimization direction. Herein, TRPO makes use of a so-called *advantage function* $L_\pi(\pi') \approx J(\pi') - J(\pi)$ to evaluate the performance of the current policy π and the new policy π', and to direct the policy update (Sutton & Barto 2018). The update direction is computed based on the parametrized policy using the *conjugate gradient algorithm* (Schulman & Levine et al. 2015). However, this does not determine how far to go in the direction of the gradient.

Therefore, a trust region is defined around the current policy, i.e. the distance is constrained to keep the policy update close to the old policy. For TRPO, the trust region concept by Kakade & Langford (2002) is applied that implements the trust region restriction efficiently. Moreover, the *Fisher information matrix* enables an efficient computation independent of the parametrization and resolves the parameter mapping issue (Schulman & Levine et al. 2015).

All in all, the efficient computation without increasing the number of parameters are two theoretical advantages of TRPO compared to DQN. Moreover, the TRPO-algorithm shows a good computational performance in terms of universality, training data efficiency, i.e. the number of required training iterations, and robustness of the performance level also in stochastic environments (Schulman & Levine et al. 2015).

Table 4.1: Comparison of the presented advanced reinforcement learning algorithms DQN and TRPO.

	DQN	**TRPO**
Learning object	value function	policy
Approximation method	ANN	ANN
Continuous state	yes	yes
Large action space	yes	yes
Universality	low	high
Robustness	low	high

In conclusion, TRPO is selected based on the results of the theoretical and experimental comparison of both advanced RL-algorithms in Table 4.1. However, note that changing the

RL-agent type does not require an entirely new design and implementation as most model-free algorithms are, in principle, interchangeable because they are all based on the concept of an MDP. Mostly, hyper-parameters are different and some algorithms are, for instance, not able to handle continuous state or action values.

4.4.1.2 Episode definition

Having decided on the RL-algorithm, an episode definition is required as policy-based optimization algorithms are, by definition, multi-step methods and, therefore, need an episode definition to optimize the policy and maximize the accumulated reward (see Section 2.3.4).

Due to the dynamics and continuous operation in production applications, a natural episode definition is not self-evident. Hence, an assumption needs to be made. On the contrary, RL-applications in robotics or strategic games have an apparent episode definition due to the round-based setup, e.g. one robotic arm movement or one chess match. Moreover, Pardo & Tavakoli et al. (2017) stress in their work the relevance of episode time limits, which are caused by the episode definition. Four alternative episode definitions are considered in the present work to evaluate the most suitable modelling assumption. These are, on the one hand, time-based, meeting the timely nature of production operations, and, on the other hand, action-based referring to a fixed number of performed actions:

- *Time-based episode*: An intuitive way to define episodes is to subdivide it according to the time-based nature of production processes. An episode represents a period of fixed length, such as a shift or a working day.
- *Action-based episode* $Source \rightarrow Machine$: Fixed number of dispatching actions from the subset of actions that release an order, i.e. dispatch an order from a source to a machine.
- *Action-based episode* $Machine \rightarrow Sink$: Fixed number of actions, dispatching and transporting orders to a sink and thereby exiting the system.
- *Action-based episode* $A_{valid,t}$: Fixed number of any type of action that is valid in S_t.

A more detailed explanation, why these action subsets are selected, is given in the following sub-section together with the definition of the action representation.

4.4.2 Action representation

Finding an adequate action representation is rather straight forward because feasible decision alternatives are usually well-known and can be derived from the considered production control task. So, in the present work the actions are defined by the order dispatching decision-making problem.

However, having set the control task, any RL-application requires an evaluation of the level of detail for the action modelling. The two extremes are called *macro*- and *micro*-action modelling. A macro-action is a high-level action that is followed by several decisions that are taken by a deterministic procedure. For instance, a macro-action determines whether to transport an order or wait. Which order is transported, is determined subsequently. In contrast to that, micro-actions define every single step. Determining which order to transport on which route and how to move from one location to another would be a detailed micro-action modelling approach.

In the present work, an intermediate approach is developed. There are five subsets of dispatching decisions (following Section 4.2.2): $Source \rightarrow Machine$, $Machine \rightarrow Machine$, $Machine \rightarrow Sink$, $Empty$, and $Idle$. Besides that, any dispatching integrates two (sub-) decisions (following Section 4.3.2): which order to transport next and to which machine, within the feasible machine group, should the order be transported. So, in order to meet both requirements, the following dispatching **action definition** is used: an action $A_{O \rightarrow D}$ consists of a dispatching origin resource $O \in \mathcal{R} \cup \varnothing$ and dispatching destination resource $D \in \mathcal{R} \cup \varnothing$. Because this definition determines the origin and destination but not the distinctive production order, the order is selected subsequently by a deterministic procedure. This goes as follows: if there is more than one order available for an origin-destination-pair, the order with the longest waiting time is selected from the respective outbound buffer of the origin resource, i.e. the FIFO-principle is applied.

Two further modelling assumptions are made. First, if the origin O is not the current location of the dispatching resource, moving from the current location to the origin is included implicitly. Second, the empty set $\varnothing$ is allowed for O and D, which means that the dispatching origin or destination is not specified. This is required to represent the $Empty$ and $Idle$ decision subsets. In conclusion, this action representation is able to uniquely identify a dispatching decision that can be executed by the transport resource.

So, merging the five dispatching decision subsets with the origin and destination definition gives the summary depicted in Table 4.2. The shown **action subsets** map the relevant resource types, i.e. machines, sources, and sinks. Apparently, a source can only be an action's origin as well as a sink only an action's destination. Moreover, an action from a source to a sink is in no case appropriate. The idle A_{idle} and empty-handed actions A_{empty} are modelled either to have no specified origin and destination or just no origin. The remaining action subsets were already explained before. All action subsets mentioned in Table 4.2 pooled together define the set of executable actions:

$$A_{exec} := A_{idle} \cup A_{empty} \cup A_{S \to M} \cup A_{M \to M} \cup A_{M \to S} \quad 4.1$$

Table 4.2: Available action subsets for the reinforcement learning dispatching agent.

$A_{O,D}$	∅	Source	Machine	Sink
∅	A_{idle}	A_{empty}		-
Source	-	-	$A_{S \to M}$	-
Machine	-	-	$A_{M \to M}$	$A_{M \to S}$
Sink	-	-	-	-

It is important to point out that the action subsets have a substantially different influence on the performance measures. For instance, $A_{S \to M}$ releases an order into the system, which raises the inventory within the system and maybe also the machine utilization. Analogously, $A_{M \to S}$ has the opposite effect. Orders are taken out of the system, and the order cycle time is recorded. The RL-agent needs to learn to distinguish the different action subsets to choose its actions wisely. These effects were considered in the previously introduced episode design (see Section 4.4.1.2) and are also relevant for the reward function in Section 4.4.4.

The RL-agent is, in principle, allowed to choose from all actions in any state. However, depending on the current state S_t, not all actions are feasible. $A_{valid,t}$ defines the set of valid actions depending on S_t. Actions are infeasible, if the inbound buffer of the destination resource is full, the outbound buffer of a resource has no orders that can be transported, or the destination resource cannot perform the next process step, i.e. is not in the same machine group. As stated in Section 4.3.2, a recursion mechanism is implemented to handle invalid actions. Thereby, the agent is obliged to learn the action validity, too.

Action-mapping As the output of the RL-algorithm is always a discrete numerical value[1], an *action-mapping* is required to map the numerical value to the just defined dispatching actions that can be executed in the production environment. The above-mentioned action

[1] Recall that, looking one step closer into the implementation, the actions represent the output layer of the ANN and one output neuron is assigned to each action.

module implements the mapping. Two alternatives are considered in the remainder of this work to also analyse the effect of differently-sized action sets:

- **Direct mapping** enumerates every action of the subsets illustrated in Table 4.2 and, hence, *directly* maps any reasonable action. This gives an action set of the size $|A_{idle}| + |A_{empty}| + |A_{S \to M}| + |A_{M \to M}| + |A_{M \to S}|$. For instance, in the setup of Figure 4.5 with one source, two machines, and one sink, the idle action gets the value 1 and the first empty action to the source the value 2, to the first machine the value 3, and so on and so forth, which makes in total ten actions.

- **Resource mapping** is less restricted and enumerates any possible origin-destination-resource-pair, i.e. the entire two-dimensional matrix shown in Table 4.2 and does not consider just the reasonable action subsets. Hence, the action set has a much larger size and includes actions that are not valid in any case. The size equals to $|\mathcal{R}| \cdot (|\mathcal{R}| - 1) + |\mathcal{R}| + 1 = |\mathcal{R}|^2 + 1$, i.e. the number of edges in a complete bidirectional graph plus one for each empty action to a resource and adding one idle action. In the example of Figure 4.5 this makes in total 17 actions.

Both action-mappings are investigated and compared in Chapter 5.

4.4.3 State representation

The decision basis for determining the optimal RL-action is the state representation. It is also called *state vector*. Generally speaking, the state needs to be comprehensive enough but, then again, not too large to be intractable. The state vector elements are also called *features*. As for the action representation, the state information is transformed into numerical values that are interpretable by the RL-algorithm. The way of how to efficiently represent the state is elaborated in this section. Chapter 5 evaluates the herein presented modelling alternatives.

In the setup of the present work, all information is provided by the discrete-event simulation environment. The simulation is, in principle, able to provide any kind of information. Therefore, it presents an idealistic setup, which does not hold in any real-world application. So, one has to consider that the selected state features are also available for decision-making in reality to ensure transferability from the virtual to the real environment. Moreover, the information needs to be available in near real-time in order to serve the adaptivity requirement. As stated in Section 2.1.1 the semiconductor industry is a positive example in that sense as, for instance, location information is available via an indoor-location system and, hence, this information can be used in the training phase, too.

4.4.3.1 Approach towards the state representation design

The approach followed in the present work for determining the state representation is supported by the Cross-Industry Standard Process for Data Mining (CRISP-DM by Chapman & Clinton et al. (2000)). This is due to the similarity of the state vector as one major data input in reinforcement learning with supervised or unsupervised learning approaches, for which the CRISP-DM was initially developed. This sub-section focuses on two phases of the CRISP-DM: data selection and data pre-processing. The alternative state representations are presented afterwards in Section 4.4.3.2.

Data selection The first phase focuses on data selection. As stated before, the simulation provides nearly any kind of information. This allows the investigation of having additional data that might currently be not available in the real-world system. Eventually, when deploying the RL-system, the additional information first needs to be provided, e.g. by additional sensors. Moreover, it is recommended to consider just the relevant and necessary information in order to limit the amount of data, which might be another restriction in practice.

In the present work, process data that is related to the dispatching action, agent, resources, and orders are considered:

- *Action*-related data: feasibility of dispatching actions
- *Agent*-related data: current location
- *Resource*-related data: availability status, remaining processing time, inbound and outbound buffer availability, and total processing time (including waiting orders)
- *Order*-related data: waiting time and location information

These data categories, on the one hand, describe the current system state as-is, i.e. what one can see and observe on the shopfloor. On the other hand, they consider objective-related information, i.e. which data is required to reach the objectives. Furthermore, these categories are similar to the information that is used by established heuristic approaches outlined in Section 2.2.2.

Data pre-processing In most cases, the raw data from the shopfloor needs to be pre-processed, meaning that data values are transformed. However, it can also mean that new data values are generated based on the existing data. Furthermore, data pre-processing is applied to reduce the amount of training data and the number of decision-relevant features. In the following, simple mathematical pre-processing operations are presented together with an example that are related to production control problems:

- *Summing up* all order processing times in a buffer in front of a machine gives the total processing time or workload.
- *Subtract* the processing time from the total cycle time to get the waiting time, which is, in turn, the relevant part of the cycle time that can be influenced by the dispatching agent.
- *Dividing* the buffer fill level by its capacity gives the relative fill rate of the buffer or, vice versa, determines the availability of a buffer.
- Calculate the distance of an order *relative* to the current location of the RL-agent, instead of providing absolute location coordinates.

Further normalization operations are recommended, in particular when working with ANNs, to achieve a fast and robust convergence and reduce numerical instabilities (García & Luengo et al. 2015). The following operations are assessed to be suitable for the order dispatching application:

- *Min-Max-Normalization*: x is an element of the original value range $[x_{min}, x_{max}]$ and transformed to a new value range $[a, b]$ via:

  $$x' = \frac{(x - x_{min})(b - a)}{x_{max} - x_{min}} + a \tag{4.2}$$

- *Z-Score-Normalization*: If x_{min} or x_{max} are not known, the dataset contains various outliers, or the min-max-normalization is not reasonable for the desired application, the z-score-normalization can be applied to a dataset DS with given mean μ_{DS} and standard deviation σ_{DS} via:

  $$x' = \frac{x - \mu_{DS}}{\sigma_{DS}} \tag{4.3}$$

  The transformed dataset DS' has the mean $\mu_{DS'} = 0$ and standard deviation $\sigma_{DS'} = 1$. In the present work, the entire dataset and value range are not known upfront. Therefore, an incremental calculation of the mean and standard deviation is suggested. Since the RL-agent influences the course of the data through its behaviour and makes its

Machine states	Encoding		
operating	1	0	0
idle	0	1	0
broken	0	0	1

State element	State entry nr. and descr.		Value
Available orders for dispatching	1.	Machine 1 to machine 2	1
	2.	Machine 1 to machine 3	0
	3.	Machine 2 to machine 1	1
	4.	Machine 2 to machine 3	1
	5.	Machine 3 to machine 1	0
	6.	Machine 3 to machine 2	1
Remaining processing time	7.	Machine 1	43
	8.	Machine 2	67
	9.	Machine 3	12

Figure 4.10: One-hot coding example for three machine states of one machine (left), and one-dimensional state vector for an exemplary three machine setup (right).

decisions based on the normalized data, it also must be ensured that past data has a limited influence on the most recent calculation. The exponentially weighted mean and variance are suitable in that case (the parameter is set to $\lambda = 0.01$ if not stated differently):

$$\mu_t = (1 - \lambda) \cdot \mu_{t-1} + \lambda \cdot x_t \qquad 4.4$$

$$\sigma_t^2 = (1 - \lambda) \cdot (\sigma_{t-1}^2 + \lambda \cdot (x_t - \mu_{t-1})^2) \qquad 4.5$$

Finally, non-numerical, categorical data needs to be transformed into a numerical representation without losing any relevant information. For instance, order-machine-allocations or machine states are categorical information[1]:

- *One-hot coding*: If a categorical information has n different categories, it can be represented by n binary values. Figure 4.10 on the left gives an example based on the machine state information, for which three states are considered. These are represented by a one-hot coding of three binary values.

The pre-processed data features are integrated into one state vector. Figure 4.10 on the right illustrates a simplified example of three machines. The information about the available orders from any origin to any destination machine (six state entries) and the remaining processing time of all machines (three state entries) are integrated into the state vector.

[1] An overview of approaches for "coding" categorical information is given by Jeff Hale (2018), *Smarter Ways to Encode Categorical Data for Machine Learning*. `https://towardsdatascience.com/smarter-ways-to-encode-categorical-data-for-machine-learning-part-1-of-3-6dca2f71b159` (accessed on 17.06.2020), and Max Kuhn & Kjell Johnson (2019), *Feature Engineering and Selection: A Practical Approach for Predictive Models*. `http://www.feat.engineering/` (accessed on 17.06.2020).

4.4.3.2 State representation for an adaptive order dispatching in a job shop

The reasonable state representation for the dispatching decision is based on the approach above and, on the one hand, considers information that is currently used by established dispatching heuristics and, on the other hand, additional information that is based on findings throughout this research. In this section, every investigated state element is listed and explained. A **state element** is denoted as capital letter S followed by a subscription referring to the information feature. All state elements are calculated based on the current time step t. However, for reasons of improved readability, the following formulas omit the subscription index t if not explicitly needed. Note that all state elements are compatible with each other, and so the state vector S_t can be just a single state element or a combined set.

Action-related state information First, it is reasonable to indicate for every available action whether it is **valid** or not in a certain state, i.e. whether it is an element of $A_{valid,t}$ or not. As invalid actions can be also part of the set of actions $\mathcal{A}$, this indication is highly relevant for learning action validity. The information is encoded in a binary variable va_i and the state element is named S_{VA}:

$$va_i := \begin{cases} 1 & a_i \in A_{valid,t} \\ 0 & \text{else} \end{cases} \quad \forall a_i \in \mathcal{A} \tag{4.6}$$

Agent-related state information Additionally, the information on the current **location** of the dispatching agent is of interest as machines are spread due to the job shop layout. This is, in particular, the case if the dispatching agent is the system bottleneck and, hence, the transport route optimization is important. The NJF heuristic is grounded on this information and reasoning, too. The location is encoded using the one-hot coding for categorical information, i.e. every possible resource location $\mathcal{R}$. The state element is named S_L and the binary variable for every resource l_i:

$$l_i := \begin{cases} 1 & \text{if the dispatching agent is at resource } i \\ 0 & \text{else} \end{cases} \quad \forall i \in \mathcal{R} \tag{4.7}$$

Machine-related state information As machines break down from time to time, the machine **availability** is another important information. For instance, a machine that is not available is less likely a "good" destination D of a dispatching action. The machine state S_{MF} for every machine is defined as follows:

$$mf_i := \begin{cases} 1 & \text{if machine } i \text{ is not available} \\ 0 & \text{else} \end{cases} \quad \forall i \in M \tag{4.8}$$

Next, it is relevant whether the machine is currently processing an order or not. More precisely, every machine has a variable indicating the **remaining processing time** rpt_i, which gives the state element S_{RPT}. This variable indicates whether the machine is soon finishing an operation and in consequence an order is available in the outbound buffer and another order is instantly taken from the inbound buffer. rpt_i is calculated based on the remaining processing time RPT_i divided by the average processing time APT_i at this machine. The fraction is applied to scale the variable value and allow the comparison of different machines:

$$rpt_i := \frac{RPT_i}{APT_i} \quad \forall i \in M \tag{4.9}$$

Apart from the remaining processing time and machine availability, the inbound and outbound **buffer availabilities** are important to capture the workload in front of a machine as well as the number of waiting orders that are finished and can be dispatched. Both information allow the conclusion on the criticality of a machine, i.e. the risk of starvation or blocking. *Starvation* means that the inbound buffer is empty and *blocking* happens when the outbound buffer is fully occupied. For instance, the FLNQ heuristic captures the buffer availability, too. S_{BEN} represents the inbound buffer and S_{BEX} the outbound buffer remaining free slots. As for the remaining processing time, the variable is scaled based on the capacity CAP_i^{EN} and CAP_i^{EX} and the number of occupied buffer slots OCC_i^{EN} and OCC_i^{EX} at machine i. So, the variables ben_i and bex_i have the value range $[0, 1]$:

$$ben_i := 1 - \frac{OCC_i^{EN}}{CAP_i^{EN}} \quad \forall i \in M \tag{4.10}$$

$$bex_i := 1 - \frac{OCC_i^{EX}}{CAP_i^{EX}} \quad \forall i \in M \tag{4.11}$$

Finally, the workload of orders waiting in front of a machine, i.e. the **total processing time**, is provided as state element S_{BPT}, containing the state variable bpt_i for every machine i. This information is redundant to the buffer fill level, though still relevant as processing times vary on a large range. Hence, the information on just the buffer fill level might not be accurate enough. PT_i gives the sum of the waiting orders' processing time. Again, the variable is scaled according to APT_i multiplied by the capacity of the inbound buffer CAP_i^{EN}, and then shifted by 1 to take also negative values, e.g. if the total processing time is below the average:

$$bpt_i := \frac{PT_i}{CAP_i^{EN} \cdot APT_i} - 1 \quad \forall i \in M \tag{4.12}$$

Order-related state information Next, the state vector element S_{WT} deals with the **waiting time** of orders that are available for dispatching, i.e. orders that are placed in an outbound buffer. The variable wt_i covers all source and machine outbound buffers and indicates the longest waiting time WT_i^{max}, similar to the heuristics FSFO and FIFO. Z-score-normalization is applied to normalize the waiting time values according to the mean waiting time WT_i^{mean} and standard deviation WT_i^{std} with respect to the resource type at which the order is currently located, i.e. either source or machine. The reasoning behind the latter is the separation of orders that are waiting at a source at the entrance of the job shop system and orders that are inside the system, which do have different waiting time values as the total waiting time is accumulative:

$$wt_i := \frac{WT_i^{max} - WT_i^{mean}}{WT_i^{std}} \quad \forall i \in M \cup SO \tag{4.13}$$

This formula is clarified by giving an example: assuming there are multiple machines with an average waiting time $WT_i^{mean} = 100$ and standard deviation $WT_i^{std} = 10$. There are two orders in the outbound buffer of the first machine with waiting times 120 and 80. So, the longest waiting time for the first machine is $WT_1^{max} = 120$ and the waiting time state value is calculated as follows $wt_1 = \frac{120-100}{10} = 2$.

Next, state S_{AT} represents the **time** it takes to **perform an action** $a_i \in \mathcal{A}$. This allows the RL-agent to get a grasp of temporal effects of specific actions. Any invalid action is set to zero. For all valid actions, the time to move from the current location to the origin O of the order $t_{\rightarrow O}$, the time to perform the transport $t_{O\rightarrow D}$, as well as handling times t_{load} and t_{unload} are considered. Min-max-normalization is applied, and the variable is scaled by at_{max}, which represents the maximum possible value that can be determined for any system based on the layout and movement speed, i.e. taking the maximum distance between any two resources in the layout, dividing it by the movement speed, and adding the handling times. Based on this information, the dispatching agent can consider the time that passes by when selecting an action, which helps to prioritize actions if the agent is the system bottleneck. Moreover, the composite dispatching rule of Mönch & Fowler et al. (2013) captures this information:

$$at_i := \begin{cases} \frac{t_{\rightarrow O} + t_{O\rightarrow D} + t_{load} + t_{unload}}{at_{max}} & a_i \in A_{valid,t} \\ 0 & \text{else} \end{cases} \quad \forall a_i \in \mathcal{A} \tag{4.14}$$

4.4.4 Reward function

The objective of the RL-agent is to maximize the expected cumulative reward, resulting from a continuous evaluation of the actions taken. Therefore, the reward must be defined so that its maximization leads to the desired behaviour. In the present case, the desired behaviour is described by the overall performance objectives. However, as there are various ways of modelling the reward function that poses a challenge, particularly when looking at new RL-applications, such as production control, which are not yet well-studied (Ng & Harada et al. 1999; Dewey 2014; Hadfield-Menell & Milli et al. 2017; Ou & Chang et al. 2019). Most authors agree that the **reward design** is one of the most critical steps in the implementation of reinforcement learning and requires expertise and knowledge about the application domain. Recalling the reward function definition $r(s, a, s')$, the reward design is not limited to the purposeful definition of the reward in accordance with the objectives, but it must also reconcile and correlate to the state and action representation.

A poor reward design bears, for example, the risk of *reward hacking* (Russell & Norvig 2016). This is the case when "[...] a vacuum cleaner ejects collected dust so that it can collect even more dust." (Hadfield-Menell & Milli et al. 2017). Thereby, the agent maximizes its reward but does not implement the desired behaviour.

The remainder of this sub-section explains, first, two reward modelling concepts with its advantages and disadvantages (see Section 4.4.4.1) and, based on this, Section 4.4.4.2 outlines the developed approach towards the reward design. Lastly, alternative reward representations are introduced for the adaptive order dispatching (see Section 4.4.4.3).

4.4.4.1 Sparse and modelled reward

In episodic environments, the RL-agent receives a reward based on the performance in each episode (see Section 2.3.2). This is especially straightforward in board games such as chess. In that case, winning a game gives a positive reward and losing a negative reward. The possibility of a draw might occur, whereby the RL-agent is neither rewarded nor punished. Modelling a similar reward function for a generic production control agent could go as follows:

$$r_{sparse}(\bullet)^{1} = \begin{cases} 1 & \text{production plan fulfilled} \\ -1 & \text{production plan not fulfilled} \\ 0 & \text{else} \end{cases} \qquad 4.15$$

[1] For reasons of improved readability, the reward function $r(s, a, s')$ is shortened to $r(\bullet)$.

Due to the possibly large number of state transitions within an episode, the RL-agent is rarely updated according to positive or negative feedback. This setting is called **sparse reward** and hereinafter defined as follows (Sutton & Barto 2018): during the episode, except in terminating states, the RL-agent receives the neutral reward value zero. If the reward is triggered only after a series of actions, it is for the agent hard to identify the relationship between its actions taken and the rewards received for the chain of actions. This is also called *credit assignment problem* (Sutton & Barto 2018). The key question is, what are the most relevant actions, which are decisively involved in achieving the reward maximization and consequently need to receive higher credits than those with a negative effect. This drawback of sparse reward is opposed by the advantage that it rewards only the achievement of the objective. How the objective is achieved, is not considered, and the agent is free to make its decisions towards it. This, in turn, enables the agent to learn policies that outperform, for instance, human experts such as shown by Silver & Schrittwieser et al. (2017).

On the contrary, **modelled reward** or **reward shaping** extends the reward function by incorporating additional rewards within an episode and on the way towards the objective (Kaelbling & Littman et al. 1996; Randløv & Alstrøm 1998; Ng & Harada et al. 1999). This type of reward modelling is preferred if the problem complexity does not, or only with high computational effort, permit the achievement of the objective via sparse reward. So, the resulting reward function is composed of two parts:

$$r(\bullet) = r_{sparse}(\bullet) + r_{modelled}(\bullet) \tag{4.16}$$

The modelled reward is designed in accordance with the application and, usually, requires an extensive understanding of the problem domain to avert contradictions with the overall objective represented by the sparse reward. Continuing with the simplified production control example from Equation 4.15, a modelled reward could consider the in-time completion of an order:

$$r_{modelled}(\bullet) = \begin{cases} 0.01 & \text{order finished in-time} \\ -0.01 & \text{order not finished in-time} \\ 0 & \text{else} \end{cases} \tag{4.17}$$

The modelled reward needs to be scaled with respect to the value range of the sparse reward. Therefore, the modelled reward value is much smaller in this example, but still the numbers are arbitrary values. Moreover, the sum of both rewards should be in a constant range.

In conclusion, the following needs to be considered when sparse and modelled rewards are both considered. Any adjustment of the reward function changes the solution space of the original MDP and, hence, a new MDP is given. This can cause serious issues, because the optimal policy for the original MDP can no longer be found as the modified MDP may have a different optimal policy. Continuing with the vacuum cleaner example: The initial sparse reward is $+1$ if the entire room is clean. Additional reward $+1$ is modelled for the collection of dust to guide the cleaner towards an efficient cleaning route. Since the RL-agent does not know the real problem to solve and only aims at maximizing the return, the optimal policy with the modelled reward strives for a permanent collection of dust and, hence, deviates from the desired policy.

To ensure that the optimal policy of the modified MDP corresponds to the optimal policy of the original MDP, the modelled reward function needs to be representable through a real-value function $\Phi(\bullet)$ and discount rate γ such that for all states $s, s' \in \mathcal{S}$ and actions $a \in \mathcal{A}$ the following holds (Ng & Harada et al. 1999; Wiewiora 2011):

$$r_{modelled}(s, a, s') \overset{!}{=} \gamma \cdot \Phi(s') - \Phi(s) \qquad 4.18$$

If this holds, the modelled reward function is called a potential-based shaping function, and a necessary and sufficient condition is given to guarantee consistency with the original MDP.

Summarizing both reward concepts reveals that the definition of an appropriate reward function should be based on *what* the objective of the RL-agent is and not *how* it should solve the respective task. Only if the task is too complex, the agent should be guided towards its desired behaviour by a more detailedly modelled reward, however, keeping the risks of modifying the MDP and ignoring better options by limiting the agent's degree of freedom in mind. Furthermore, the efficiency of a reward signal can be increased by adjusting the state and action representation accordingly.

Chapter 5 investigates sparse and modelled reward functions separately in order to examine their applicability for an adaptive order dispatching in a complex job shop. Therefore, the following sub-section describes the approach that is developed and applied in the present work to define sparse and modelled reward functions that are appropriate in principle, and the computational results will demonstrate their applicability.

4.4.4.2 Approach towards the reward function design

First, it is necessary to define the objectives for which the performance is evaluated and optimized. In the domain of PPC it is, in particular, crucial to investigate the performance for the dilemma of contradicting objectives (see Section 2.1.2). Having selected the objectives, it has to be considered how the KPIs are correlated to the agent's state and action representation.

Optimization objectives The performance evaluation is based on the logistical performance indicators defined in Section 2.1.2. As stated in Section 3.1 and Section 3.2, the research focus is on multiple objectives, i.e. resource- and order-related indicators. The following KPIs are considered in the present research:

- Minimize the average order **cycle time** CT
- Maximize the average machine **utilization** U

The utilization U is defined in Section 2.1.2 as manufacturing uptime utilization and serves as a resource-related indicator, which is of particular interest in semiconductor operations. Moreover, in accordance to the lean philosophy, a short order cycle time CT is preferred to ensure a time-efficient order processing in job shop work centres.

Afterwards, one has to check for both KPIs if these are too aggregated to be used directly as reward for the agent's performance. For instance, the overall equipment effectiveness (OEE) is an aggregated performance measure that is not necessarily influenced by a single dispatching action. Thus, the herein presented approach considers the previous work of Stricker (2016), which deals with production-related KPIs, their relationship to each other, and the right aggregation level.

In fact, the cycle time is an aggregated performance indicator that is defined as the sum of processing, waiting, setup, and transport time. As the processing time and the transport time are fixed and pre-determined – the transport capacity is just one, so there is no waiting time due to consolidated transport of multiple orders – the waiting time is the only part of the cycle time that can be influenced by the dispatching agent. This will be considered in Section 4.4.4.3. In contrast to that, the utilization is an appropriate performance measure, as it is defined as uptime utilization and hence downtimes are already excluded.

Table 4.3: Influence of action subsets on optimization objectives (positive effect $+$, negative effect $-$, or positive as well as negative effects $\sim$).

Action subset	Utilization	Cycle time
A_{idle}	$\sim$	
A_{empty}	$\sim$	
$A_{S \rightarrow M}$	$+$	$-$
$A_{M \rightarrow M}$	$+$	$+$
$A_{M \rightarrow S}$	$\sim$	$+$

Relate reward objectives with action and state space Next, it is required to understand the environment dynamics and especially the state-action-reward relations better in order to define a suitable reward function $r(s, a, s')$ that is based on these. Table 4.3 illustrates these relationships on a qualitative level with respect to the objectives. The table does not show the state separately, due to reasons of comprehensibility and readability, but the following explanation of the effects considers both aspects and distinguishes different states that might occur:

- A_{idle} and A_{empty}: If the transport resource is highly utilized, both action subsets are most likely not beneficial for the objectives. However, idling to wait at a location for an order that will arrive soon or moving empty-handed to a location where an order is expected can, in principle, be an advantageous decision of the RL-agent. Hence, there is no clear positive or negative effect.

- $A_{S \to M}$: Supplying a starving machine increases directly the overall machine utilization. If the machine inbound buffer is already filled with an order or the machine is currently broken, the effect is neutral. However, it also increases the inventory within the system and, hence, causes rising average cycle times (see Little's Law in Section 2.1.2).

- $A_{M \to M}$: Moving orders from machines to machines increases the destination machine's utilization, too (see explanation before). Moreover, it directly affects the cycle time positively, as waiting in an outbound buffer of a machine is aborted. Generally speaking, $A_{M \to M}$ actions contribute to the levelling and distribution of inventory within the system and between the machines.

- $A_{M \to S}$: Transporting an order from a machine to a sink increases the machine's utilization only if the machine is blocked due to a full outbound buffer. Except for that rather rare case, this action does not influence the utilization. Other than that, it finishes the order processing and, hence, has a positive effect on the cycle time, too (see explanation before).

Intuitively, a positive effect in Table 4.3 should be rewarded by a positive value and, vice versa, by a negative value. According to Sutton & Barto (2018), the value range of the reward should be fixed and the interval $[-1, 1]$ is recommended. The best state-action-pair should receive the highest value $+1$ and -1 for the worst. In between, an exponential curved reward function is recommended to achieve a continuous function with monotonously increasing gradients towards the optimal state (Sutton & Barto 2018).

4.4.4.3 Reward function for an adaptive order dispatching in a job shop

After the optimization objectives are determined and understood, the reward function can be designed accordingly. First, modelled and sparse rewards are explained. After that, a *tree-based* reward is introduced as an enhanced modelled reward function. All reward functions are evaluated in Chapter 5.

Modelled reward Because of the different influences derived in Table 4.3, the first modelled reward function is a **constant** reward function r_{const}, which rewards the action subsets $A_{S \to M}$ and $A_{M \to S}$ separately, as those subsets show the most ambiguous effects on the objectives. The reasoning is to analyse whether a reward for those two action subsets is enough to optimize either objective according to Table 4.3. Two constant reward values ω_1 and ω_2 are assumed for the action subset the selected action belongs to:

$$r_{const}(\bullet) := \begin{cases} \omega_1 & a \in A_{S \to M} \cap A_{valid,t} \\ \omega_2 & a \in A_{M \to S} \cap A_{valid,t} \\ 0 & \text{else} \end{cases} \quad \forall a \in \mathcal{A}, s \in \mathcal{S} \tag{4.19}$$

While the constant reward does only indirectly consider the objectives, two further modelled reward functions r_{util} and r_{wt} are introduced. Both take the actual objective function value as is for the reward calculation. r_{util} rewards every valid action by the current average machine **utilization** U. The average is computed based on the period since the last action. The exponential function e is applied to achieve increasing gradients towards the optimum, i.e. the maximum utilization. Additionally, two constant parameters help to align the reward according to the normalized reward value range of $[0, 1]$. The value range of the utilization has the same interval. Invalid actions are rewarded with zero. Figure 4.11 displays the chart of the reward function:

$$r_{uti}(\bullet) := \begin{cases} e^{\frac{U}{1.5}} - 1 & a \in A_{valid,t} \\ 0 & \text{else} \end{cases} \quad \forall a \in \mathcal{A}, s \in \mathcal{S} \tag{4.20}$$

The **waiting time** reward directs towards the minimization of the order cycle time. As stated before, the cycle time is too aggregated and, therefore, just the relevant element, i.e. the waiting time, is considered. Again, invalid actions are rewarded with zero. To take into account that orders that just entered the system have a lower waiting time than orders leaving the system, the waiting time is normalized for the resource type, as already explained in Equation 4.13 in Section 4.4.3.2. Again, two constant parameters take care of the normalization

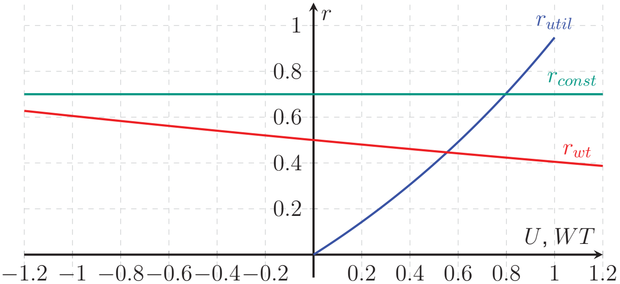


Figure 4.11: Modelled reward functions r_{const} (with exemplary $\omega_1 = \omega_2 = 0.7$), r_{util}, and r_{wt}.

for the target reward value range of $[0, 1]$. The parameters are based on several simulation experiments, from which the waiting time value range could be calculated. The normalized waiting time can also be negative and the range is not limited since the normalized waiting time value is theoretically unbound (see Figure 4.11):

$$r_{wt}(\bullet) := \begin{cases} e^{-0.1 \cdot WT_i} - 0.5 & a \in A_{valid,t} \\ 0 & \text{else} \end{cases} \quad \forall a \in \mathcal{A}, s \in \mathcal{S} \tag{4.21}$$

Moreover, two **weighted** reward functions $r_{w,util}$ and $r_{w,wt}$ are defined that integrated the two ideas behind the functions r_{const} and r_{util} or r_{wt}, which were just explained:

$$r_{w,util}(\bullet) := \begin{cases} \omega_1 \cdot r_{uti}(\bullet) & a \in A_{S \to M} \cap A_{valid,t} \\ \omega_2 \cdot r_{uti}(\bullet) & a \in A_{M \to S} \cap A_{valid,t} \\ 0 & \text{else} \end{cases} \quad \forall a \in \mathcal{A}, s \in \mathcal{S} \tag{4.22}$$

$$r_{w,wt}(\bullet) := \begin{cases} \omega_1 \cdot r_{wt}(\bullet) & a \in A_{S \to M} \cap A_{valid,t} \\ \omega_2 \cdot r_{wt}(\bullet) & a \in A_{M \to S} \cap A_{valid,t} \\ 0 & \text{else} \end{cases} \quad \forall a \in \mathcal{A}, s \in \mathcal{S} \tag{4.23}$$

Sparse reward Sparse reward functions are just limited to a reward at the end of each episode. The sparse reward functions are defined in a similar way to the modelled reward. Analogously to r_{const}, the reward function $r_{const,ep}$ gives a constant reward at the end of every episode:

$$r_{const,ep}(\bullet) := \begin{cases} 1 & \text{if } a \text{ ends episode} \\ 0 & \text{else} \end{cases} \quad \forall a \in \mathcal{A}, s \in \mathcal{S} \tag{4.24}$$

Next, the sparse complements of r_{uti} and r_{wt} are $r_{uti,ep}$ and $r_{wt,ep}$ and calculated the same way, except the fact that the utilization is averaged over the entire episode and the waiting time is averaged for all orders that have been dispatched to a sink within the episode:

$$r_{uti,ep}(\bullet) := \begin{cases} e^{\frac{U}{1.5}} - 1 & \text{if } a \text{ ends episode} \\ 0 & \text{else} \end{cases} \quad \forall a \in \mathcal{A}, s \in \mathcal{S} \tag{4.25}$$

$$r_{wt,ep}(\bullet) := \begin{cases} e^{-0.1 \cdot WT} - 0.5 & \text{if } a \text{ ends episode} \\ 0 & \text{else} \end{cases} \quad \forall a \in \mathcal{A}, s \in \mathcal{S} \tag{4.26}$$

Multi-objective reward functions In addition to just a single rewarded objective, as previously defined, a **weighted sum** of multiple reward functions r_i via weight factors ω_i is investigated. This follows the idea of multi-objective optimization[1]:

$$r_{res}(\bullet) := \sum_{i=0}^{N} \omega_i r_i(\bullet) \quad \forall a \in \mathcal{A}, s \in \mathcal{S} \tag{4.27}$$

Tree-based reward Finally, a calculation routine for a *tree-based* reward is developed, which aims at the optimization of multiple objectives and is based on the idea of a look-ahead approach (see the work by Chen & Xia et al. (2015) and Silver & Hubert et al. (2018) presented in Chapter 3). In order to address the dilemma of contradicting objectives, it incorporates additional domain knowledge about the job shop system and order dispatching task under consideration. Ou & Chang et al. (2018) show that a performance increase can be achieved if additional system understanding is incorporated in the reward design.

Take the following as an illustrating example for the reasoning behind the tree-based approach: three out of five machines in a job shop are blocked since their inbound and outbound buffers are all filled. As a consequence, the machine utilization drops and order waiting times

[1] Note that by the weighted sum of multiple reward functions also modelled and sparse reward functions can be combined. However, this is not explicitly analysed in the present work as no significant results could be obtained in preliminary experiments.

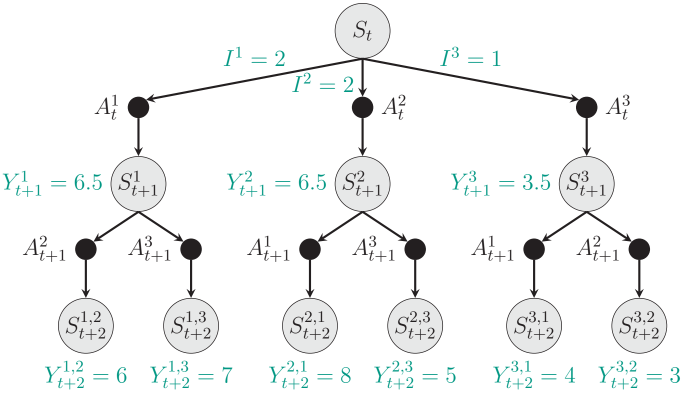


Figure 4.12: Example for the tree-based reward definition with three possible actions in state S_t. The evaluation Y of each state and the rank indices I are highlighted. The evaluations of the nodes on the deepest level are arbitrary distance values Y_{dist}. The other values are computed based on the avg aggregation rule.

accumulate. Herein, the optimal dispatching behaviour would be to dispatch the finished orders in the outbound buffers first, so that the machines can continue processing as quickly as possible. Therefore, the minimization of the total transport distance to these three machines is the best way to minimize the machines' and orders' idle time. This far-sighted decision-making should be the goal of the RL-agent. The tree-based reward function is designed to achieve precisely this goal and reward the selected action A_t according to a tree of future state evaluations. It has to be mentioned that a reward function that considers several consecutive states and actions is somehow contrary to the reward function definition $r(s, a, s')$, but it is not contradicting, because the future states and actions in the herein presented approach are just estimates.

Figure 4.12 shows an example for the tree-based reward calculation. The reward calculation compares the evaluation of the selected action with all other possible action evaluations and defines a rank-based reward. The following formal explanation is based on Figure 4.12.

Given a state S_t, there are $A(S_t) = \left\{A_t^1, \dots, A_t^{|A(S_t)|}\right\}$ possible actions. S_t is drawn on the top layer of the tree and called *root*. The root has $|A(S_t)|$ child nodes S_{t+1}^i for each action $A_t^i \in A(S_t)$. A child represents an estimated following state S_{t+1}. Analogous to the root, every child node again has $|A(S_t)| - 1$ further child nodes for any left action $A_{t+1}^j \in A(S_{t+1}) = A(S_t) \setminus A_t^i$. The node subscript in Figure 4.12 defines the level within the tree or, in other words, the number of actions that need to be taken to reach that node, and the superscription indicates the action sequence. The number of nodes in a tree is given by $\frac{|A(S_t)|!}{(|A(S_t)|-m)!}$, where m represents the number of look-ahead layers, i.e. by default the number of actions in state S_t. m is a parameter that can be set by the user to control the tree size and, hence, the computational effort.

Next, an *evaluation function* and an *aggregation function* are required to compute the reward for the action selection on the root-level. The evaluation starts on the deepest look-ahead layer of the tree, i.e. the leaves of the tree, in order to incorporate the most far-sighted behaviour. The nodes on the deepest level are evaluated for the summed distance Y_{dist}, i.e. the total distance of all actions that are performed to reach that node. For comparison, the machine utilization Y_{util} is considered as alternative evaluation criteria.

After that, the aggregation function $f_{aggr}(\bullet)$ determines the evaluation of the nodes in the layers above. For instance, the first node S^1_{t+1} on layer $t+1$ is based on the values of all its children, i.e. $Y^1_{t+1} = f_{aggr}\left(\left\{Y^{1,2}_{t+2}, Y^{1,3}_{t+2}\right\}\right)$. The following alternative aggregation functions are investigated: $\min(\bullet)$, $\max(\bullet)$, or $\operatorname{avg}(\bullet)$. $\min$ represents an optimistic estimate since the reward is calculated assuming an optimal behaviour in all subsequent steps. On the contrary, $\max$ corresponds to the worst-case-perspective. The reward calculation assumes that the agent always chooses the worst action in the subsequent steps. Finally, avg corresponds to the non-weighted expected value of all subsequent steps. The descriptions of $\min$ and $\max$ are reversed in case of a maximization evaluation, such as the machine utilization Y_{util}.

Finally, from the aggregated values Y_{t+1} of the nodes in the first look-ahead layer, the reward of the agent is determined by comparing the evaluations of all possible actions via an index function $I(\bullet)$. The function indicates the rank according to a sorted list of all aggregated values Y_{t+1}. Whether sorting in ascending or descending order depends on the optimization direction. The highest reward is given for the highest rank.

So, the **tree-based** reward for action A_t can be determined using the normalized index value according to:

$$r_{tree}(\bullet) := \begin{cases} \frac{|A(s)|-I(a)}{|A(s)|-1} & a = A_t, s = S_t, a \in A_{valid,t} \\ 0 & \text{else} \end{cases} \qquad \forall a \in \mathcal{A}, s \in \mathcal{S} \qquad 4.28$$

One final remark on the computation time of the tree-based reward. The computation is strongly influenced by the number of tree nodes, which grows more than exponentially as the number of look-ahead layers increases. For this reason, the necessary time for a reward calculation beyond a certain amount of look-ahead layers is too high. Therefore, a workaround is integrated that allows the limitation of the tree to a pre-defined maximum number of nodes.

Another advantage of limiting the tree based on the number of nodes compared to the number of layers is that, especially when there are just a few possible actions, all actions are assessed, i.e. very far-sighted. The tree is just limited when the number of actions is vast, in which case the trade-off between the available actions should not be as critical.

4.4.5 Function approximation and hyper-parameters

After an appropriate RL-model is found, the algorithm's fine-tuning needs to be performed. Therefore, RL-algorithms come with hyper-parameters that require specification. The parameters that are relevant to the TRPO-algorithm are already introduced in Section 2.3 and explained hereinafter. The parametrization is based on the analysis of several computational experiments performed by (A_Schäfer 2018), (A_Jakubik 2018), (A_Kaiser 2019), (A_Theiß 2019), and (A_May 2019) as well as the author's publications in Stricker & Kuhnle et al. (2018), Kuhnle & Röhrig et al. (2019), and Kuhnle & Schäfer et al. (2019). Table 4.4 summarizes the default parameter setting findings and computational results are captured in Figure A4.1 of Appendix A4. In general, the investigations show that hyper-parameters influence primarily the learning speed and the robustness of the agent's performance.

Table 4.4: Default hyper-parameter definition of the TRPO-algorithm developed in the present work.

Parameter	Default value
Learning rate α	0.001
Discount rate γ	0.9
ϵ-greedy strategy	- (turned off)
ANN configuration	input layer (states) $\times 128 \times 128 \times$ output layer (actions)
ANN activation function	$\tanh$

The **learning rate** α is a parameter well-known from gradient descent-based methods. It determines how much the error that is determined by the difference of the learned estimation and the actual value goes into the policy update (see for comparison Equation 2.30). If the learning rate is too high, according to Goodfellow & Bengio et al. (2016), the learning curve oscillates strongly. If, on the other hand, it is too small, the learning process is slowed down. However, significant effects on the achieved performance after the learning process cannot be found for varying learning rates within reasonable boundaries in the present work. Within the scope of this work, a learning rate of $\alpha = 0.001$ has shown to be suitable.

The parameter γ corresponds to the **discount rate**, which is included in the calculation of the discounted reward (see Equation 2.12). To account for uncertainties about the future and for numerical reasons, it is set to a value of less than one. Comparing it to the learning rate, the discount rate has an application-specific intuition. The time lag in terms of time steps t between an action and its effect on the performance indicators, i.e. the reward, is reflected by the discount rate. Hence, applications in which the reward is directly related to an action without any effect on future states and rewards do not require a high discount rate value. Looking at the computational results of the present work, a higher learning speed is observed for smaller γ values. This can be explained as follows: if a larger discount rate is used, the agent must estimate further future rewards, thus, showing more far-sighted behaviour compared to an agent with a small discount rate. However, agents with a high discount rate achieved a significantly higher performance in terms of average utilization and order cycle time. Hence, the trade-off between learning time and achieved performance prevails when fine-tuning this parameter. If not stated differently, the value of $\gamma = 0.9$ is used as default, which has shown good performance in most experiments.

The **ϵ-greedy strategy** is turned off in this work, i.e. the value is set to zero. Experiments with the policy-based TRPO-algorithm showed good results with no additional exploration induced by ϵ. The policy of the TRPO-algorithm is by itself stochastic and is not exploiting too fast with the risk of getting stuck in a locally optimal solution. Moreover, setting ϵ to a positive rate entails the risk of a negative influence on the agent's performance as random actions are performed that are most likely sub-optimal and do not result from the agent's policy.

In terms of the **ANN configuration**, which represents the parametrized policy π_θ, an MLP network with two hidden layers and 128 nodes per layer was found to be reasonable. $\tanh$ is defined as activation function. The results for larger networks are similarly good, but several times the number of network weight update steps are necessary. Moreover, the computation time to perform an update is longer for a larger ANN. Prior studies also showed that other network architectures, such as convolutional neural networks (CNN) or a two-dimensional state representation instead of a one-dimensional state vector, are applicable but do not lead to an improved performance (A_Kaiser 2019). Generally speaking, one reason for using a CNN is the poor scalability of the one-hot encoding. The higher the number of resources, the higher the number of state variables required for the one-hot encoding. However, the problem size considered in the present work seems still manageable for an MLP network of the above mentioned size.

In general, it can be stated that there is not a unique optimal hyper-parameter configuration. The presented results are based on the best knowledge according to various computational experiments.

4.5 Evaluation and benchmarking

The presented RL-modelling alternatives need to be evaluated in several experiments. Each of it covers a range of multiple simulation runs. The experiments aim to test the problem-solving ability of the RL-based order dispatching and the evaluation process is described in this section. It follows a three-step approach. First, the *state of convergence* is defined for that it is said that the agent has finished the learning process and the performance does not change significantly any more. This state is required for the next step, the evaluation of the performance based on the KPI values. Lastly, the RL-algorithm is compared to benchmark algorithms that represent the current state-of-the-art approach for order dispatching in practice. In addition, the benchmarks help to analyse and understand the agent's policy in more detail.

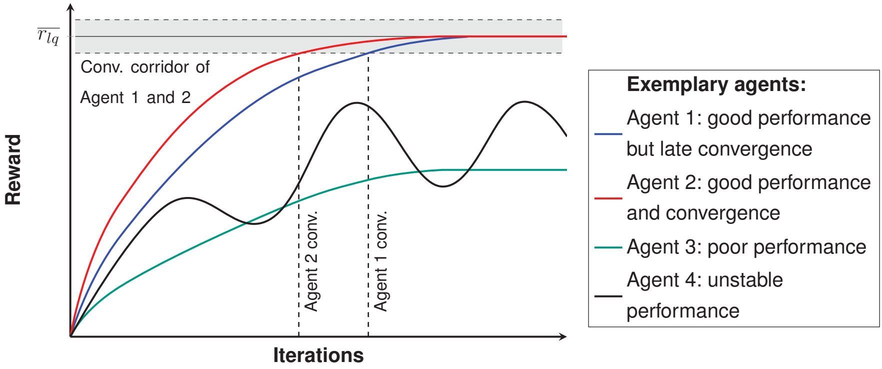


Figure 4.13: Schematic illustration of the course of agent's reward per learning iteration of four exemplary agents. For Agents 1 and 2 the convergence corridor and point of convergence are depicted, too.

To get a first understanding of the range of possible results that can be observed in experimental runs, Figure 4.13 sketches four characteristic reward curves for fictional RL-agents. The reward is displayed over time, i.e. learning iterations or dispatching actions. Note, the time between two consecutive actions differs, however, it is sufficiently small to be neglected for these figures. The reward is used as the primary and most native evaluation measure to analyse the agent's performance, as it shows the feedback the agent receives. It indicates if the agent can maximize its reward over time. Further, it is assumed that all shown agent rewards are scaled to the same value range. For instance, Agent 1 shows good performance, but Agent 2 reaches the same level earlier, i.e. the training phase is shorter. On the other hand, Agent 3 performs poorly, maybe due to a locally optimal solution, which is likely due to a poor RL-agent modelling as the agent is not directed enough or capable at all to capture the intended behaviour. Agent 4's performance is not stable, which is not suitable in practice. In most cases, an oscillating performance hints that either the agent hyper-parameters are not properly refined or the environment dynamics hinder a stable performance. Hence, a proper reward curve should look like the curves of Agent 1 and 2.

4.5.1 Analysis of convergence

Two elements influence the state of convergence in the approach of the present work. On the one hand, there is the discrete-event simulation. Simulation experiments are, according to Gutenschwager & Rabe et al. (2017), only valid when limited to the phase after the *transient phase*. On the other hand, the RL-agent requires a training period to reach its state of convergence. Chapter 5 determines, without loss of generality, first the simulation's transient phase based on established approaches described by Gutenschwager & Rabe et al. (2017). After that, the RL-agent's state of convergence is computed, as it is assumed that this takes significantly longer than the simulation to converge. Anticipating the results and explanation of Section 5.1.4, it can be stated that the transient phase can be neglected in the application considered in the present work. A comparison with the training period shows that the transient phase is a mere fraction of the time to reach the state of convergence.

Looking next at the agent's convergence in theory, the following applies: although value- and policy-based RL-algorithms introduced in Section 2.3 are proven to converge, there is no generic convergence guarantee for advanced RL-algorithms, in particular with function approximators such as ANN (Sutton & Barto 2018). Therefore, a procedure is developed to compute the state of convergence.

The convergence definition that is used in the following is based on the reward signal as it represents the optimization measure of the agent. So, looking at the reward one can evaluate the agent's performance (see Figure 4.13). If the reward reaches a certain level and does not deviate from that level in the following iterations, it is assumed that the agent has reached its **state of convergence**. The following calculation rule is applied (A_Kaiser 2019):

1. Calculate the average reward $\overline{r_{lq}}$ of the last quarter of the entire experiment run.

2. Check for every intersection i of the reward with $\overline{r_{lq}}$, whether the average reward $\overline{r_{i \to end}}$ from intersection i to the end satisfies the following condition:

$$|\overline{r_{lq}} - \overline{r_{i \to end}}| < \epsilon_{conv} \cdot \overline{r_{lq}} \tag{4.29}$$

   ϵ_{conv} represents an arbitrarily pre-defined threshold. If not stated differently, the threshold is set to $\epsilon_{conv} = 0.005$. Note, the list of intersection indices i is sorted in ascending order.

3. If the condition is met, convergence is assumed, starting from intersection i.

This procedure does not prove or necessarily imply that the RL-agent converges. It computes the point in time from whereof the state of convergence starts, given the agent converges at all. In the present work, the convergence assertion is based on a visual inspection of the course of the reward curve, as shown in Figure 4.13. So, double-checking every experiment visually is a prerequisite and prevents the analysis of artificial results.

Confidence interval investigation To avoid that the mean performance of an RL-agent resembles just a sequence of random events of a single experiment, several simulation runs with different *seed*-values are performed. Seeds are implemented in the simulation to control all stochastic processes, such as machine breakdowns or processing times, and, at the same time, ensure reproducibility of the results. Two simulation runs with the same seed value have the same sequence of stochastic events and, thus, give the same results. There is, however, a minor limitation to that statement. As the behaviour of the RL-agent cannot be seeded due to the nature of the ANN implementation, the results still vary in a small range.

Given the fact that seed values control the simulation experiments, the next question is, how to determine the number of repetitions that are required to get meaningful results. Looking at average performance indicator values (see Table 4.5), the confidence interval concept following Robinson (2014) is applied. The **confidence interval** CI indicates how accurate the average real value can be estimated. The interval is calculated according to the formula:

$$CI = \mu \pm t_{n-1,\beta/2} \cdot \frac{\sigma}{\sqrt{n}} \tag{4.30}$$

Here, μ is the average value, σ the standard deviation, n the number of simulation runs, t the t-distribution with $n-1$ degrees of freedom, and β the confidence level. A frequently chosen confidence level is $\beta = 0.95$. The confidence interval specifies the range around the calculated average within which the real average value lies with a probability of 95%. The real average value can only be reached for an infinite number of simulation runs. Computational results for the confidence interval are shown the next chapter in Section 5.1.4.

4.5.2 Performance evaluation criteria

After having analysed the reward signal as an indication of the agent's learning performance, the state of convergence, and the confidence interval, a concluding statement of the overall system performance is essential to derive insights for decision-makers in practice. Various system parameters are recorded in the simulation at constant time intervals so that a detailed database is given that allows an in-depth investigation of the progression of the performance as well as a benchmarking in comparison to other algorithms or the current real-world performance. The used performance evaluation criteria are listed in Table 4.5 and shortly explained hereinafter (recalling also Figure 4.3).

Table 4.5: Overview of the selected performance evaluation criteria, their relation, and if they are dependent variables or actively varied in experiments (independent).

Evaluation criteria	**Related to**	**Type of variable**
Utilization of machines U	machine	dependent
Waiting time WT	order	dependent
Utilization of the dispatching agent U_{disp}	agent	dependent & independent
Reward R	agent	dependent
Variability $\alpha\text{-}value$	system	dependent
Inventory I	system	dependent & independent
Throughput TP	system	dependent

The **machine utilization** measures the averaged value-adding usage of machines. It is calculated based on the cumulative time all machines have been processing orders in a certain period. Machines that cannot finish processing due to blocking, i.e. a full outbound buffer, are not recorded as utilized. The main objective is to maximize the overall utilization and, hence, reduce any efficiency losses caused by blocking or starvation. The standard deviation of the utilization σ_U may also be considered in the evaluation to see if machines are utilized differently.

The **waiting time** is the main criteria to evaluate the cycle time as order-level performance, i.e. the customer service level in the absence of order due dates. It covers all waiting phases that an individual order undergoes, e.g. in inbound or outbound buffers of machines. Minimizing the average value is the primary optimization objective. Again, the standard deviation σ_{wt} is considered as an indicator for the consistency of the dispatching performance and to evaluate the chance of orders that are "missed" by the dispatching agent and reside in the system for a long time.

The calculation of the **dispatching agent's utilization** differs slightly from the machine utilization calculation. Handling and transporting are value-adding activities and the time required to move to a pick-up location is non-value-adding but necessary and, therefore, included in the measure. The variance of the utilization indicates if the agent has phases with varying workload. As there is just a single agent, no further insights can be obtained from the variation.

As already stated, the **reward** is an artificial evaluation criterion that is calculated as defined in Section 4.4.4.3 and helps to analyse the agent's learning performance. However, in the following chapter it is regarded as a subordinate measure when looking at operational performance analysis.

The **variability** $\alpha\text{-}value$ has its origin in the literature on operating curves by Boebel & Ruelle (1996) and Aurand & Miller (1997), which in turn are based on Kingman's Equation and Little's Law (see Section 2.1.2). The calculation is comprehensively explained in Hilsenbeck (2005) according to the formula:

$$\alpha\text{-}value = \frac{(\text{dynamic flow factor} - 1) \cdot (1 - U)}{U} \qquad 4.31$$

The dynamic flow factor is a rolling indicator of the flow factor, which is the fraction of order cycle time and raw processing time. Without going into further details on the calculation, it is so far sufficient to understand that it is proportional to the flow factor and the inverse of the machine utilization. Hence, by combining both indicators, the $\alpha\text{-}value$ is an information-rich measure of the system's overall performance, where a small value indicates a better performance. For a small value, the utilization is high and the flow factor small, i.e. both are close to the value 1.

The **inventory** level is closely linked to the order waiting time, due to Little's law, and broadly seen as bounded capital. According to Nyhuis & Wiendahl (2012), it is of major concern in operations management, and many PPC-approaches look at the optimal inventory level. In the following, it measures the system's overall performance, too. The same applies to the **throughput** measure, counting the number of finished jobs within a period.

As highlighted in Table 4.5, the utilization of the dispatching agent and the maximum allowed inventory level are varied as independent variables in the following chapter in order to obtain different validation scenarios with heterogeneous system characteristics.

Pareto-optimal solutions Finding an optimal balance for multiple performance measures is non-trivial. To this, the Pareto-concept introduced in Section 2.1.2 is used to evaluate the performance of the adaptive order dispatching agent, when both optimization objectives, i.e. utilization and waiting time, are considered. As a result, a two-dimensional Pareto-optimal frontier can be approximated that allows an investigation of the agent's multi-objective performance and identification of Pareto-optimal RL-agent configurations.

4.5.3 Benchmark algorithms

This sub-section describes the benchmark algorithms that are considered and represent state-of-the-art heuristic approaches that are established in practice as well as in academia (see Section 2.2.2). All heuristics need to ensure that no deadlocks arise throughout the production processes and, at the same time, follow the optimization objectives.

In concrete terms, the heuristics VALID, FIFO, NJF, and EMPTY are developed as benchmarks for the adaptive order dispatching. The heuristics follow the same control flow chart as depicted in Figure 4.6 and Figure 4.7. In particular, just the agent module in Figure 4.7 is replaced with the heuristic decision-making procedure and the reward module is not required.

The **VALID heuristic** is a simple refinement of a heuristic that selects dispatching actions entirely at random. Therefore, the VALID heuristic considers just the set of valid actions $A_{valid,t}$ in every iteration. Although this seems trivial, it is an important benchmark to analyse if the learning algorithm is able to optimize its dispatching policy and thereby outperforms the VALID heuristic, as the RL-policy is initialized with random values.

Next, **FIFO heuristic** selects the order that is ready for dispatching with the longest waiting time within the system. It refers to the time of order release and from then on the total waiting time is added up. Hence, it also mirrors the sequence of order arrival. After selecting the order, the destination needs to be determined. In doing so, that machine out of the set of all machines in the same machine group is selected, which has the lowest number of orders in its inbound buffer. This rule levels the workload and reduces the average waiting time of orders in machine inbound buffers. If the order selected in the first step is finished with processing and needs to be delivered to a sink, it is dispatched to the closest sink. In fact, the FIFO heuristic is a combined FIFO and FLNQ rule.

When focusing on the performance of the dispatching agent, the **NJF heuristic** is a predominant rule in practice (Mönch & Fowler et al. 2013). The evaluation of the agent's utilization is equivalent to the total transport distance, as the transport capacity is limited to one. The order selection for the NJF rule is conditioned to the distance from the agent's current location to the respective order origin. In case of more than one available order at the same location, the secondary selection rule refers to the longest order waiting time. The dispatching destination is, analogously to FIFO, determined via the machine with the lowest inbound buffer fill level within the same machine group or the nearest sink.

EMPTY focuses primarily on machine utilization. The list of available orders is sorted according to their next processing time. Orders that are finished and need to be taken to a sink are assigned with an arbitrarily large number. The order with the shortest time is selected, as for the SPT rule. SPT is known as optimal sequencing heuristic in a one-machine setup. It optimizes the waiting time, by keeping the queue size small. The destination is determined according to the inventory level of the destination inbound buffer, too. So, EMPTY supplies machines that are most likely to run empty first and, in addition to that, raises the inventory in the system as orders to sinks are less likely dispatched.

Eventually, according to Burke & Gendreau et al. (2013), finding high-performing heuristics is difficult for any real-world problem. In the present case, the next order, its destination, the route of the dispatching agent, the machine availability, the inventory level at machines, the order waiting time, and a continuous production process need to be considered at the same time and finely tuned to optimize the operational performance. The computational results in Chapter 5 show how good the heuristics and the RL-agent are able to handle all these.

4.6 Implementation of adaptive order dispatching system

Once the modelling choices are made, the production and decision framework are formalized and the RL-based dispatching algorithm is described, the entire conceptual model and framework can be implemented. This is outlined in the following by pointing out some essential implementation considerations. The implementation is realized in the Python programming language using mainly two libraries: the discrete-event simulation framework `SimPy`[1] and the modular TensorFlow-based library for RL-applications `tensorforce`[2].

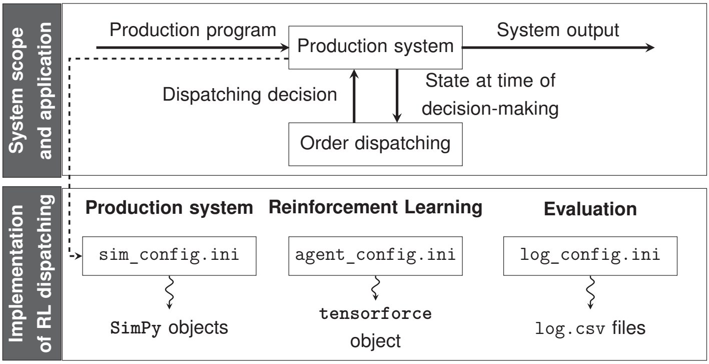


Figure 4.14: Implementation framework.

[1] SimPy – A free discrete-event simulation package based on Python (v3.0). `https://pypi.org/project/simpy/` (accessed on 17.06.2020).

[2] Tensorforce – A TensorFlow library for applied reinforcement learning (v0.4). `https://pypi.org/project/Tensorforce/` (accessed on 17.06.2020).

Figure 4.14 resumes the methodological approach introduced in the front matter of this chapter (see Figure 4.1) and replaces the phases of the approach by the key inputs and outputs that are, at the same time, the interfaces towards the user. First, the `sim_config.ini`-file contains all time-invariant and time-variant information that are required to describe and model the production system, e.g. machine parameters, job shop layout, distances, stochastic processes, and simulation duration. So, based on this configuration file, the simulation is created and all simulation objects are initialized. Each resource type is implemented as a class with separately defined properties and procedures. The process logic of each resource type is based on its tasks and implemented in `SimPy` by *processes* that consist of activities and *events*. Events control the interaction of multiple processes and are executed successively. Recalling the process flow in Figure 4.6, the two most important events are `production.changed()` and `resource.reactivate()`. The two events are part of a main controller unit monitoring all processes and changes happening. Each resource first waits for the main controller for reactivation, before it can select an action and execute it. Reactivation is only triggered if there is a change in the production system state, expressed by triggering the event `production.changed()`. This process flow results in a chain of actions that represents the real-world production process and, in addition, considers the key concepts of an MDP, as all processes are controlled by the main controller.

Next, the RL-agent is created according to the modelling specified in the `agent_config.ini`-file, e.g. state features, reward function, hyper-parameters, and action-mapping. The TRPO-algorithm itself is provided by `tensorforce` as introduced by Schulman & Levine et al. (2015). Lastly, the evaluation phase is based on log-files that are recorded while the simulation is running. It is set up according to the `log_config.ini`-setting.

4.7 Summary of approach and framework

This chapter described a reinforcement learning approach for an adaptive order dispatching in a complex job shop. The dispatching decision addresses the question of which and where should the next order be dispatched to. Reinforcement learning represents an adaptive decision-making algorithm that determines the next action based on the current state and the dispatching policy is continuously adjusted and optimized according to the reward signal. The developed RL-agent aims at machine utilization (resource-related KPI) and order cycle time (order-related KPI). For an extensive evaluation, a simulation-based training environment is presented. Finally, the approach of modelling RL-agents is described (see also Figure A3.1 in Appendix A3), which is, in certain parts, generic enough to be applicable for adjacent production control tasks, such as order release, sequencing, or capacity control. Hence, the requirements stated in Section 3.1 are met, and the next chapter continues with the application and evaluation.

Summarizing the key insights gives the following list:

1. The adaptive control architecture presents a generic decision-making framework that can be either performed by an RL-agent or a heuristic algorithm.
2. TRPO shows, as representative of policy-based RL-algorithms, theoretical advantages in terms of universality and robustness.
3. The action representation of an RL-agent is specified by the application domain. A proper modelling, i.e. a condensed action space, reduces the computational effort.
4. Insights from supervised learning are applicable for the design of the state representation, such as one-hot encoding. However, domain-knowledge is required to define the decision-relevant input data, since including all data is an in-efficient approach.
5. The reward design requires the most domain-knowledge as it directs the learning and adjusting the reward function may change the MDP's underlying optimal policy.
6. A profound evaluation and benchmarking of RL-agents is required as the knowledge on the applicability and plausibility of reinforcement learning in PPC is still limited.

5 Evaluation and computational results

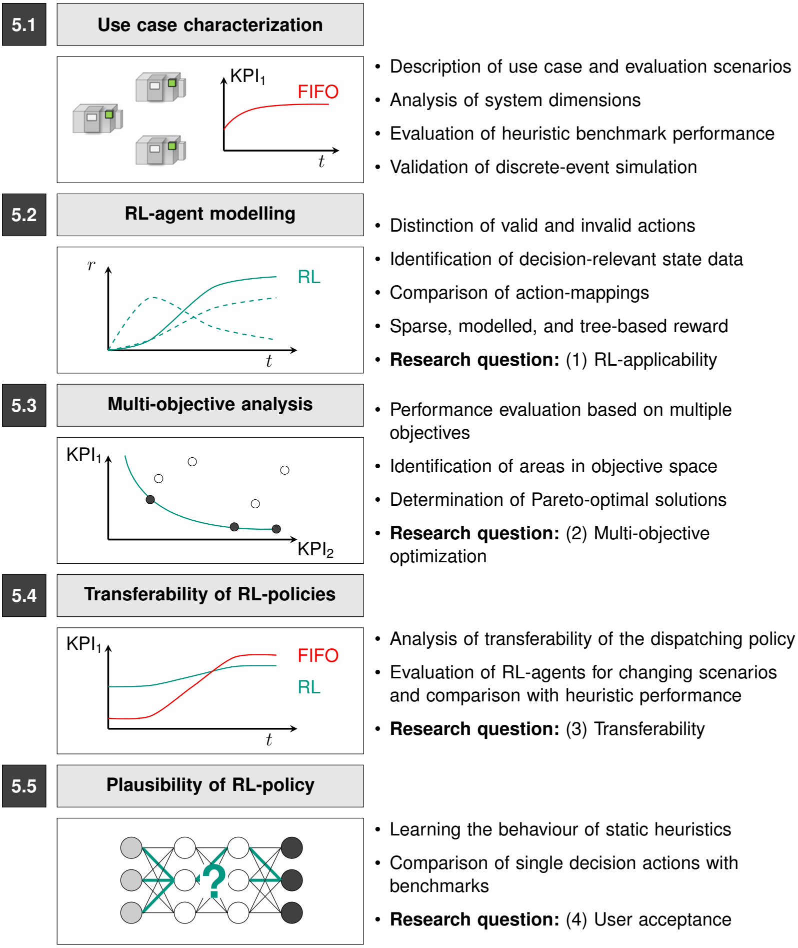
For the evaluation of the RL-based adaptive order dispatching in a complex job shop, first, a use case description is required, emphasizing, in particular, the characteristics of the problem setup. The use case needs to sufficiently fulfil the previously stated requirements in order to rate as validation setup. The subsequent sections refer to the research questions stated in the introduction. All obtained findings are summarized at the end of this chapter.

An overview of industry-wide trends in the semiconductor industry was already offered in Chapter 1. The considered complex job shop system represents a distinctive *work centre* area taken from a wafer frontend manufacturer. Consistent with the industry's characteristics, the company faces short innovation cycles and capital-intensive machinery equipment. The sales volumes are high, but the margins per unit sold low. This combination makes high resource utilization essential to ensure economic success. Moreover, following the theory of operating curves and the lean philosophy, short order cycle times are a similarly crucial operational performance measures.

To face these challenges and the necessity to continue to be highly adaptable to new product generations and rapid innovation cycles, the company concentrates on automation and digitalization of all production processes. Individual work centres are already operated fully automated today. Since there is a broad diversity of product variants, a high degree of flexibility for manufacturing and material flow systems is required, too. In order to ensure the efficiency of fully automated work centres in the presence of a highly dynamic environment, this chapter evaluates the autonomous, learning-based design of an adaptive order dispatching system that controls, in particular, the order material flow. A robust control system is essential to exploit the full capabilities of all resources and avoid unused capacities due to inadequate control decisions.

Section 1.3 stated the research goal and four research questions. The primary goal is to examine an adaptive order dispatching system which is based on reinforcement learning and applicable in complex job shops. The approach summarized in Figure 5.1 states the relevant sub-steps and addresses the research questions.

First, the use case and experiment design are characterized in Section 5.1. These include the validation of the simulation model, together with the evaluation of the transient phase. Moreover, the performance of the benchmark heuristics is presented to get an understanding of the conventional dispatching performance. Next, Section 5.2 focuses on RL-agent modelling alternatives and the first research question on RL-applicability. It shows the results of the design choices introduced in Section 4.4. After that, the RL-agents are compared in the



5.1 Use case characterization

- Description of use case and evaluation scenarios
- Analysis of system dimensions
- Evaluation of heuristic benchmark performance
- Validation of discrete-event simulation

5.2 RL-agent modelling

- Distinction of valid and invalid actions
- Identification of decision-relevant state data
- Comparison of action-mappings
- Sparse, modelled, and tree-based reward
- **Research question:** (1) RL-applicability

5.3 Multi-objective analysis

- Performance evaluation based on multiple objectives
- Identification of areas in objective space
- Determination of Pareto-optimal solutions
- **Research question:** (2) Multi-objective optimization

5.4 Transferability of RL-policies

- Analysis of transferability of the dispatching policy
- Evaluation of RL-agents for changing scenarios and comparison with heuristic performance
- **Research question:** (3) Transferability

5.5 Plausibility of RL-policy

- Learning the behaviour of static heuristics
- Comparison of single decision actions with benchmarks
- **Research question:** (4) User acceptance

Figure 5.1: Overview of the approach to derive computational results and address the research questions.

two-dimensional objective space, and Pareto-optimal solutions are identified, referring to the second research question (see Section 5.3). The third question focuses on the transferability of the learning-based control system, which is covered in Section 5.4. Finally, the learned RL-policy itself is evaluated in order to understand the decision-making and, thereby, increase the plausibility of the black-box approach, which is the fourth research question.

5.1 Complex job shop application use case

In the considered job shop work centre, the *implantation* process is performed on the semiconductor wafers. Currently, the material flow of wafer lots, i.e. production orders, is handled manually by operators. Automation is only implemented for the processing operations as well as handling operations of loading and unloading a lot from the machine buffer into the machine. The aim is the full automation of the area through the use of centrally controlled and freely movable robots, handling the lot dispatching. This work deals with the control problem of optimal dispatching decision-making. In the following, the corresponding boundary conditions and production system characteristics are described. The use case was part of the research project “Intro 4.0 – Empowerment and Implementation Strategies for Industry 4.0” (Lanza & Nyhuis et al. 2018) and funded by the German Federal Ministry of Education and Research (BMBF).

5.1.1 Production system parameters

Within the job shop, a continuous incoming flow of lots can be expected. Hence, it is assumed that there is always a lot available to be released into the system. Besides that, a high product variety with changing machine requirements and fluctuating process times must be processed. As already stated, a production order is considered as a lot and represented by a single transport box. A box full of wafers equals to considerably more processing time, i.e. workload, than a box with just a single wafer.

The layout of the work centre is drawn in Figure 5.2 on the left. The characteristic values and parameters are summarized in Table 5.1. The layout and values represent real-world parameters that are taken from the industrial use case. According to Section 4.6, they define the `sim_config.ini` input file.

Machines M_1 to M_8 process, on average, around 490 orders per day. The machines are grouped into sub-areas according to the work centre layout (see Figure 5.2 on the left). The sub-areas represent separate rooms, i.e. geographically spread locations. One can say that the layout of a semiconductor factory is never optimally arranged and the result of a continuous evolving factory. Moreover, each sub-area has a lift where orders are released into the system and also leave the system again, i.e. representing the source SO and sink

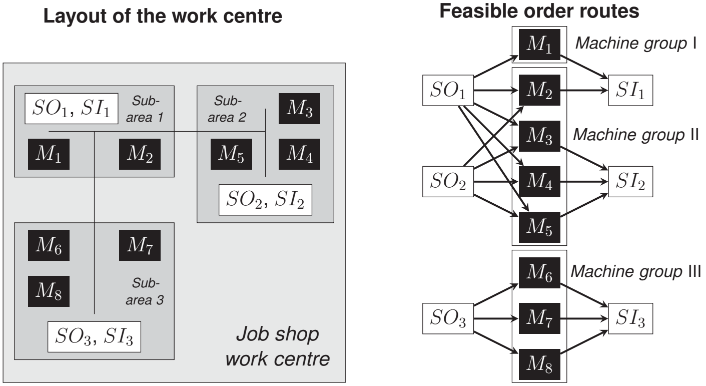


Figure 5.2: Schematic layout and feasible order routes of the considered job shop system.

*Table 5.1: Machine parameters (*in arbitrary time unit, TU).*

Machine	Machine group	Sub-area	Buffer capacity	Demand rate	Mean process time*	MTBF*	MTOL*
M_1	I	1	*scenario-specific*	63.3	19.7	1158	174
M_2	II	1		43.8	28.1	1368	156
M_3	II	2		31.8	39.5	1506	156
M_4	II	2		36.4	30.8	502	108
M_5	II	2		50.1	25.3	1578	138
M_6	III	3		77.7	16.3	972	102
M_7	III	3		80.5	15.6	750	78
M_8	III	3		105.0	11.0	888	162

SI resources. Hence, the source and sink locations fall at the same place. The sub-area determines also, which product order variants arrive at the respective source. For example, the first source releases orders for M_1 and M_2, the second for M_3 to M_5, and the third source for M_6 to M_8 (see also Table 5.1).

Three machine groups are defined according to Table 5.1, indicating which machines are technically able to process the same orders. One can see that the machine group and sub-area definitions do not match for the first and second machine group and sub-area. That is why, for instance, jobs from the first source can be also dispatched to machines in the second sub-area. In order to clarify the difference, and because it is an important feature of the use case, all feasible order routes between sources, machines, and sinks are depicted in Figure 5.2 on the right, including the machine group and sub-area definitions. For example, source SO_1 releases orders for machines M_1 and M_2, according to the sub-area definition. However, an order for machine M_2 may be also dispatched to an alternative machine in the same machine group and, hence, machines M_3 to M_5 are also reachable.

Moreover, Figure 5.2 on the right reveals that completed orders are always dispatched to the closest sink. Furthermore, one can see that the action subset $A_{M \to M}$ is not required in this use case, as all orders are just processed on a single machine within the work centre, and the next process step is always outside the system scope in another work centre. Hence, the available action subsets introduced in Table 4.2 can be reduced by one subset to: $A_{idle}, A_{empty}, A_{S \to M}$, and $A_{M \to S}$.

Each machine has a certain number of so-called load ports representing the inbound and outbound buffer capacities (see Figure A2.1 of Appendix A4). The buffer capacities are varied in the computational experiments and, therefore, defined later in the course of this section when the experiment scenarios are described (see Section 5.1.2). The lots to be processed are loaded to the machine from the outside, and within the machine, they are unloaded and processed automatically. After processing, the wafers are stored again in the same transport box in an outbound load port. So, after processing the entire lot, the box can be accessed from the outside of the machine.

Due to the variable numbers of wafers per lot, the process times per order vary between 0.5 and 150 time units (TU). They are modelled as exponential distribution with the mean values stated in Table 5.1. For reasons of simplicity, process times are shown in the machine parameter list, although they are assigned to the order (see Figure 4.2). The MTBF and the MTOL are known for every machine, based on analyses of historical data on scheduled maintenance activities as well as unexpected downtimes due to breakdowns (see Table 5.1). Again, the exponential distribution is assumed.

The layout of the work centre is given by the distances that are outlined in Table 5.2. As stated before, the entire system is operated by a single dispatching operator and the transport capacity is one. An average speed v is assumed. The speed is specified according to the experiment scenario in Section 5.1.2. For loading and unloading lifts, i.e. sources and sinks, the operator needs in total $0.5\ \mathrm{TU}$, which also equals the handling time at machines. The system is operated in a three-shift model and breaks are neglected as a secondary operator jumps in when the primary operator takes its break.

Lastly, orders are released at the sources. Sources continuously provide new orders, i.e. whenever an order is taken from a source a new order is instantly available. The distribution of the next order variant is taken from the demand rate in Table 5.1. The demand rate represents the average orders per machine per day.

Table 5.2: Transport distances (in arbitrary distance unit, DU).

	SO_1, SI_1	SO_2, SI_2	SO_3, SI_3	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
SO_1, SI_1	0	23.7	13.4	9.5	9.5	26.9	24.9	34.1	9.7	9.7	18.1
SO_2, SI_2	23.7	0	34.9	30.7	16.7	6.4	2.7	10.4	28.3	28.3	36.7
SO_3, SI_3	13.4	34.9	0	25.3	18.4	35.6	33.6	43.9	5.7	5.7	6.9
M_1	9.5	30.7	25.3	0	15.1	33.9	31.9	41.0	21.9	21.9	30.3
M_2	9.5	16.7	18.4	15.1	0	19.7	17.7	26.8	15.4	15.4	23.8
M_3	26.9	6.4	35.6	33.9	19.7	0	6.9	15.9	34.4	34.4	42.8
M_4	24.9	2.7	33.6	31.9	17.7	6.9	0	10.7	32.4	32.4	40.8
M_5	34.1	10.4	43.9	41.0	26.8	15.9	10.7	0	41.3	41.3	48.6
M_6	9.7	28.3	5.7	21.9	15.4	34.4	32.4	41.3	0	2.2	9.2
M_7	9.7	28.3	5.7	21.9	15.4	34.4	32.4	41.3	2.2	0	8.1
M_8	18.1	36.7	6.9	30.3	23.8	42.8	40.8	48.6	9.2	8.1	0

5.1.2 Use case evaluation scenarios

The use case needs to depict all characteristics that are necessary to investigate the research questions. So, in order to appropriately answer these questions, multiple scenarios are required to examine the adaptivity and transferability of the RL-agent. Two scenarios are derived from the general use case just described (see Table 5.3). The first scenario is the base case, representing the real-world setup with comparatively small buffer capacities. Moreover, the dispatching transport speed is slow. In the second scenario, the speed of the dispatching operator is increased as well as load port capacities at machines are doubled.

Table 5.3: Parameters specifying the two use case scenarios.

Scenario	**Parameter**	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8
Scenario 1 – slow speed & small buffers	Inbound buffer	2	2	1	2	2	2	2	1
	Outbound buffer	1	2	1	2	2	2	2	1
	Operator speed v	0.3							
Scenario 2 – fast speed & large buffers	Inbound buffer	3	4	2	4	4	4	4	2
	Outbound buffer	3	4	2	4	4	4	4	2
	Operator speed v	1.0							

However, note that, to some extent, the cases are artificial referring to a real semiconductor manufacturer, as machine load port capacities cannot be changed that easily. Varying the operator speed, on the other hand, is more realistic and, for instance, aims at analysing different velocities of a freely moveable robot. Conclusively, the two scenarios enable the evaluation of the dispatching algorithms for different test cases and whether the RL-algorithm can be transferred to other production system setups. Nevertheless, the number of two scenarios is obviously not sufficient to make generic statements. This fact is discussed again in Chapter 6.

As the experimental results in the following confirm, both scenarios differ in multiple ways. For example, the bottleneck is, in the first scenario, the dispatching operator and, hence, route optimization is more critical than in the second scenario. Limited buffer capacities increase the chance of empty inbound buffers, i.e. starving and idling machines, and reducing the transport speed makes sub-optimal operator moves even more critical. In the second scenario, the machines limit the overall system performance and order dispatching needs to focus on levelling of the overall machine utilization and minimization of order waiting times due to possibly higher inventory levels as the buffer capacities are less restrictive. These theoretical considerations are incorporated in the following investigations in order to avert misleading interpretations of potentially artificial results.

5.1.3 Analysis of system dimensions

Before looking at the computational results derived from simulation experiments, the system's capacity dimensions are analysed for both scenarios to gain further system understanding. The estimations focus on machine and transport resources, and the numbers are shown in Table A5.1 and Table A5.2 of Appendix A5.

Apparently, machine capacities are sufficient for the daily demand rate with an overall average uptime utilization of 98.3%, including breakdowns and other downtimes. Just the first machine is slightly overloaded. However, it has to be considered that this rough estimation does not take into account the possibility to change the order assignment to alternative machines in the same machine group and it is just looking at mean values.

It is worth noticing that the machine utilization including downtimes varies, i.e. the average availability of machines differs. Hence, some resources are more reliable than others.

Considering the dispatching operator's capacity dimension, three cases are conceived to take the two use case scenarios into account as the capacity is different in both scenarios due to the varying transport speed. First, for the fast dispatching operator, the total transport and handling workload can be covered by a single transport resource with the utilization of 79.6% in an average estimation. The average case means that transport distances to order pick-up and destination locations are evenly distributed, i.e. assuming the probability of the operator's location relative to the pick-up location is evenly distributed. Second, if the speed is reduced, the utilization would rise to 179.5% in the average case, which clearly indicates a limitation of the system output, i.e. bottleneck. Hence, a third estimation is performed. The slow speed is sufficient with a utilization of 97.9%, when transport distance are, in any case, the shortest possible distances, i.e. the dispatching operator does not need to move to a pick-up location, and it can always continue with another job. This assumes that the transport route is perfectly optimized.

All in all, these estimations, in particular for the operator's capacity, are bound to assumptions that substantially influence the results. However, one can state that the capacities are sufficient in both scenarios, though more constrained in the first scenario.

5.1.4 Benchmark heuristics and validation of simulation model

The application of the benchmark heuristics to the two use case scenarios gives the baseline of the performance. At the same time, the discrete-event simulation can be validated by looking at the heuristics and considering the just described theoretical observations. The heuristics implement the procedures outlined in Section 4.5.3. First, the VALID heuristic is the absolute performance baseline as any optimizing algorithm should outperform the random selection. Especially for learning algorithms, it is still a common and essential benchmark, for instance, to evaluate how consistent the RL-agent differentiates valid and invalid actions. FIFO minimizes the longest waiting time of any order in the system. NJF focuses on the utilization of the dispatching resource, minimizes the transport distance, and empty-handed walks are reduced. Finally, EMPTY supplies machines evenly with orders and the chance of machines idling is reduced, i.e. the utilization is maximized.

Table 5.4 reports the performance indicators of the heuristics for both scenarios. The average utilization U, waiting time WT, inventory I, dispatching agent's utilization U_{disp}, and $\alpha\text{-}value$ are shown. The measures are computed based on the data after the state of convergence has been reached and, additionally, averaged over three simulation repetitions. The reasoning behind the number of repetitions is given in the next paragraphs.

In general, it can be seen that the average inventory in Scenario 1 is lower due to the limited buffer capacities. Similarly, the order waiting time values are smaller, which follows from Little's Law. Moreover, the machine utilization is, on average, lower and the operator utilization higher, what indicates that the limited buffer capacities and the slower dispatching speed results in a bottleneck at the dispatching resource and a higher chance of starvation and blocking of machines. Hence, it can be confirmed that Scenario 1 is said to be more constrained than Scenario 2 (see Section 5.1.3). Finally, the heuristics' performance varies on a larger scale in the more constrained Scenario 1, as there is more potential for optimization.

Table 5.4: Performance of benchmark heuristics in both scenarios.

Heuristic	**Scenario 1**					**Scenario 2**				
	$U[\%]$	$WT[\text{TU}]$	I	$U_{disp}[\%]$	$\alpha\text{-}value$	$U[\%]$	$WT[\text{TU}]$	I	$U_{disp}[\%]$	$\alpha\text{-}value$
VALID	56.3	86.1	**6.1**	94.3	3.74	83.3	**114.1**	18.8	63.2	0.70
FIFO	68.7	**81.4**	6.9	88.3	1.95	84.4	**114.1**	**18.6**	55.4	0.57
NJF	**82.3**	89.8	10.9	**75.3**	**0.64**	84.9	115.3	19.5	**50.6**	**0.51**
EMPTY	68.8	92.6	11.4	83.8	2.31	**86.3**	120.9	20.4	55.6	0.55

Analysing the heuristics in more detail confirms that the dispatching behaviour matches the intentions of the heuristic rules. It is noticeable that VALID scores best in terms of the inventory level in the first scenario, although it selects an action without any context and process knowledge. However, as it randomly selects an order it reduces the waiting time, but also increases the operator utilization.

FIFO focuses on the waiting time and performs best in regard to this measure in both scenarios. The longest waiting orders are mostly orders in outbound buffers of machines. As a result, the inventory level is reduced, too.

NJF performs best in terms of the $\alpha\text{-}value$, indicating that the rule represents a near-optimal dispatching policy in both scenarios, since the $\alpha\text{-}value$ aggregates both performance indicators, utilization and waiting time. In particular in Scenario 1, NJF achieves good results, because it minimizes empty walks at the bottleneck resource, what in turn also results in higher machine utilization.

EMPTY performs similarly well as FIFO in terms of the $\alpha\text{-}value$, but focusing instead more on machine utilization. In particular in the second scenario with high inventory levels, the heuristic reaches the highest utilization. Moreover, the rule pushes inventory into the system and the highest inventory levels are recorded. However, its drawback is that it only performs well if the dispatching operator is not the bottleneck and running at high utilization.

The results for both scenarios show that heuristics suffer from the trade-off and dilemma between both performance objectives. A higher machine utilization comes with longer waiting times, e.g. for NJF and EMPTY, whereas a lower waiting time reduces the machine utilization, e.g. for FIFO. This strengthens the motivation of the research question, how an RL-agent performs in terms of multiple objectives, however, knowing that the underlying trade-off can never be resolved entirely.

Transient phase Using discrete-event simulation for evaluation requires the specification of the transient phase in which the simulation has not yet reached a stable performance (Gutenschwager & Rabe et al. 2017; Law 2014). The transient phase is, in most production simulations, caused by the process of filling up the system with orders and ends when all buffers reach their average fill level. As there is no standard approach to determine the transient phase, a simple visual inspection is chosen, according to the recommendations of Gutenschwager & Rabe et al. (2017). The performance indicators are plotted directly over the simulation time, and the end of the transient phase is visually estimated.

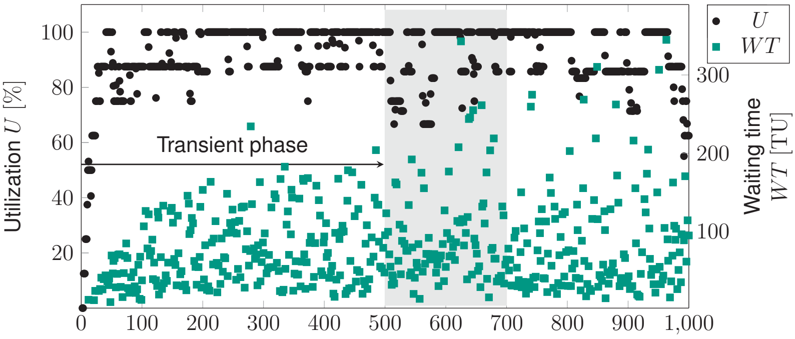


Figure 5.3: Exemplary utilization and waiting time data for the first 1000 iterations of the NJF heuristic and highlighting the estimated end of the transient phase between iteration 500 to 700.

Looking at Figure 5.3, the course of the utilization and waiting time for the NJF heuristic is depicted. After just a few 500 to 700 iterations[1] a performance level is reached that does not change more than the oscillation due to the stochastic production processes.

Comparing this with the usual RL-agent training duration of over one million iterations, the transient phase is negligible and does not influence the learning process. Therefore, the transient phase is stated here once, but not explicitly considered in the aftermath. Moreover, it is assumed that disabling the agent's learning during the transient phase, such as accomplished by Waschneck (2018), is not required. This is also due to the reasoning that the agent should be able to act in any situation in the best way, as long as the agent is able to generalize. Hence, a system with initially just a few orders is similarly important to the agent as a fully loaded system.

[1] From hereupon the time axis shows the iteration steps, i.e. the number of interactions between the simulation environment and the dispatching agent, which equals the number of dispatching actions performed and correlates with the simulation time.

Number of repetitions After determining the transient phase, the simulation duration and the appropriate number of simulation repetition runs are determined. It is worth noticing, looking at Figure 5.3, that the raw data that goes into the performance indicators in Table 5.4 fluctuate considerably. The standard deviations and the coefficients of variation CV for each performance indicator are shown in Table 5.5 for the first scenario. Similar figures apply for the second scenario just on another scale. The performance variation is caused by the stochastic production processes and the dispatching heuristic itself. The influence of the heuristic can be seen, as the coefficient of variation, i.e. the ratio of standard deviation and mean value, is not consistent for all heuristics. Recall that the shown values are already based on three simulation repetitions. A comparison with the RL-agent's performance in terms of variation is performed at a later stage in Section 5.2.2.

Table 5.5: Comparison of performance indicators' mean, standard deviation, and coefficient of variation (CV) for the heuristics in the first scenario.

Heuristic	$U[\%]$	σ_U	CV_U	$WT[\text{TU}]$	σ_{WT}	CV_{WT}	I	σ_I	CV_I
VALID	56.3	20.9	0.37	86.1	**97.0**	**1.13**	**6.1**	**2.5**	0.41
FIFO	68.7	20.9	0.30	**81.4**	99.6	1.22	6.9	2.7	0.40
NJF	**82.3**	24.6	0.30	89.8	118.3	1.32	10.9	**2.5**	**0.21**
EMPTY	68.8	**20.1**	**0.29**	92.6	110.1	1.19	11.4	3.8	0.32

Table 5.6: Confidence interval (CI) for the waiting time taken from NJF heuristic data in the first scenario and based on the confidence level of $\beta = 0.95$.

Repetition	WT **per repetition**	σ_{WT} **per repetition**	$\pm CI$**-value per repetition**
1	89.2331	116.3701	0.00057
2	90.3584	119.5932	0.00059
3	89.8561	118.8976	0.00058
WT **for all repetitions**	89.7958		
σ_{WT} **for all repetitions**	0.5627		
$\pm CI$**-value for all repetitions**	0.6367		

The simulation duration and number of repetitions are determined with the help of Table 5.6 and the confidence interval concept introduced in Section 4.5.1. The table shows the figures with $\beta = 0.95$ for the NJF heuristic and the waiting time criterion, as the waiting time shows the highest coefficient of variation 1.32 (see Table 5.5) and is, therefore, an appropriate worst-case estimation for all other heuristics and performance measures. One can see that, according to the CI-value per repetition in the rightmost column of Table 5.6, the confidence is high enough to ensure that the simulation duration, here in total two million iterations, is sufficient. On the other hand, the CI-value in the lower section of Table 5.6 indicates that three repetitions, which is the default for any experiment in this chapter, gives accurate overall results, too. However, as the CI-value is roughly 0.5 an interpretation of performance values in decimal digits is not recommended.

Validation of discrete-event simulation The conclusion of the first computational results demonstrated so far is that the discrete-event simulation is regarded as accurate enough to represent the real-world manufacturing area. First, an in-depth verification of the simulation model was performed by multiple parties throughout this research and leaves a negligible chance of minor errors in the implementation. Second, several interviews with industrial experts confirm that the results of the benchmark dispatching heuristics are appropriate and match real-world values. Third, the heuristics perform as one would expect from their theoretical description. Lastly, as Table 5.4 reveals, the fundamental trade-offs between the logistical performance indicators can be confirmed, too.

Based on these findings, the following sections focus on the RL-algorithm evaluation. It is from now on assumed that the loss of accuracy between the simulated and real-world environment does not negatively influence the interpretation of the results. Of course, this fact is still considered in the discussion in Chapter 6.

5.2 Performance evaluation of Reinforcement Learning algorithm

The focus from hereupon is on the RL-algorithm. First, three further problem-specific parameters are introduced in addition to the hyper-parameters already defined in Table 4.4. These parameters are defined according to the experiments shown in Figure A6.1 in Appendix A6. The maximum number of recursions, when an invalid action is chosen, is set to 5 repetitions and, if the recursion limit is reached, the idle-action is performed with a time duration of 2 TU (see Section 4.4.2 for the details on the invalid action handling procedure). Both parameters support the RL-agent in the learning process, but have almost no effect on its final performance, as the number of invalid actions is negligibly small for a converged RL-agent. Numbers on the rate of invalid actions are reported later in Table 5.7. The episode length, i.e. the number of actions within an episode, is set by default to 100. This allows the agent to operate for enough iterations to evaluate the performance accurately.

Moreover, for the first presented RL-agents, the action subsets introduced in Table 4.2 are reduced to the subsets $A_{S \rightarrow M}$ and $A_{M \rightarrow S}$, and the direct action-mapping is used. Any moves that are not related to an order, i.e. A_{empty} or idling A_{idle}, are not considered. Later, RL-agents are investigated that use these actions, too. This assumption is, amongst others, due to reasons of comparability with the benchmark heuristics that also do not consider these action subsets. The explicit numbering of in total 20 feasible actions is shown in Table A7.1 in Appendix A7. These are the following (see also Figure 5.2):

$$A_{S \rightarrow M} := \{a_{SO_1 \rightarrow M_1}, ..., a_{SO_1 \rightarrow M_5}, a_{SO_2 \rightarrow M_2}, ..., a_{SO_2 \rightarrow M_5}, a_{SO_3 \rightarrow M_6}, ..., a_{SO_3 \rightarrow M_8}\} \quad 5.1$$

(12 actions)

$$A_{M \rightarrow S} := \{a_{M_1 \rightarrow SI_1}, a_{M_2 \rightarrow SI_1}, a_{M_3 \rightarrow SI_2}, a_{M_4 \rightarrow SI_2}, a_{M_5 \rightarrow SI_2}, a_{M_6 \rightarrow SI_3}, a_{M_7 \rightarrow SI_3}, a_{M_8 \rightarrow SI_3}\} \quad 5.2$$

(8 actions)

For that given scope, the RL-agent needs to recognize valid and invalid actions in any state. This is investigated first in Section 5.2.1. Furthermore, the agent needs to optimize its performance based on the learning data it receives through the interaction of state information, action selection, and reward feedback. Therefore, the results in Section 5.2.2 reveal insights into the modelling of RL-agents. Section 5.3 continues with the investigation of the multi-objective performance.

The following sub-sections only show the results for Scenario 1 in the main matter, and the results for Scenario 2 are collected in the Appendix. If not stated differently, three repetitions are computed and the performance measures are reported as average values after the state of convergence. Note that the average is calculated for the time-independent KPIs, e.g. waiting time, as a simple mean and for the time-dependent KPIs, e.g. utilization and inventory, as weighted mean considering the actual time per measurement interval, i.e. iteration or episode, as weight factor.

Correlation of all performance indicators Figure A8.1 in Appendix A8 summarizes the performance measures of all experiments presented in this chapter, i.e. heuristics as well as RL-agents. It shows two-dimensional scatter plots for all possible combinations of the evaluation criteria machine utilization U, waiting time WT, dispatching operator's utilization U_{disp}, throughput TP, $\alpha\text{-}value$, and inventory I (according to Table 4.5). These correlations are kept in mind when the results are explained. For example, the waiting time and inventory are correlated confirming Little's Law. Additionally, the machine utilization is a nearly equivalent measure for the throughput.

5.2.1 Validity of action selection

This sub-section starts with the results for the modelled reward, and after that, the sparse rewards are presented. The first investigation primarily focuses on the distinction of valid and invalid actions and the effect of rewarding different action subsets separately.

Modelled reward Table 5.7 summarizes the results for state information S_{VA} and reward function r_{const} in the first scenario (see Table A9.1 for Scenario 2). The factors ω_1 and ω_2 determine the constant reward depending on the selected action subset. The constant reward is used to evaluate whether the RL-agent is able to distinguish valid and invalid actions, and the action subset weights should measure if there is a difference between the two action subsets that needs to be considered in the aftermath.

Table 5.7: Mean performance results for RL-agents receiving state information S_{VA} and reward signal r_{const} with varying factors ω.

Agent	**Reward and weight factor**	**Scenario 1**				
	$(\omega_1 \widehat{=} A_{S \to M}, \omega_2 \widehat{=} A_{M \to S})$	$U[\%]$	$WT[\text{TU}]$	I	Share of $A_{invalid,t}[\%]$	Iteration of convergence
1	$r_{const}, \omega_1 = 1, \omega_2 = 0$	**66.5**	139.2	17.4	0.18	24,000
2	$r_{const}, \omega_1 = 0.5, \omega_2 = 0.5$	64.3	97.5	9.7	**0.16**	**20,333**
3	$r_{const}, \omega_1 = 0, \omega_2 = 1$	60.3	**73.4**	**5.7**	0.18	37,000

The results reveal that an RL-agent that is based solely on the simple state representation S_{VA} with binary state information and rewarded for valid actions is able to learn a meaningful dispatching policy. The learned policy outperforms the VALID heuristic in at least some performance measures (see Table 5.4). Hence, in case the state representation or reward function are hard to define and not known upfront, this simple configuration is able to achieve reasonable results.

The $A_{invalid,t}$ values shown in Table 5.7 represent the rate of selecting an invalid action per 100 actions. The rate is an average for the last one million actions when the state of convergence is already reached. The number of invalid actions decreases from an initial average rate of roughly two invalid actions per performed valid action to the values depicted in the table. In conclusion, the RL-agent can successfully learn the validity of actions as a rate of 0.16%

invalid actions is negligible. So, it is not explicitly considered in the following computational results.

Moreover, changing the weights indicates that there is a substantial difference between the action subsets. First, for Agent 1 the action subset $A_{S \to M}$ is enforced by ω_1 what increases the waiting time and inventory as orders are pushed to the machines. Likewise, the machine utilization is improved because the chance of empty inbound buffers is reduced. However, the agent is not entirely neglecting actions from subset $A_{M \to S}$ whose respective actions are not rewarded. Analogous considerations can be made for Agent 3, which emphasizes $A_{M \to S}$ and, thereby, minimizes the order waiting time. In other words, changing the weights results in a transition of the two concepts: *push* and *pull* production.

Sparse reward The same computational experiments are also performed for the sparse reward design. Here the reward function $r_{const,ep}$ is used with an episode definition analogous to the previous setting. For instance, the episode ended after 100 valid $A_{S \to M}$ actions. The results are depicted in Table A9.2 in Appendix A9.

However, these agents do not perform reasonably well. They take much more time to converge and achieve a stable performance. In contrast to Agents 1 to 3 that converge within below $100,000$ iterations in Scenario 1 (see Table 5.7), the sparse agent for the simple constant reward function $r_{const,ep}$ requires even above two million iterations (see Table A9.2 in Appendix A9). The reason for that is the less frequent learning feedback the RL-agent receives in the sparse case.

All in all, providing rewards only based on the episode end, e.g. end-of-production targets, leads to a significantly longer learning period. Although theoretically well-suited, the sparse reward comes with considerable disadvantages when applied in production control applications. Rewarding actions directly via a modelled reward guides the RL-agent more quickly and insistently in the direction of the objectives. Therefore, the sparse reward function is not further investigated and it is referred to (A_Theiß 2019) for more computational results that confirm these findings.

5.2.2 State, action, and reward design alternatives

Next to learning valid actions, RL-agents need to capture the interplay between state information, action selection, and reward feedback in order to maximize the accumulated reward. Hereinafter, each modelling element, i.e. state, action, and reward, is investigated in depth.

The alternative modelling approaches reveal decisive insights into the capabilities of a data-driven, RL-based order dispatching. Note that the values shown for the standard deviation in Table 5.8 are investigated afterwards in Section 5.2.3.

Table 5.8: Mean performance results for RL-agents with varying state information and reward signals, aiming to optimize specific production performance indicators.

Agent	State	Reward	Scenario 1						
			$U[\%]$	σ_U	$WT[\mathrm{TU}]$	σ_{WT}	I	σ_I	α-$value$
2	S_{VA}	r_{const}	**64.3**	22.7	97.5	107.8	9.7	2.9	2.81
4	S_{VA}, S_{AT}	r_{const}	60.6	**20.8**	109.6	109.0	10.8	3.1	3.81
5	S_{VA}, S_{WT}	r_{const}	60.9	22.8	**90.1**	**97.1**	**7.5**	**2.8**	**2.72**
6	S_{VA}	r_{util}	76.5	**22.8**	102.0	120.8	12.1	2.5	1.39
7	S_{VA}, S_{WT}	r_{util}	73.8	23.1	105.2	120.7	12.1	2.8	1.57
8	S_{VA}, S_{AT}	r_{util}	76.5	23.8	**98.6**	**116.4**	**11.8**	2.8	**1.27**
9	S_{VA}, S_{BEN}, S_{BEX}	r_{util}	**78.0**	22.9	101.6	121.0	12.1	**2.3**	1.28
10	S_{VA}	r_{wt}	54.7	25.5	80.2	109.4	7.2	2.9	2.73
11	S_{VA}, S_{WT}	r_{wt}	**55.7**	**25.4**	**76.0**	**104.7**	**6.4**	**2.7**	**2.45**

Table 5.8 compares the alternative modelling approaches, which are introduced in Section 4.4.3 (see Table A9.3 in Appendix A9 for Scenario 2). The state vector size varies from 20 entries for S_{VA} to up to 40 entries for S_{VA}, S_{AT}. The reward is the same for both action subsets $A_{S\rightarrow M}$ and $A_{M\rightarrow S}$, i.e. equal action subset weights.

The numbers disclose that the influence of the modelling on the performance is significant. In the first place, the multi-objective performance is not considered in detail. Hence, an agent modelling is called “good” or “better” with respect to a single referred performance measure and not in terms of the Pareto-definition.

The intuitive presumption that more state information is better in any case, cannot be confirmed when comparing, for instance, Agents 2, 4, and 5. The latter two agents have the same constant reward weights as Agent 2 but are extended with further information on the required action times and order waiting times. However, their performance is not better for all performance indicators. Just an improvement of the waiting time and inventory is achieved for Agent 5, when including the waiting time information in the state vector. Therefore, more information is not necessarily beneficial for the RL-agent, as it does not generally support the agent to accumulate higher rewards.

To further validate this presumption, additional RL-agents are investigated and shown in the table. r_{util} rewards the average machine utilization and r_{wt} minimizes the average waiting time. For both scenarios, a positive improvement and optimization of the rewarded performance indicator are conceivable. For instance, modifying Agent 2 to Agent 6 increases the average machine utilization.

Comparing Agent 7 to Agent 6, it is perceived that providing additional state information on order waiting times to an agent that is rewarded for machine utilization does not increase its performance. This confirms the previous results. In contrast, enhancing the state vector with reward-relevant information enables the agent to optimize the specific performance indicator. For machine utilization r_{util} these observations are apparent for Agents 8 and 9. For the waiting time r_{wt}, Agent 11 depicts a reduced waiting time based on the additional information on order waiting times.

However, the dilemma of contradicting objectives prevails and the optimization of one objective simultaneously impairs another performance indicator. Looking at Agents 6 to 11 and, in particular, at the α-$value$, these agents are as such not able to improve multiple performance indicators significantly at the same time.

In conclusion, the combination of state information with an appropriate reward function that utilizes all state entries shows good results, e.g. Agents 9 and 11. Still, it might occur that these agents just optimize a single objective to the disadvantage of others. This investigation is continued in Section 5.3. The number of iterations, i.e. dispatching actions, until the state of convergence is reached increases for agents with an extended state vector. This is due to the increased number of relationships between state, action, and reward that need to be captured and updated accordingly in the policy. Additionally, an extended state vector adds neurons and weights to the ANN that slightly raise the computation time of a network update.

Episode design The episode design regulates the policy update procedure and reward processing. Table 5.9 summarizes the results for varying episode designs for two configurations with state vector $S_{VA}, S_{AT}, S_{BEN}, S_{BEX}$ and S_{VA}, S_{AT}, S_{WT} as well as reward signal r_{util} and r_{wt} (see Table A9.4 in Appendix A9 for Scenario 2). These state-reward-pairs already showed a good performance in the previous experiments.

Looking at all results and comparing it to Table 5.8, the influence of the episode design is less significant and the performance does not vary a lot. Moreover, as discovered in the previous results, all agents rewarded with r_{util} consistently come with higher machine utilization and the ones rewarded with r_{wt} reach a lower average waiting time.

Analysing the results in more detail, the optimization approach of the TRPO-algorithm needs to be taken into account. Policy gradient methods increase during the optimization (learning) phase the likelihood of an action execution relative to its impact on the cumulative reward received in that episode. If certain actions result in a high cumulative reward, the likelihood of executing these actions is increased in the policy. In addition to that, an episode ends when the limit is reached. Hence, there is a trade-off between the action subsets if just one subset increases the episode counter, but both subsets are rewarded. When an episode ends, for instance, after 100 $A_{M\to S}$ actions, the RL-agent collects additional reward by performing $A_{S\to M}$ actions without getting closer to the episode end. The agent learns this relationship and increases the likelihood of those actions. Thus, the higher machine utilization for the $A_{M\to S}$ episode limit and r_{util} for Agent 14 can be explained. Analogous considerations can be made for Agent 17, where the episode type $A_{S\to M}$ is used and the waiting time is minimized.

Table 5.9: Mean performance results for RL-agents with fixed state information and reward functions r_{util} and r_{wt} when varying the episode design.

Agent	State	Reward	Episode design	Scenario 1			
				$U[\%]$	$WT[\mathrm{TU}]$	I	α-*value*
12	S_{VA}, S_{AT}, S_{BEN}, S_{BEX}	r_{util}	100 $A_{valid,t}$	79.5	96.3	12.0	1.05
13			100 $A_{S\to M}$	80.0	**90.0**	**10.8**	**0.88**
14			100 $A_{M\to S}$	**81.4**	95.3	12.2	**0.88**
15			100 time steps	79.3	95.0	11.7	1.08
16	S_{VA}, S_{AT}, S_{WT}	r_{wt}	100 $A_{valid,t}$	52.8	80.9	6.9	2.88
17			100 $A_{S\to M}$	54.4	**72.8**	**5.8**	2.61
18			100 $A_{M\to S}$	52.6	83.8	7.6	3.02
19			100 time steps	**57.4**	76.6	6.8	**2.48**

The introduction of a time-based episode type for Agent 15 and 19 establishes a temporal link between production processes and the agent's behaviour. In this case, the agent must optimize the number of any action during the episode. For the reward r_{wt}, this is achieved by implicitly increasing the utilization through as many dispatching actions as possible and, at the same time, keeping the waiting time as low as possible.

Action-mapping Besides the episode design, the action-mapping is an important RL-design feature. The two action-mapping alternatives direct mapping and resource mapping, introduced in Section 4.4.2, are investigated. All agents converge, hence, both mappings work in principle (see Table 5.10 for Scenario 1 and Table A9.5 in Appendix A9 for Scenario 2).

Nonetheless, the direct mapping is easier in terms of computation time to capture and optimize for the RL-agent, because, for instance, Agent 12 has just 20 actions (see Table A7.1 in Appendix A7) to select from, Agent 22 has an action space of 122 entries (eleven resources to the power of two, plus one idle action), and 32 actions for Agent 20 (eleven empty actions and one idle action in addition to the usual 20 actions). This is why the agents also need significantly more time until they converge (direct mapping: $475,917$ iterations on average, resource mapping: $5,935,167$ iterations on average for Scenario 1).

Table 5.10: Mean performance results for RL-agents with fixed state information and reward signal when varying the action-mapping as well as the executable action subsets.

Agent	State	Reward	Mapping	Action subsets	Scenario 1			
					$U[\%]$	$WT[\text{TU}]$	I	$\alpha\text{-}value$
12	$S_{VA}, S_{AT}, S_{BEN}, S_{BEX}$	r_{util}	direct	$A_{S \to M}, A_{M \to S}$	79.5	96.3	12.0	1.05
20			direct	$A_{S \to M}, A_{M \to S}, A_{empty}, A_{idle}$	**80.6**	**95.0**	**11.8**	**0.95**
21			resource	$A_{S \to M}, A_{M \to S}$	**66.4**	84.8	**8.1**	1.97
22			resource	$A_{S \to M}, A_{M \to S}, A_{empty}, A_{idle}$	64.4	**82.7**	**8.1**	**1.96**
16	S_{VA}, S_{AT}, S_{WT}	r_{wt}	direct	$A_{S \to M}, A_{M \to S}$	**52.8**	**80.9**	6.9	**2.88**
23			direct	$A_{S \to M}, A_{M \to S}, A_{empty}, A_{idle}$	48.9	83.1	**6.5**	4.1
24			resource	$A_{S \to M}, A_{M \to S}$	**53.1**	73.4	**4.9**	**3.29**
25			resource	$A_{S \to M}, A_{M \to S}, A_{empty}, A_{idle}$	52.4	**71.9**	**4.9**	3.61

Looking at the results for reward function r_{util}, it is apparent that direct mapping leads to a higher average machine utilization but also higher, however proportionally less, waiting times and inventory levels. So, the overall performance in terms of the $\alpha\text{-}value$ for r_{util} improves for the direct mapping, which cannot be seen as clearly for the reward function r_{wt}. For the latter case, Agents 24 and 25 are in the first scenario able to optimize the waiting time objective unilaterally with the resource mapping. The second scenario is, in general, less affected by the change of the action-mapping.

Agents 20 and 22 are able to chose empty-handed moves and idling as actions. It is noteworthy that these agents perform better in terms of the $\alpha\text{-}value$ than their counterparts, i.e. Agents 12 and 21, except for Agent 20 in the second scenario, which is slightly worse. Hence, the agent is able to use these additional actions in such a way that it is in favour of the objectives. These insights are reconsidered in the discussion and outlook of Chapter 6. However, note that the results for the r_{wt}-rewarded Agents 23 and 25 are not as clear in that case.

In conclusion, providing more freedom in the action selection by using the resource mapping or considering idling and empty moves comes with a substantially extended learning period but increase the performance slightly.

Tree-based reward The last experiments in this sub-section evaluate the tree-based reward. The reward configurations are set as follows (see Table 5.11 for Scenario 1 and Table A9.6 in Appendix A9 for Scenario 2): the latest state representation $S_{VA}, S_{AT}, S_{BEN}, S_{BEX}$ is kept and a flexible look-ahead tree is used, which is limited to at most 1500 tree nodes. Depending on the evaluation function, either $\mathrm{avg}(\bullet)$ is used for the transport distance or $\max(\bullet)$ for the utilization. These configurations are based on the results of previous research undertaken by (A_Behrendt 2019).

Table 5.11: Mean performance results for RL-agents with the tree-based reward calculation and varying the tree evaluation function.

Agent	State	Evaluation	Tree size	Scenario 1			
				$U[\%]$	$WT[\mathrm{TU}]$	I	α-$value$
26	$S_{VA}, S_{AT}, S_{BEN}, S_{BEX}$	Distance	1500	**80.1**	91.0	10.9	**0.91**
27		Utilization	1500	73.0	**86.7**	**9.6**	1.23
12		*Reference Agent 12*		79.5	96.3	12.0	1.05

Table 5.11 reveals that the RL-agents reach a slightly different performance level. Agent 26 is characterized by a high utilization, but also high waiting times and inventory stocks. Nonetheless, it is performing better than Agent 27 in terms of the α-$value$ and is able to outperform the reference Agent 12 in the first scenario. That confirms the presumption that optimizing the distance over several future dispatching actions is reasonable and increases the performance.

In summary, it cannot be stated as conclusively as expected that the tree-based reward calculation achieves better results in any case. Nevertheless, integrating a look-ahead into the reward calculation can have a positive effect on the performance. Moreover, the tree-based reward is compatible with multiple evaluation functions that can consider different objectives.

5.2.3 Performance variance analysis

The mean, upper quantile, and lower quantile of the utilization, waiting time, and inventory for the benchmark heuristics and two RL-agents are shown in Figure 5.4, with the data taken from Table 5.5 and Table 5.8. Note that the two reference RL-agents, Agents 9 and 11, are taken as representative examples. They are primarily chosen based on their mean performance, the first in terms of utilization and the second for the waiting time. Their performance variation is representative for other RL-agents.

The utilization results show that the spread is, by and large, similar for the heuristics and RL-agents. Nearly the same applies to the waiting time. However, the waiting time spread is much larger. Remarkable is the waiting time variation of FIFO since the mean is very close to the upper quantile. Thus, it is confirmed that FIFO enforces short order cycle times on average and, in particular, concerning the upper limit. More interesting are the inventory results. Apparently, the inventory levels of NJF and Agent 9 are varying significantly less than the other agent and heuristics. In fact, Agent 9 shows the smallest coefficient of variation. Hence, the agent indirectly learns to keep the inventory level constant and, thereby, keep the machine utilization on a stable high level, without being explicitly rewarded for it.

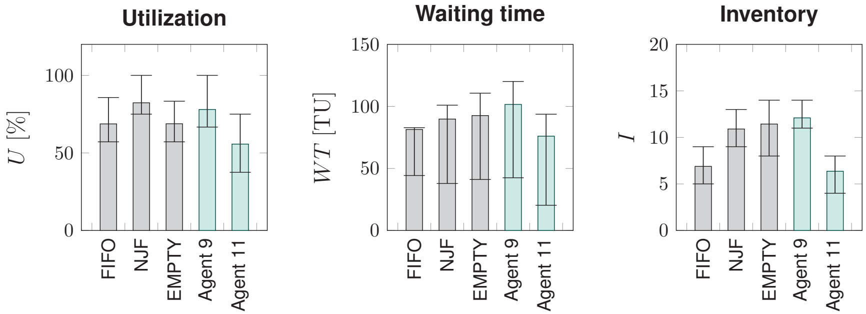


Figure 5.4: Variance analysis comparing the mean, upper quantile 0.75, and lower quantile 0.25 of the heuristic benchmarks and two reference RL-agents for Scenario 1.

In conclusion, the heuristic benchmarks and RL-agents show a comparable level of robustness in terms of performance variance. So, the main reason for performance variations can be seen in the stochastic production processes and less in an unstable dispatching performance.

5.3 Analysis of multi-objective performance

Multi-objective optimization is a special challenge for RL-agents, as for any optimization approach in general (Domschke & Drexl et al. 2015). Improving single performance indicators can lead to poor performance concerning another or all other objectives. Therefore, this section puts the aim at investigating the performance in terms of multiple objectives and particularly the question, how RL-agents are able to handle these.

5.3.1 Multi-objective reward functions

One approach of multi-objective optimization is the following: combine the approaches of action weights from Table 5.7 and rewarding a specific objective as in Table 5.8. Herein, the agent is rewarded based on an objective when selecting a valid action and the reward depends on the action subset chosen. In doing so, not only the reward value but also the action subset and its specific effect on the system performance are integrated to guide the RL-agent. Additionally, the aim is to actively influence the inventory level, being of central importance for managing the performance of production systems in general (Wiendahl & Reichardt et al. 2014).

Table 5.12: Mean performance results for RL-agents with fixed state information and the reward functions $r_{w,util}$ and $r_{w,wt}$ while varying weight factors of the action subsets.

Agent	State	Reward	Weight factor	Scenario 1			
			$(\omega_1 \widehat{=} A_{S\to M}, \omega_2 \widehat{=} A_{M\to S})$	$U[\%]$	$WT[\mathrm{TU}]$	I	α-$value$
28	$S_{VA}, S_{AT},$ $S_{BEN}, S_{BEX},$ $S_L, S_{MF},$ S_{BPT}, S_{RPT}	$r_{w,util}$	$\omega_1 = 0.25, \omega_2 = 0.75$	77.1	**69.4**	**5.0**	0.93
29			$\omega_1 = 0.5, \omega_2 = 0.5$	**82.0**	88.1	10.5	**0.73**
30			$\omega_1 = 0.75, \omega_2 = 0.25$	78.1	120.7	18.0	1.54
31	$S_{VA}, S_{AT},$ $S_{BEN}, S_{BEX},$ $S_L, S_{MF}, S_{WT},$ S_{BPT}, S_{RPT}	$r_{w,wt}$	$\omega_1 = 0.25, \omega_2 = 0.75$	51.5	**71.8**	**4.1**	3.40
32			$\omega_1 = 0.5, \omega_2 = 0.5$	53.7	80.0	6.8	**2.93**
33			$\omega_1 = 0.75, \omega_2 = 0.25$	**58.3**	157.7	19.2	5.12

The two weighted reward functions $r_{w,util}$ and $r_{w,wt}$ are designed for that purpose. An extended state vector is used to have sufficient information available and, in addition to that, analyse the RL-agents' performance for even larger state spaces as investigated so far. S_{WT} is omitted for reward signal $r_{w,util}$ due to its unverifiable effect on the performance. Of course, the simulation duration of one repetition run is considerably longer comparing to the previous agents as the state space is extended, but the performance measures shown are still based on the state of convergence. The results are presented in Table 5.12 for Scenario 1 and in Appendix A9 in Table A9.7 for Scenario 2.

The first conclusion is that for Agents 29 and 32, which have the same reward function as r_{util} and r_{wt} just scaled by the factor of 0.5, the performance is superior to their counterpart Agents 12 and 16. Hence, the additional state information is used in a beneficial way in both scenarios.

Moreover, the expected behaviour regarding the inventory level is observable for Agents 28 and 31. The agents perform actions to sinks more likely when weight ω_2 is increased. This, in turn, lowers the average inventory level and, subsequently, the average waiting time. Vice versa holds for the weight ω_1. However, the effect on the machine utilization is less significant. In both scenarios, the best performance is achieved for the balanced weights $\omega_1 = \omega_2 = 0.5$ for both objectives.

It is noticeable that the agents rewarded with $r_{w,wt}$ show, in general, a relatively poor machine utilization. However, when rewarded for actions towards machines, i.e. weight ω_1, the utilization as well as average waiting time increase significantly. The rise is contradicting to the reward objective. Overall, $r_{w,util}$-agents outperform others in terms of the $\alpha\text{-}value$. This leads to the presumption that the RL-agent benefits from the utilization reward more, i.e. it can capture the correlation between actions, states, and utilization reward more easily than for the waiting time reward.

For both reward signals, the results indicate that an action weighting is beneficial and able to influence the waiting time as well as the inventory as one would expect. However, equal weights do still show the best overall performance, in terms of the $\alpha\text{-}value$. Hence, weighting different action subsets can be an important design feature when looking for specifically tuned RL-agents. Eventually, the $\alpha\text{-}value$ of Agent 29 reveals the best overall performance, including the computational results that are presented next.

Next, the reward function is extended to a weighted sum of the reward functions r_{util} and r_{wt}. In doing so, both objectives are integrated in the reward and the focus can be intuitively adjusted by changing the weights. The results are shown in Table 5.13 for Scenario 1 and in Appendix A9 in Table A9.8 for Scenario 2. In these experiments, the state vector is, again, extended in order to enable the agent to use the entire state information for optimization. Action subsets are not rewarded separately.

Positive results can be obtained looking at the machine utilization and increasing the weight for r_{util} in both scenarios. There is a positive correlation between the average machine utilization and its reward weight factor. The same positive correlation applies to the inventory level. Moreover, the $\alpha\text{-}value$ indicates a superior performance when enforcing the machine utilization reward.

Table 5.13: Mean performance results for RL-agents with fixed state information and a multi-objective reward function when varying weight factors of the multi-objective rewards.

Agent	**State**	**Reward**	**Scenario 1**			
			$U[\%]$	$WT[\text{TU}]$	I	$\alpha\text{-}value$
34	$S_{VA}, S_{AT},$ $S_{BEN}, S_{BEX},$ $S_L, S_{MF}, S_{WT},$ S_{BPT}, S_{RPT}	$0.75 \cdot r_{util} + 0.25 \cdot r_{wt}$	**80.4**	**91.4**	11.0	**0.88**
35		$0.5 \cdot r_{util} + 0.5 \cdot r_{wt}$	77.9	91.7	10.8	1.07
36		$0.25 \cdot r_{util} + 0.75 \cdot r_{wt}$	66.5	97.7	**10.7**	2.14

However, for r_{wt}, only the second scenario shows similar results, i.e. the waiting time reward weight correlates positively with a waiting time improvement. Scenario 1 reveals a somehow unexpected agent behaviour, which might be due to the fact that Scenario 1 is more restrictive than Scenario 2 and, therefore, harder to optimize.

In general, all three Agents 34 to 36 do not show a superior performance to the previously presented experiments for the action subset weights in Table 5.12. Their performance is in between Agents 29 and 32 in terms of the $\alpha\text{-}value$. Hence, weighting objectives is less successful than one might anticipate in the first place. One reason might be that it is hard for the agent to grasp the different objective function elements and a weighted multi-objective reward signal is either too noisy or too flat, i.e. without a clear gradient towards the optimum.

5.3.2 Pareto-optimal Reinforcement Learning configurations

The previous results demonstrated the variety of feasible solutions, depending on the RL-modelling choice. The term solution, in this case, refers to a single RL-agent or benchmark heuristic. Especially when multiple objectives are optimized, the Pareto-concept helps to identify Pareto-optimal solutions (see Section 2.1.2). Figure 5.5 and Figure 5.6 show in two scatter plots the mean performance in terms of waiting time and machine utilization for all RL-agents and benchmark heuristics considered so far in this chapter. Solutions to the right or the lower part of the diagram are preferable.

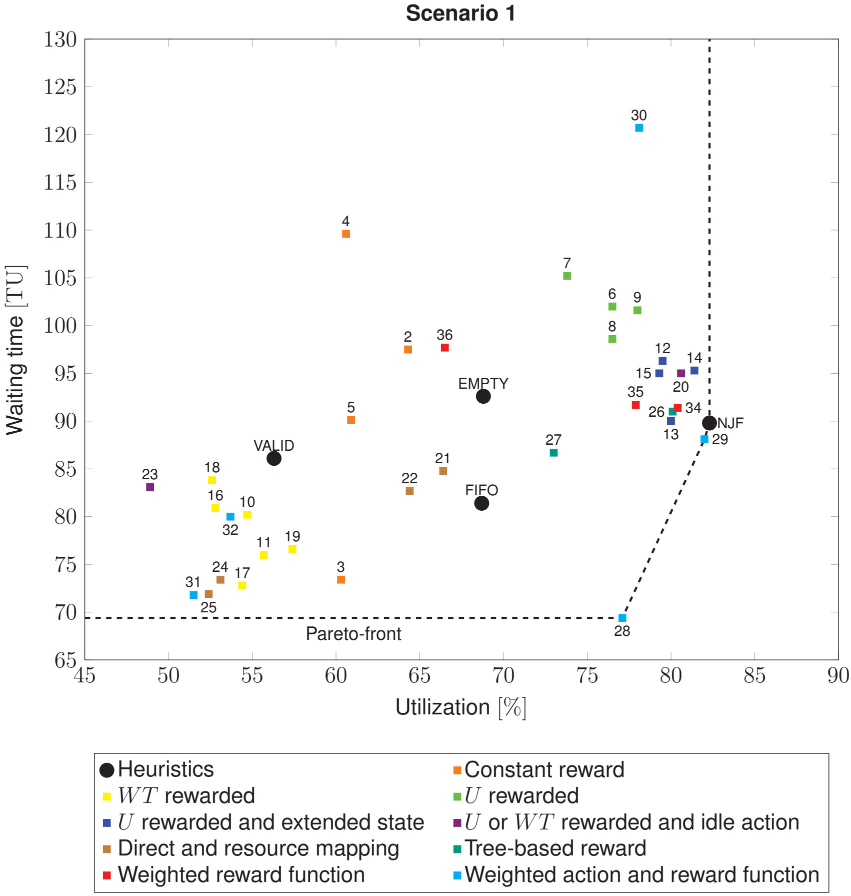


Figure 5.5: Overview of RL-agents and comparison with benchmark heuristics in Scenario 1 (Agents 1 and 33 are not shown as they are well outside the set value range).

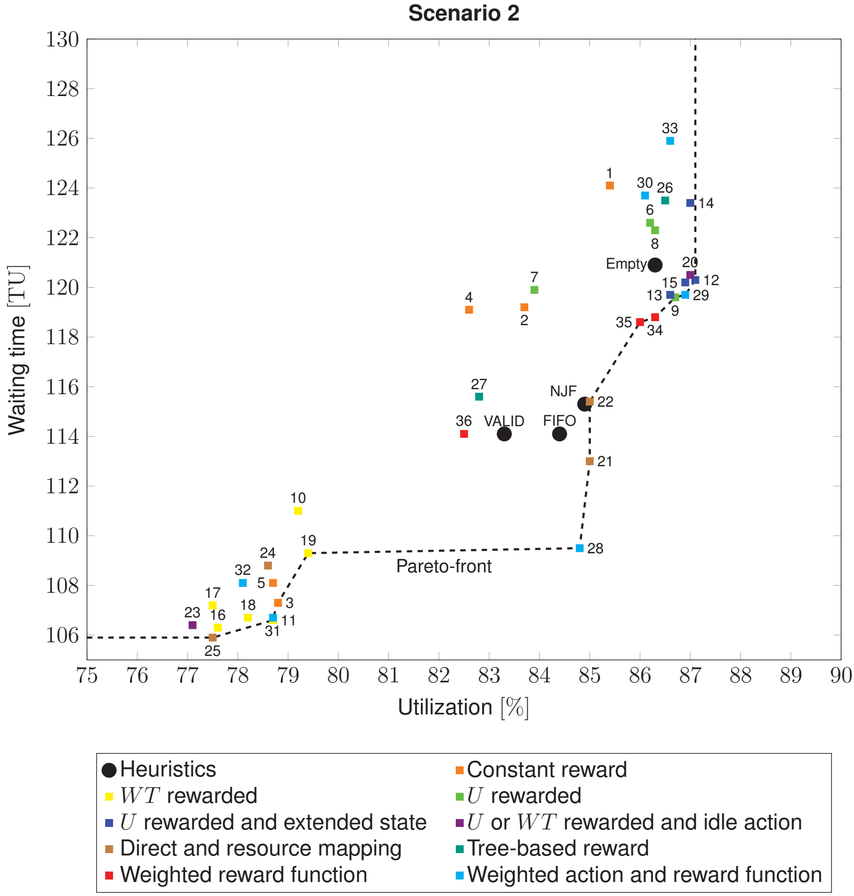


Figure 5.6: Overview of RL-agents and comparison with benchmark heuristics in Scenario 2.

Furthermore, the Pareto-front of all Pareto-optimal solutions that are not dominated by any other solution is indicated in the figure. It is important to state that the displayed Pareto-front resembles just an estimate of the expected border of Pareto-optimality as the theoretically optimal points are unknown and the front assumes that the solution space is continuous. Mostly, RL-agent solutions span the Pareto-front. Just in the first scenario, the NJF heuristic is included. Hence, the performance of the heuristics is Pareto-dominated by RL-agents.

Moreover, reinforcement learning enables a broader coverage of the solution space. This is, in particular, relevant for decision-makers in practice as they are not just interested in a single solution that is the optimal operation point under specific conditions and objectives. Depending on the current situation and objectives, the optimal solution varies. Hence, a set of (Pareto-optimal) solutions is preferred and allows the selection of the best solution according to the current requirements.

The scatter plots also conclude what was already explained in detail before, how the different concepts and ideas behind the RL-agents are evident in the achieved performance. The colours highlight agents that follow a similar concept, i.e. they are previously reported in the same table. For instance, agents with waiting time and utilization rewards are clearly distinguishable in the two-dimensional solution space and the extended state for utilization reward further improves the performance.

Connecting the weighted reward functions, i.e. Agents 34, 35, and 36 highlighted in red, suggests a straight line, which confirms the linear weighted sum modelling choice. Figure A9.2 in Appendix A9 displays the straight line and the agent solutions in a separate scatter plot. Such a result is promising as further solutions, i.e. operating points, can be designed by changing a single weight factor.

Finally, Agent 28 shows a “remote” solution in both scenarios, comparing to the other solutions that are accumulating in similar areas of the solution space. So, using weights for different action subsets combined with the utilization reward $r_{w,util}$ allows reaching new areas in the solution space and extend the Pareto-front. The areas in the solution space with several solutions further support the idea of feasible operating points at which a production system can be operated, what was introduced as central idea of the production operating curves by Nyhuis & Wiendahl (2012).

Comparing both scenarios, some further differences are evident. Scenario 1 has fewer buffer capacities and the operator’s speed is reduced, putting the focus on improving the dispatching performance as the bottleneck resource. Hence, waiting times do not vary so much and are shorter due to, on average, less occupied buffers comparing to Scenario 2. Scenario 2, on the other hand, is less constrained. Hence, the agent’s performance does not spread as much as in Scenario 1, when looking at the x- and y-axis scaling. Moreover, the highest machine utilization is reached for the NJF heuristic in the first scenario. This is counter-intuitive when just looking at its rule description. It would be assumed that EMPTY achieves the highest utilization, as in Scenario 2. However, the optimization of the bottleneck leads in Scenario 1 to the highest machine utilization for the NJF heuristic. This is further analysed in the following Section 5.4.

All in all, the results support the hypothesis that RL-based order dispatching enables production managers to find more desirable operation states than just using static, predefined heuristics. Moreover, reinforcement learning provides a convenient way to identify new control policies by varying the reward function, state representation, or action representation.

5.3.3 Computational effort

Because the multi-objective agents are the most computational complex, a brief overview of the computation time is given in this sub-section. The experiments were conducted on an Intel Xeon E5-2698 v4 2.2 GHz with 20 cores and 256 GB RDIMM DDR4 system memory. Processual parallelization of a single simulation experiment on multiple cores was not activated since no improvement could be achieved. As the used ANNs are rather small comparing to other Deep Learning applications, the computation speed is mostly limited by the sequential processing of the discrete-event simulation, i.e. most computation time is required for the execution in the simulation framework.

Generally speaking, RL-agents converged faster in Scenario 2 than in Scenario 1. Simple agents such as in Table 5.7 or Table 5.8 required, for instance, three to five million iterations, which is equivalent to approximately three to five hours computation time. Hence, one million iterations are approximately equivalent to a one-hour computation. Additional state information and changing the reward function increase the training time, as the time until the state of convergence is reached is considerable longer. The agents presented in Table 5.13, for instance, took 20 to 25 million iterations in Scenario 1 and 10 to 13 million in Scenario 2.

5.4 Transferability of dispatching policy

This section builds on and continues the two-dimensional Pareto-analysis. Herein, the transferability of the agent's dispatching policy is investigated. First, the solutions for both scenarios are integrated to compare their overall performance (see Section 5.4.1). After that, the change from one to the other scenario is examined in Section 5.4.2. The results yield insights into the research question, whether the RL-policy is transferable for changing production system characteristics.

5.4.1 Integrated comparison of multiple scenarios

When comparing the performance of the heuristics and RL-agents for both scenarios, the upper charts of Figure 5.7 display the solutions from the previous figures next to each other. Associated solutions in both scenarios are coloured in the same way. The performance of the heuristics is shown in Figure A9.1 in Appendix A9. The heuristics vary largely comparing both scenarios. Not only the location in the scatter plot changes but also the ranking of the heuristics as, for instance, VALID and FIFO are apart in the first scenario, whereas in the second scenario they are close to each other. This is because the rule-based heuristics do not necessarily take the system characteristics and its bottleneck into account. The relative poor performance of NJF in Scenario 2 compared to its Pareto-optimality in Scenario 1 supports this presumption, too.

In contrast, Agents 28 and 29 are almost robust when comparing both scenarios (see Figure 5.7). Herein, robustness is understood in terms of the location relative to the other solutions in the two-dimensional objective space. If a solution is in the same area of the objective space, although the scenario is changing, this solution is regarded as robust. Additionally, they show an overall superior performance when looking at both performance indicators and comparing it to the heuristics as well as other RL-agents. However, it has to be stated that, in particular, the results for the heuristics are just limited to four examples and, hence, it cannot be generalized and further investigations are required.

The bottom chart of Figure 5.7 goes one step further in analysing the location and spread of RL-agents in the objective space. It separates the entire space that is spanned by all solutions found into four quadrants. The quadrants are bounded based on the minimum, maximum, and midway between the minimum and maximum value of the respective solution space. The quadrant definition has a significant influence on the depicted results, but it is assumed that the computed solutions span a feasible and reasonable solution space. For each quadrant, Figure 5.7 shows the probability that a solution, i.e. an RL-agent, is in exactly that quadrant in Scenario 1's and Scenario 2's solution space. For instance, 88.9% of all RL-agents in the first quadrant, i.e. the bottom left quadrant, are in that quadrant in Scenario 1 and 2. Again, the results for the benchmark heuristics are displayed in Figure A9.1 of Appendix A9. The fact that the overall mean quadrant-similarity of all RL-agent is 81.9% and above 75.0% for the heuristics is an additional hint that supports the above-mentioned conclusion that the performance of heuristics varies more for different scenarios. However, note that these results are just an approximation as just four heuristics are analysed.

To sum it up, although the two scenarios are substantially different, the performance of a specifically designed RL-agent is robust with respect to production system characteristics. In other words, an RL-agent that shows a "good" performance in one scenario is also most likely appropriate in another. This is due to the fact that reinforcement learning learns by itself and the interaction with the actual production environment *how* to achieve a reward maximization and is not forced to follow a predefined rule that is based and limited to certain assumptions concerning the system characteristics.

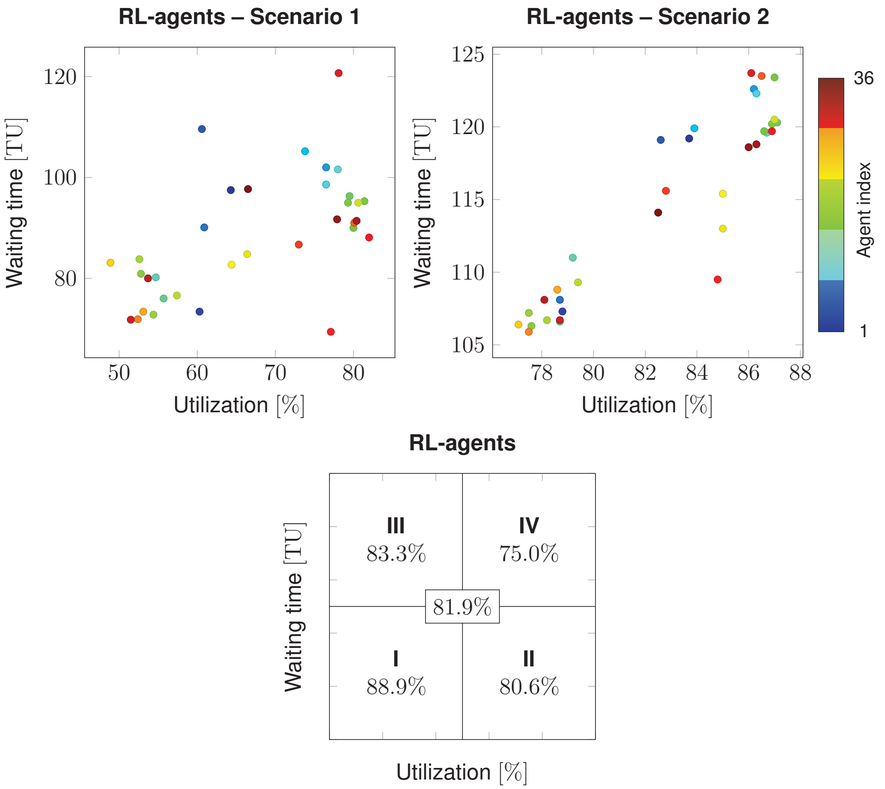


Figure 5.7: Comparison of RL-agents in Scenario 1 and 2 (top). Colouring indicates the solution index to capture the association between one solution in both scenarios. The four-quadrant-plot (bottom) depicts the probability that a solution that is in a specific quadrant in the solution space of Scenario 1 is also in that same quadrant in Scenario 2. The limits of the quadrants are defined by the minimum and maximum value of the respective solution space.

5.4.2 Investigation of changing scenarios

So far, the two scenarios are investigated separately, i.e. the RL-agent or heuristic is just applied in one scenario. As a conclusion, the elementary transferability of the RL-agent's design choice could be demonstrated. In this sub-section, two heuristics and one RL-agent first operate in a production system that is parametrized according to Scenario 1, and after ten million iterations it is changed to the second scenario. Figure 5.8 depicts the course of the moving mean of the machine utilization and the average order waiting time for FIFO, NJF, and Agent 12. Agent 12 is chosen due to its efficiency in both scenarios, i.e. low computational effort but still good performance. As before, the reported figures are mean values based on three repetition runs.

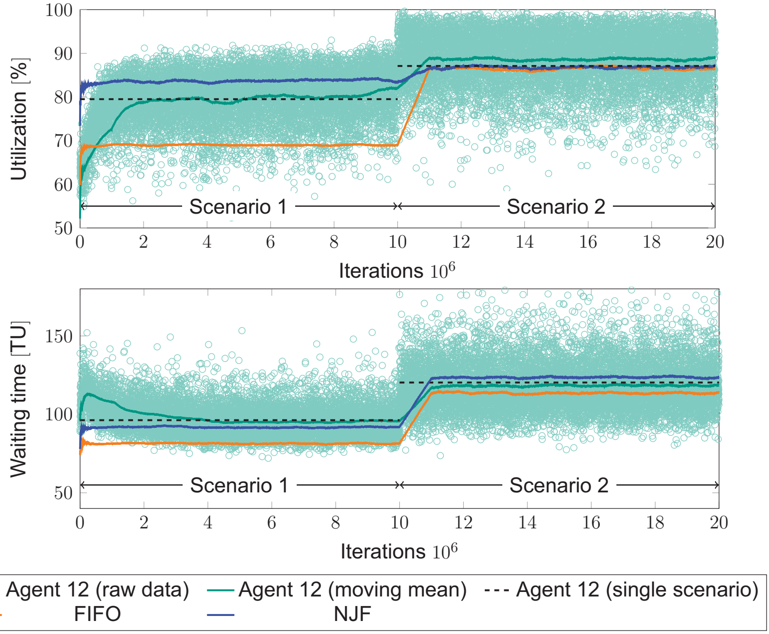


Figure 5.8: Average machine utilization and order waiting time when changing from Scenario 1 to Scenario 2 after ten million iterations. The dashed line depicts the RL-agent's performance in both scenarios separately (see Table 5.9 and Table A9.4). The transparent green scatter plot shows the KPI's raw data.

The RL-agent needs its training period until a stable performance level is reached in the first ten million iterations. When the scenarios change, both heuristics and the RL-agent adjust their performance level according to the characteristics of Scenario 2. Although this is expected for the rule-based heuristics, since they operate based on a deterministic procedure, it is not obvious for the RL-approach. However, the RL-agent adjusts its performance similarly fast. No second training phase can be seen in the performance measures, looking at the mean performance line plots and the transparent raw data scatter plots of Figure 5.8. Moreover, the agent achieves almost identical performance values as for a separated training in either Scenario 1 or Scenario 2. The utilization and waiting time are even slightly better after the change in the second scenario, which might be due to the extended learning period and additional training data. Therefore, it is concluded that an RL-agent adapts to changing production conditions and no significant training phase is required.

Another advantage of the RL-agent designs presented in Section 4.4 and analysed here is their “technical” transferability to similar use cases as presented by the two scenarios. This means that the state and action representations are generic enough that the agent design can be applied to both scenarios. Moreover, the agent can re-use the learned knowledge, i.e. already adjusted ANN-weights (also referring to the research on *transfer learning*). For instance, the state element S_{BEN} gives the relative inbound buffer fill level as a relative measure, what is applicable and has the same interpretation in both scenarios. Hence, the ANN does not need to be extended by additional input or output neurons. An adjustment is only required if, for instance, the number of machines is changed, because then the input and output layer of the ANN are changed, which modifies the structure of the network.

5.5 Dispatching policy plausibility analysis

The final investigations address the last research question on the plausibility of RL-agent’s dispatching policy in order to better understand the underlying reasoning and, thereby, increase the confidence and trust in the black-box algorithm. It, first, deals with the validity check, whether RL-agents can imitate heuristic behaviour (see Section 5.5.1). Extending this analysis, Section 5.5.2 focuses on action-level differences compared to heuristic decision-making.

5.5.1 Imitation of heuristic behaviour

The validity check and imitation analysis aim to examine whether the RL-agent is able to learn deterministic heuristic decision-making procedures. By passing heuristic-relevant system state information and a constant reward for the selection of the action, which the respective heuristic would take, it is intended that the RL-agent captures the rule-based heuristic policy. The herein presented results are based on Scenario 1 and 2, but just show one heuristic example. However, it is likely to expect similar results for further heuristics as the described state and reward design follows the same mechanism for any heuristic and, therefore, the learning task is similar.

In the following, the NJF* heuristic with a slightly adjusted rule-setting than the NJF is selected as reference heuristic. NJF* is transformed in such a way that still the nearest available order is selected, however, the destination is the predefined next order destination without considering machine groups. In this way a simple decision rule is given that just relies on a single rule and state element.

The RL-agent receives information on all possible actions S_{VA} as well as the action time state entries S_{AT}, indicating the distance to the next job. These are equivalent to the decision-relevant information of NJF*. The reward r_{NJF^*} is based on whether the selected action corresponds to the NJF*-action a_{NJF^*}:

$$r_{NJF^*}(\bullet) := \begin{cases} 1 & a = a_{NJF^*} \\ 0 & \text{else} \end{cases} \quad \forall a \in \mathcal{A}, s \in \mathcal{S} \qquad 5.3$$

The results for the moving mean reward per episode over the training phase are shown in Appendix A10 in Figure A10.1 and the mean performance indicators are summarized in Table 5.14. Two RL-agents are presented. For Agent 37, the discount rate γ is set to 0.1 and, for Agent 38, it is the default value of 0.9. In particular, Figure A10.1 reveals that the smaller γ-value is slightly beneficial for the learning speed and final reward level. Although the differences are just small, the explanation goes as follows: in this experiment, the reward is just related to the current state and action selection and, hence, no time delay needs to be considered. Since the delay is usually controlled by the γ-parameter, and a large value extends the period, just a small γ-value is adequate for the imitation of a heuristic rule.

Table 5.14: Comparison of RL-agents imitating the NJF heuristic with the NJF* performance.*

Agent	**γ-value**	**Scenario 1**			**Scenario 2**		
		Reward	$U[\%]$	$WT[\text{TU}]$	Reward	$U[\%]$	$WT[\text{TU}]$
37	0.1	0.948	81.9	92.0	0.941	84.7	123.5
38	0.9	0.943	81.9	93.0	0.910	85.2	124.0
NJF*		-	81.7	92.2	-	84.7	123.9

Given the above-defined reward function, the mean reward value directly translates into the probability of selecting the same action as NJF*. So, according to Table 5.14, the similarity is close to 95%. Moreover, the performance measures utilization and waiting time confirm a high conformity and imitation capability of the RL-agent, as the average performance values are nearly identical. Hence, the RL-agent is able to approximate policies that are similar to rule-based heuristics. This fact increases the confidence in the learned RL-policy and presumes that an RL-policy can reach a near-deterministic level.

5.5.2 Decision-making comparison on an action-level

After having seen that the agent learns rule-based dispatching policies and achieves similar results, it is crucial to understand the decision-making behaviour of RL-agents in more detail. One approach is to compare RL-agents and heuristics on an action-level. In the following, an RL-agent is compared to FIFO, NJF, and EMPTY.

The averaged results from three repetitions are reported in Figure 5.9. It shows a Venn diagram of the average probabilities that actions similar to heuristics are chosen. All possible cases are reported, i.e. in particular the overlapping areas when two or all three heuristics select the same action are separated. Again, the first scenario is more constrained and, therefore, the probabilities are smaller and overlapping actions are less likely.

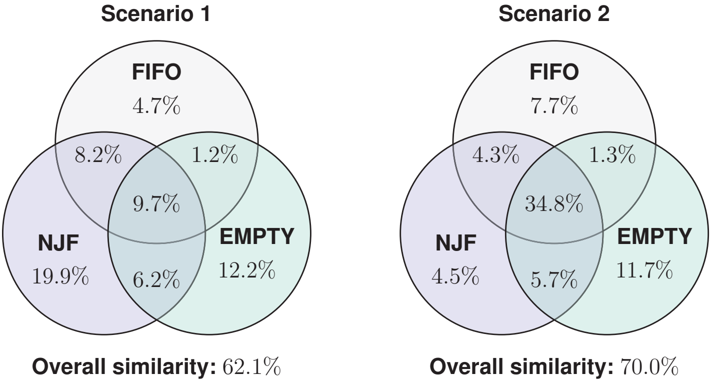


Figure 5.9: Venn diagrams for the average probability that the same actions are selected comparing three heuristics with reference Agent 12 based on the last 500,000 actions.

The overall probability of selecting the same action as one of the three heuristics is well above 50% in both scenarios. However, a clear picture on which rule is preferred cannot be derived. According to Figure 5.5, the performance of Agent 12 is nearby the NJF heuristic in Scenario 1 and the overlapping probability for NJF is the highest, too. The same applies for Scenario 2, in which the RL-agent is closer to EMPTY and, therefore, the probability is also the highest for this heuristic.

However, taking these interpretations further, several obstacles need to be considered (May 2019): First, the number of possible actions is influencing the results to a reasonable extent and could lead to an overestimate of the presented percentage values. For example, when there is only one valid action, then all heuristics and the RL-agent are similar. Thus, Figure A10.2 in Appendix A10 shows a frequency histogram of the number of possible actions at every decision point. As the mean value is 7.4 actions per iteration, the risk of overestimation does not seem to apply here, but still in above 10% of all occasions just a single action is feasible. Second, the comparison is limited as the actual heuristic dispatching decisions are not executed, meaning that it could happen that for several iterations, a heuristic selects the same action and, thus, biasing the data. This risk cannot be removed entirely.

All in all, the results present first insights into the behavioural investigation of RL-based dispatching policies. However, an in-depth analysis of the internal RL-processes, i.e. mainly the ANN (e.g. *saliency maps*), as well as a path-dependence analysis of multiple actions would lead most likely to more insights. First attempts in this direction are found in the work of (A_May 2019). Moreover, Section 6.2 provides an outlook into this direction.

5.6 Summary of evaluation and computational results

This chapter evaluates the method developed in Chapter 4 for an RL-based adaptive order dispatching in a complex job shop. The results are based on a discrete-event simulation of a semiconductor frontend manufacturer.

The research questions as displayed in Figure 5.1 are covered and discussed in the next chapter. Hereinafter, the key insights from the computational experiments are summarized:

1. Derived from a complex real-world job shop, a simulation environment is implemented that can be used as accurate training environment (digital twin). Moreover, not just a single validation scenario is designed to also evaluate the transferability of the results.

2. The RL-based system is able to make integrated dispatching decisions in real-time.

3. Comparing the performance of RL-agents to benchmark heuristics reveals that, first, even simple agent configurations present reasonable results and, second, more complex and sophisticated agents, taking domain expertise into account, outperform benchmark heuristics.

4. The ANN structure, learning rate, discount rate, and episode design are hyper-parameters and modelling choices that have a less significant influence on the results. Here default values are recommended.

5. The presented reinforcement learning algorithm is able to autonomously learn the following:

   a) If not all actions are feasible in every iteration, the agent is able to reliably distinguish the validity of actions.

   b) Rewarding action subsets separately has a significant effect and reveals the differences between push and pull production.

   c) The performance indicator variances of heuristics and RL-agents are comparable. The most constant inventory level was found for an RL-agent.

   d) Sparse reward functions are less efficient in terms of training speed and required amount of training data. Hence, an additional modelled reward is recommended in production control applications.

   e) Extending the state information improves the results, whenever the information is related to the reward and decision-making. Generally speaking, more state information does not always yield an improvement.

   f) An action representation that is not condensed to just the relevant actions increases the training effort, too. Though, adding extra actions, such as idling, can have a profound effect on the learned control strategy.

   g) Finally, the dispatching policy of an RL-agent can be close to a deterministic rule. Further analyses on the plausibility reveal that the learned strategy shows an above 50%-similarity with benchmark heuristics.

6. In a multi-objective solution space, reinforcement learning covers the solution space more extensively, e.g. by using weighted reward functions or a reward that distinguishes different action subsets. All in all, new Pareto-optimal RL-solutions can be identified.

7. RL-agents show a more robust performance with respect to changing production system characteristics and modifying the scenario setup within the learning process does no harm. Hence, the results show the ability to transfer a learned policy.

6 Discussion and outlook

This work presented an approach for an adaptive order dispatching that is based on reinforcement learning. The approach was successfully applied in a job shop setting that is derived from a semiconductor manufacturing application. The entire approach and computational results are discussed in Section 6.1 along the questions stated in Section 1.3 and the research deficit from Section 3.2. Moreover, the simulation-based training is examined separately. Next, Section 6.2 describes directions for forthcoming research.

6.1 Discussion

The research goal of this work was to increase the applicability of reinforcement learning in PPC by the design of an adaptive order dispatching system that is comprehensibly applicable to a broad range of complex production environments. To evaluate this goal, the derived and leading research questions are discussed critically in the following.

RL-applicability: *Is it possible to obtain an adaptive order dispatching system autonomously based on real-time operational data?*

The adaptive order dispatching system builds on a modular architecture to fully address the first research question. First, a production simulation framework is chosen that is able to represent a broad range of job shop-like systems. In particular, the necessity for agile and flexible production systems propagates job shops as they meet changing market requirements more efficiently. Logistic operations are similarly important, which are neglected in most state-of-the-art research work. Moreover, stochastic processes are enclosed to correspond to the dynamic nature of production processes. All in all, the simulation framework acts as valid and accurate *digital twin*, which allows to run the data-driven training in parallel to real-world operations and, thereby, significantly reduce the training time. However, the presented framework does not thoroughly cover all relevant production system characteristics and PPC tasks, such as setup processes, changing order release policies, or maintenance management. So, further extensions would be required to broaden the scope.

Second, the order dispatching decision framework integrates twofold decisions, namely: the transport decision as well as the next processing machine decision. The developed framework is based on the concept of an MDP, which adequately covers sequential decision-making problems. It is also not merely compatible with RL-algorithms but rule-based heuristics, too. Moreover, an MDP fulfils the requirement to derive dispatching decisions adaptively as every action is based on the current state. Finally, the execution of a trained RL-algorithm is real-time-compatible.

The integration of the production and decision framework gives a system that is able to obtain dispatching policies that are only based on data and, hence, limits the manual effort a practitioner has to spend. The feedback of the learning algorithm is directly taken from the production framework, what allows a dynamic performance evaluation and demonstrates the successful application of a discrete-event simulation to train RL-agents autonomously. Eventually, an extensive list of experiments showed that RL-agents are competitive and, when appropriately designed, even superior to benchmark heuristics. However, the presentation of different RL-modelling approaches also indicated that the application of reinforcement learning is still not conceivable when entirely foregoing domain knowledge. Simple RL-agents show a performance that is not altogether out of range, but surpassing benchmark heuristics requires intensive studies of a wide range of RL-agents.

Multi-objective optimization: *Do the modelling design choices of reinforcement learning extensively cover a multi-objective solution space, comparing to heuristic benchmarks?*

The performance is evaluated based on two objectives, i.e. order- and resource-related measures. In reinforcement learning, the objectives are translated into the reward function. In contrast to mathematical optimization, where a well-defined objective function rigidly determines the objective space, the reward mechanism in RL-algorithms is less straightforward. It is conditional to the definition of state, action, episode, and hyper-parameters, such as the discount rate. For example for the state representation, it could be concluded that state features have to correlate to the reward to have a positive effect. In conclusion, Figure 5.5 and Figure 5.6 demonstrated the extensive coverage of the solution space for a broad range of RL-agents. This is, in particular, relevant for production managers in practice, as they are able to select the most appropriate agent configuration.

However, the results affirm that modelling features that are beneficial in other domains, such as using a sparse reward, are not vividly transferable to the domain of production control. The computational effort does not outweigh the benefits with respect to a comfortable and intuitive reward design. Moreover, a comparison of RL-agents that are rewarded for machine utilization r_{util} with the waiting time reward r_{wt} showed heterogeneous results in such a sense that optimizing the waiting time seems to be harder. This might be due to several reasons. One could be that multiple reasons cause an order to idle, e.g. after order release, before processing, or after processing, which are hard to capture and relate to the reward. Finally, the tree-based reward provided reasonable results, but one would have expected a more significant improvement.

The investigation of rule-based heuristics was limited to four heuristics. An extensive study of further heuristics was not conducted. Nevertheless, additional heuristics can, in principle, be designed to cover the solution space more extensively, too. Still, the advantages of reinforcement learning are the multitude of modelling choices and that every modelling choice is just one part of the entire RL-mechanism. For instance, the weighted reward function demonstrated that the performance in the two-dimensional objective space can be projected (see Figure A9.2). In contrast, adjusting heuristic decision rules requires an in-depth investigation and knowledge about the production system characteristics.

Transferability: *Does the learning-based approach show a superior performance in terms of changing system characteristics in comparison to existing benchmarks?*

A superior performance for single production scenarios was already concluded before. This question was evaluated based on two scenarios that depict two extreme manifestations of the considered real-world use case. Apparently, two scenarios are not enough to profoundly state general conclusions. Still, the reported results at least support the hypothesis that the learning-based approach is able to handle changing production system characteristics. By varying both scenarios in a single experiment and comparing both scenarios separately, the RL-agent showed an adaptivity to adjust its dispatching policy according to new system characteristics, without any performance compromises. In contrast, the heuristics are not able to modify their decision rules and so their performance cannot be projected.

Furthermore, this work exhibited two ways to react to changing system conditions: on the one hand, an RL-agent balances minor variations via its training ability and, on the other hand, a new re-training can be easily initiated if the system characteristics change significantly via a modified simulation setup.

User acceptance: *Is the acceptance of practitioners supported by a plausibility analysis of the RL-based control policy?*

The RL-algorithm is, by itself, a black-box approach that does not allow a complete understanding of its internal procedures. This fact is a prevailing drawback of any advanced Deep Learning algorithm and causes reluctance when applied in reality (Goodfellow & Bengio et al. 2016). The present work made a first step into the direction of evaluating RL-policies by an analysis of the policy on an action-level and a comparison with well-understood heuristics.

Although having obtained initial results, still more research is required to extend the knowledge in this area and to capture in which ways the RL-policy differs from existing dispatching approaches. It is, in particular, of interest whether the learning algorithm reaches the state of reliably self-optimizing production systems. If understood entirely, the insights taken from RL-policies could also be translated back into the design of rule-based production control systems or even the design of advanced manufacturing systems.

Discussion of the application setup In addition to the research questions, the dispatching application use case is discussed, once more, and compared to state-of-the-art scheduling research in order to provide an additional note. In the present case, transportation requires a significant amount of time. Hence, it is worth considering the dispatching route. As the literature review showed, not many authors consider transports explicitly. Moreover, dispatching rarely integrates the order selection and next processing decision. So, a rough classification of the herein presented use case with respect to established scheduling approaches goes as follows: transport times differ for orders as they are placed at different locations and processed on specific machines. Hence, from an order point of view, the time before the actual operation starts shows parallels to machine setup times, which are considered in most scheduling approaches. Hence, although dealing with separate problems, analogies can be found between the considered dispatching and the broad scheduling literature.

Discussion of the simulation-based approach Before looking at future research directions, concluding remarks on the simulated training environment are outlined. The accuracy of the simulation highly affects the applicability of the presented approach as well as the transferability of the results towards a real-world application. Herein, two risks remain unsolved. First, the risk of deploying a simulation-trained RL-agent and then situations occur, i.e. states, that never happened in the simulated environment before. The second risk is that the real dynamics, e.g. stochastic processes, do not match with the simulation assumptions. Both lead to an unknown RL-behaviour. However, the first risk seems less likely, as the investigation of changing scenarios showed a robust performance. Moreover, the state is based on normalized values such as waiting times or buffer fill levels that vary within a realistic range in the simulation, too. In contrast, the second risk can only be faced and assessed in a real-world RL-application. Eventually, both risks are significantly reduced if the input data that describes the production system is accurate, which is supported by the industrial digitization. So, more accurate digital models of production systems are likely in the near future. Furthermore, this issue is already well-addressed in recent RL-research activities. In fact, most recent breakthrough applications rely on simulated environments and, still, demonstrate a successful deployment, such as the board game Go (Silver & Schrittwieser et al. 2017), the computer game StarCraft[1], or the robotic hand solving Rubik's cube[2]. Finally, one might consider seeing it the other way round: autonomous RL-agents may further enhance digital twins as they allow the representation and integration of human-like behaviour.

[1] Vinyals et al. (2019), *AlphaStar: Mastering the Real-Time Strategy Game StarCraft II* `https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii` (accessed on 17.06.2020).

[2] Akkaya et al. (2019), *Solving Rubik's Cube with a Robot Hand* `https://openai.com/blog/solving-rubiks-cube` (accessed on 17.06.2020)

6.2 Outlook and further considerations

The present work contributes to the research on the application of reinforcement learning to the domain of production planning and control. It brings the autonomous control of production systems one step closer to reality. Nevertheless, in the course of this work, further avenues for future research are identified.

First, the presented results barely touched on the investigation of state, action, and reward design. In particular, the question of how these elements are related to each other is interesting. If the relationships are better understood specifically for the application domain, even more advanced RL-agents could be designed. The question of which state information is necessary to optimize one particular or multiple objectives or whether the encoding can be improved are two examples. For instance, Silver & Hubert et al. (2018) include, in addition to the current state, the latest states into the state vector. However, the state representation drives the data intensity, which is particularly relevant for practitioners, and, hence, a state vector that is as condensed as possible needs to be found. Finally, the trade-off between sparse and modelled reward is still not sufficiently answered. Although, modelled reward showed a superior performance, it needs to be handled with care, as it perhaps changes the underlying MDP and, so, the optimal solution for the new MDP might not match to the original MDP. In conclusion, reinforcement learning modelling is still a new research area with manifold unaddressed questions. Taking mathematical optimization as an example, there are a broad range of efficient modelling choices known and available, e.g. for modelling time constraints. As shown in this work, insights from supervised learning can be translated to the state representation modelling, as both steps are similar. Further synergies have to be identified in future research in order to develop efficient reinforcement learning models. Also, advanced hyper-parameter optimization techniques and design of experiment methods could be applied to find such models.

Second, in the domain of operations management, a broad range of quantitative optimization techniques are already in use and extensively studied (Domschke & Drexl et al. 2015). Hence, the algorithmic properties of reinforcement learning need to be evaluated in comparison to existing techniques. Although having provided an explanation of the RL-algorithm and shown computational results, just a rough outline of the algorithmic properties on a high level could be provided. For instance, Waschneck & Altenmüller et al. (2018a) demonstrate good results by using a value-based DQN-algorithm, whereas this work implements the policy-based TRPO-algorithm. Therefore, an investigation of the benefits and drawbacks of both types is mandatory. Furthermore, maybe even a combination of RL-algorithms and mathematical optimization techniques is a promising avenue, as proposed by Amos & Kolter (2017).

Third, there are already some approaches to unravel artificial neural networks. Morch & Kjems et al. (1995) present the idea of *saliency*, i.e. a map that quantifies for every input vector element its importance with respect to the output. Simonyan & Vedaldi et al. (2014) continue their research and are able to draw saliency maps for convolutional neural networks. These techniques help to identify the reasoning behind the decision-making RL-policy network, which in the end raises the acceptance. Moreover, a composite parametric rule could be derived from those insights, analogously to the approach of Verma & Murali et al. (2018), or a decision tree can be used to capture hierarchical decision-rules (Guidotti & Monreale et al. 2018).

Last but not least, this work presented just a single agent system and only two scenarios. In order to support the hypotheses with additional proofs, further scenarios and applications need to be investigated. There is a large variety of production system types (Wiendahl 1997). This work focused mainly from a bottleneck perspective on the system characteristics. It would be interesting to see how reinforcement learning performs for other systems. For instance, sequence planning in automotive assembly lines is a complex task, too. Additionally, maintenance management is promising as it is a data-intensive problem. Reinforcement learning can help to implement a closed-loop maintenance system by not just predicting upcoming machine breakdowns but directly deriving an appropriate action upon the prediction (Kuhnle & Jakubik et al. 2019). Furthermore, this work just focused on the order dispatching, leaving aside other control tasks summarized by Lödding (2016). For instance, the order release policy has a major influence on the inventory level and, hence, the system performance (see Section 2.1.2). So policies such as ConWIP (constant work in process) need to be investigated, too. A related research hypothesis would be to investigate if reinforcement learning is capable to learn WIP-controlling policies autonomously without external triggers, e.g. rewarding low inventory.

Eventually, multi-agent systems, such as introduced by Paternina-Arboleda & Das (2005), show additional mechanisms, e.g. communication and coordination, that need to be investigated when multiple agents are learning and operating simultaneously. State-of-the-art production systems propagate decentral decision agents for any operation on the shopfloor, which requires a holistic architecture to be capable to ensure a robust and global-optimal performance. For instance, Heger & Voss (2019) investigate the combination of sequencing, dispatching, and routing heuristics in a flexible job shop. This research needs to be extended towards the application of RL-agents. In particular in multi-agent systems, the need to evaluate the most appropriate action scope is a paramount research question. This work just focused on a rather low-level control decision and, hence, more high-level and long-term decisions, such as planning tasks, are not thoroughly addressed.

7 Conclusion

Striving for operational excellence in the competitive and ever faster changing environment of manufacturing companies challenges existing production control methods. With the upcoming age of digitization, data-driven machine learning algorithms offer an alternative approach to optimize operations and exploit the full potential of data sources in manufacturing systems.

The latest research findings show a superior performance of reinforcement learning in a broad range of application domains, even outperforming human experts without using any human knowledge to train algorithms. However, in the field of production control, just a limited number of authors have looked into the application of reinforcement learning so far. An extensive investigation of different RL-configurations was not yet performed.

Within various tasks of production planning and control, order dispatching is key to ensure flexibility and high performance in terms of high capacity utilization and short cycle times. Motivated by the operations in complex job shops, this work closed the research gap and focused on the application of reinforcement learning for an adaptive order dispatching. Incorporating actual system data allows capturing the system behaviour more accurately than approaches that are based on static procedures or mathematical optimization that is just applicable when too many modelling simplifications are made. Additionally, the manual effort can be reduced by relying on the inference capabilities of RL-algorithms and digital representations of production systems.

A comprehensive methodology is presented for the design and implementation of RL-agents as central dispatching decision-making unit. Known RL-modelling challenges concerning the state, action, reward, and hyper-parameters are addressed. Multiple design choices are analysed based on two real-world production scenarios taken from a semiconductor manufacturer. The results reveal that RL-agents perform adaptive policies successfully and adjust their policy to different objectives. Moreover, RL-agents can outperform existing rule-based benchmarks. Extending the state representation clearly improves the performance if there is a relation to the objectives. The reward can be designed to enable the integrated optimization of multiple objectives. Finally, specific RL-agent configurations reach not just a high performance in one scenario, but a robust performance for changing system characteristics.

Hence, the research shows a substantial contribution in the direction of self-optimizing and autonomous production systems. Operations managers need to evaluate the potential of the presented approach to stay competitive in terms of flexibility and, at the same time, keeping the manual effort of designing, operating, and monitoring operation production systems in a reasonable balance.

References

References according to the scheme (A_<last name> <year>) refer to Bachelor and Master Theses at the wbk Institute of Production Science, which were supervised by the author of this dissertation.

A_Behrendt 2019
Behrendt, S. (2019), „Baumbasierte Rewardberechnung eines Reinforcement Learning Agenten zur Produktionssteuerung in der Halbleiterindustrie“. Bachelor Thesis. Karlsruhe: Karlsruhe Institute of Technology, wbk Institute of Production Science.

A_Hettich 2019
Hettich, F. (2019), „Benchmarking von Reinforcement Learning zur Produktionssteuerung einer komplexen Werkstattfertigung mit Methoden des Operations Research“. Master Thesis. Karlsruhe: Karlsruhe Institute of Technology, wbk Institute of Production Science.

A_Jakubik 2018
Jakubik, J. (2018), „Opportunistische Entscheidungen über Instandhaltungsmaßnahmen für ein intelligentes Wartungsmanagement“. Bachelor Thesis. Karlsruhe: Karlsruhe Institute of Technology, wbk Institute of Production Science.

A_Kaiser 2019
Kaiser, J.-P. (2019), „Modellierung von Reinforcement-Learning Entscheidungsstrukturen zur Produktionssteuerung in der komplexen Werkstattfertigung der Halbleiterindustrie“. Master Thesis. Karlsruhe: Karlsruhe Institute of Technology, wbk Institute of Production Science.

A_May 2019
May, M. (2019), „Reinforcement Learning in Production Control - An Analysis of Decisions, Agent Behavior and the Influence of Demonstration Data“. Master Thesis. Karlsruhe: Karlsruhe Institute of Technology, wbk Institute of Production Science.

A_Schäfer 2018
Schäfer, L. (2018), „Reinforcement Learning Agentensysteme zur Produktionssteuerung in der Halbleiterindustrie“. Bachelor Thesis. Karlsruhe: Karlsruhe Institute of Technology, wbk Institute of Production Science.

A_Theiß 2019
Theiß, F. (2019), „Curiosity-driven Reinforcement Learning in der Produktionssteuerung“. Bachelor Thesis. Karlsruhe: Karlsruhe Institute of Technology, wbk Institute of Production Science.

Abele & Reinhart 2011
Abele, E. & Reinhart, G. (2011), *Zukunft der Produktion*. München: Carl Hanser Verlag.

Acker 2011

Acker, I. J. (2011), *Methoden zur mehrstufigen Ablaufplanung in der Halbleiterindustrie*. Wiesbaden: Gabler Verlag.

Adam 1996

Adam, D. (1996), *Planung und Entscheidung: Modelle - Ziele - Methoden*. Wiesbaden: Gabler Verlag.

Adams & Balas et al. 1988

Adams, J.; Balas, E. & Zawack, D. (1988), „The Shifting Bottleneck Procedure for Job Shop Scheduling", *Management Science* 34.3, pp. 391–401.

Aggteleky 1987

Aggteleky, B. (1987), *Fabrikplanung: Grundlagen - Zielplanung - Vorarbeiten, unternehmerische und systemtechnische Aspekte, Marketing und Fabrikplanung*. 2nd ed. München: Carl Hanser Verlag.

Alizadeh & Goldfarb 2003

Alizadeh, F. & Goldfarb, D. (2003), „Second-order cone programming", *Mathematical Programming* 95.1, pp. 3–51.

Alpaydin 2010

Alpaydin, E. (2010), *Introduction to machine learning*. Cambridge, MA: MIT Press.

Amos & Kolter 2017

Amos, B. & Kolter, J. Z. (2017), „OptNet: Differentiable Optimization as a Layer in Neural Networks", *arXiv preprint* 1703.00443, pp. 1–13.

Arnold & Isermann et al. 2008

Arnold, D.; Isermann, H.; Kuhn, A.; Tempelmeier, H. & Furmans, K. (2008), *Handbuch Logistik*. VDI-Buch. Berlin, Heidelberg: Springer.

Arora & Barak 2009

Arora, S. & Barak, B. (2009), *Computational complexity: a modern approach*. Cambridge: Cambridge University Press.

Arviv & Stern et al. 2016

Arviv, K.; Stern, H. & Edan, Y. (2016), „Collaborative reinforcement learning for a two-robot job transfer flow-shop scheduling problem", *International Journal of Production Research* 54.4, pp. 1196–1209.

Åström & Wittenmark 2008

Åström, K. & Wittenmark, B. (2008), *Adaptive control*. 2nd ed. Mineola: Dover Publications Inc.

Aurand & Miller 1997

Aurand, S. S. & Miller, P. J. (1997), „The operating curve: a method to measure and benchmark manufacturing line productivity". *1997 IEEE/SEMI Advanced Semiconductor*

Manufacturing Conference and Workshop ASMC 97 Proceedings, Cambridge, MA, 10.09. - 12.09.1997. Ed. by D. T. Fletcher; S. R. McClure & J. Goodman. Mountain View: IEEE, pp. 391–397.

Aydin & Öztemel 2000

Aydin, M. & Öztemel, E. (2000), „Dynamic job-shop scheduling using reinforcement learning agents“, *Robotics and Autonomous Systems* 33.2-3, pp. 169–178.

Baker 1984

Baker, K. R. (1984), „Sequencing Rules and Due-Date Assignments in a Job Shop“, *Management Science* 30.9, pp. 1093–1104.

Baptiste & Pape et al. 2001

Baptiste, P.; Pape, C. & Nuijten, W. (2001), *Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems*. New York: Springer US.

Bauernhansl & Hompel et al. 2014

Bauernhansl, T.; Hompel, M. ten & Vogel-Heuser, B. (2014), *Industrie 4.0 in Produktion, Automatisierung und Logistik*. Wiesbaden: Springer Fachmedien Wiesbaden.

Bauernhansl & Krüger et al. 2016

Bauernhansl, T.; Krüger, J.; Reinhart, G. & Schuh, G. (2016), *WGP-Standpunkt Industrie 4.0*. Darmstadt: Wissenschaftliche gesellschaft für produktionstechnik WGP e.V.

Bauernhansl & Schatz et al. 2014

Bauernhansl, T.; Schatz, A. & Jäger, J. (2014), „Komplexität bewirtschaften - Industrie 4.0 und die Folgen“, *ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb* 109.5, pp. 347–350.

Bechte 1980

Bechte, W. (1980), „Steuerung der Durchlaufzeit durch belastungsorientierte Auftragsfreigabe bei Werkstattfertigung“. Dissertation. Hannover: Universität Hannover.

Bellman 1966

Bellman, R. (1966), „Dynamic Programming“, *Science* 153.3731, pp. 34–37.

Benders 1962

Benders, J. F. (1962), „Partitioning procedures for solving mixed-variables programming problems“, *Numerische Mathematik* 4.1, pp. 238–252.

Bertsekas 2005

Bertsekas, D. P. (2005), *Dynamic programming and optimal control*. 3rd ed. Nashua: Athena Scientific.

Bertsimas & Tsitsiklis 1997

Bertsimas, D. & Tsitsiklis, J. N. (1997), *Introduction to linear optimization*. 6th ed. Belmont: Athena Scientific.

Blackstone & Phillips et al. 1982

Blackstone, J. H.; Phillips, D. T. & Hogg, G. L. (1982), „A state-of-the-art survey of dispatching rules for manufacturing job shop operations“, *International Journal of Production Research* 20.1, pp. 27–45.

Bochmann 2018

Bochmann, L. S. (2018), „Entwicklung und Bewertung eines flexiblen und dezentral gesteuerten Fertigungssystems für variantenreiche Produkte“. Dissertation. Zürich: Eidgenössische Technische Hochschule Zürich.

Boebel & Ruelle 1996

Boebel, F. & Ruelle, O. (1996), „Cycle time reduction program at ACL“. *IEEE/SEMI 1996 Advanced Semiconductor Manufacturing Conference and Workshop. Theme-Innovative Approaches to Growth in the Semiconductor Industry. ASMC 96 Proceedings, Cambridge, MA, 12.11. - 14.11.1996*. Ed. by M. M. Kindling. Piscataway: IEEE, pp. 165–168.

Brucker & Knust 2012

Brucker, P. & Knust, S. (2012), *Complex Scheduling*. 2nd ed. Berlin, Heidelberg: Springer.

Burke & Gendreau et al. 2013

Burke, E. K.; Gendreau, M.; Hyde, M.; Kendall, G.; Ochoa, G.; Özcan, E. & Qu, R. (2013), „Hyper-heuristics: a survey of the state of the art“, *Journal of the Operational Research Society* 64.12, pp. 1695–1724.

Busoniu & Babuska et al. 2008

Busoniu, L.; Babuska, R. & De Schutter, B. (2008), „A Comprehensive Survey of Multiagent Reinforcement Learning“, *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 38.2, pp. 156–172.

Caridi & Cavalieri 2004

Caridi, M. & Cavalieri, S. (2004), „Multi-agent systems in production planning and control: an overview“, *Production Planning & Control* 15.2, pp. 106–118.

Chapman & Clinton et al. 2000

Chapman, P.; Clinton, J.; Kerber, R.; Khabaza, T.; Reinartz, T.; Shearer, C. & Wirth, R. (2000), *CRISP-DM 1.0: Step-by-step data mining guide*. Chicago: SPSS Inc.

Chen & Xia et al. 2015

Chen, C.; Xia, B.; Zhou, B.-h. & Xi, L. (2015), „A reinforcement learning based approach for a multiple-load carrier scheduling problem“, *Journal of Intelligent Manufacturing* 26.6, pp. 1233–1245.

Chopra & Meindl 2019

Chopra, S. & Meindl, P. (2019), *Supply chain management : strategy, planning, and operation*. 7th ed. London: J. Pearson & Co.

Chui & Manyika et al. 2018

Chui, M.; Manyika, J.; Miremadi, M.; Henke, N.; Chung, R.; Nel, P. & Malhotra, S. (2018), *Notes from the AI frontier: Applications and value of deep learning*. London: McKinsey Global Institute.

Crites & Barto 1998

Crites, R. & Barto, A. (1998), „Elevator group control using multiple reinforcement learning agents“, *Machine Learning* 262, pp. 235–262.

Crites & Barto 1995

Crites, R. & Barto, A. (1995), „Improving Elevator Performance Using Reinforcement Learning“. *Proceedings of the 8th International Conference on Neural Information Processing Systems, Denver, 27.11. - 02.12.1995*. Ed. by D. Touretzky; M. Mozer & M. Hasselmo. Cambridge, MA: MIT Press, pp. 1017–1023.

Csáji & Monostori et al. 2006

Csáji, B. C.; Monostori, L. & Kádár, B. (2006), „Reinforcement learning in a distributed market-based production control system“, *Advanced Engineering Informatics* 20.3, pp. 279–288.

Das & Gosavi et al. 1999

Das, T. K.; Gosavi, A.; Mahadevan, S. & Marchalleck, N. (1999), „Solving Semi-Markov Decision Problems Using Average Reward Reinforcement Learning“, *Management Science* 45.4, pp. 560–574.

Dewey 2014

Dewey, D. (2014), „Reinforcement Learning and the Reward Engineering Principle“. *2014 AAAI Spring Symposium Series, Palo Alto, 24.03. - 26.03.2014*. Ed. by C. M. Hamilton. Palo Alto: AAAI Press, pp. 13–16.

Dietterich & Zhang 1995

Dietterich, T. G. & Zhang, W. (1995), „A Reinforcement Learning Approach to Job-shop Scheduling“. *Proceedings of the 14th international joint conference on Artificial intelligence, Montreal, 20.08. - 25.08.1995*. Ed. by C. S. Mellish. San Francisco: Morgan Kaufmann Publishers Inc., pp. 1114–1120.

Domschke & Drexl et al. 2015

Domschke, W.; Drexl, A.; Klein, R. & Scholl, A. (2015), *Einführung in Operations Research*. 9th ed. Berlin, Heidelberg: Springer.

Duffie 1996

Duffie, N. A. (1996), „Heterarchical control of highly distributed manufacturing systems“, *International Journal of Computer Integrated Manufacturing* 9.4, pp. 270–281.

Dunke & Nickel 2017

Dunke, F. & Nickel, S. (2017), „Evaluating the quality of online optimization algorithms by discrete event simulation“, *European Journal of Operations Research* 25.4, pp. 831–858.

Dunke & Nickel 2019

Dunke, F. & Nickel, S. (2019), „Online optimization with gradual look-ahead“, *Operational Research* , in press, pp. 1–35.

Echsler Minguillon 2020

Echsler Minguillon, F. (2020), „Prädiktiv-reaktive Produktionssteuerung zur Steigerung der Robustheit Anwendung in agilen Produktionssystemen“. Dissertation. Karlsruhe: Karslruhe Institute of Technology.

ElMaraghy & Azab et al. 2009

ElMaraghy, H.; Azab, A.; Schuh, G. & Pulz, C. (2009), „Managing variations in products, processes and manufacturing systems“, *CIRP Annals* 58.1, pp. 441–446.

ElMaraghy & ElMaraghy et al. 2012

ElMaraghy, W.; ElMaraghy, H.; Tomiyama, T. & Monostori, L. (2012), „Complexity in engineering design and manufacturing“, *CIRP Annals* 61.2, pp. 793–814.

Ertel 2017

Ertel, W. (2017), *Introduction to Artificial Intelligence*. 2nd ed. Undergraduate Topics in Computer Science. Cham: Springer International Publishing.

Eversheim 2002

Eversheim, W. (2002), *Organisation in der Produktionstechnik - Arbeitsvorbereitung*. 4th ed. Berlin, Heidelberg: Springer.

Eversheim & Schuh 1996

Eversheim, W. & Schuh, G. (1996), *Produktion und Management »Betriebshütte«*. 7th ed. Berlin Heidelberg: Springer.

Fazlollahtabar & Saidi-Mehrabad 2013

Fazlollahtabar, H. & Saidi-Mehrabad, M. (2013), „Methodologies to Optimize Automated Guided Vehicle Scheduling and Routing Problems: A Review Study“, *Journal of Intelligent and Robotic Systems: Theory and Applications* 77.3-4, pp. 525–545.

Feo & Resende 1995

Feo, T. A. & Resende, M. G. C. (1995), „Greedy Randomized Adaptive Search Procedures“, *Journal of Global Optimization* 6.2, pp. 109–133.

Fowler & Robinson 1995

Fowler, J. & Robinson, J. (1995), *Measurement and Improvement of Manufacturing Capacity (MIMAC): Final Report*. Austin: SEMATECH Technology Transfer.

Freitag & Hildebrandt 2016

Freitag, M. & Hildebrandt, T. (2016), „Automatic design of scheduling rules for complex manufacturing systems by multi-objective simulation-based optimization“, *CIRP Annals - Manufacturing Technology* 65.1, pp. 433–436.

Freitag 2005

Freitag, M. (2005), „Modellierung und Analyse von Produktionssystemen mit Methoden der nichtlinearen Dynamik“. Dissertation. Bremen: University of Bremen.

Gabel 2009

Gabel, T. (2009), „Multi-Agent Reinforcement Learning Approaches for Distributed Job-Shop Scheduling Problems“. Dissertation. Osnabrück: University of Osnabrück.

Gabel & Riedmiller 2008

Gabel, T. & Riedmiller, M. (2008), „Adaptive Reactive Job-Shop Scheduling With Reinforcement Learning Agents“, *International Journal of Information Technology and Intelligent Computing* 24.4, pp. 14–18.

Gabel & Riedmiller 2012

Gabel, T. & Riedmiller, M. (2012), „Distributed policy search reinforcement learning for job-shop scheduling tasks“, *International Journal of Production Research* 50.1, pp. 41–61.

García & Luengo et al. 2015

García, S.; Luengo, J. & Herrera, F. (2015), *Data Preprocessing in Data Mining*. Intelligent Systems Reference Library. Cham: Springer International Publishing.

Garey & Johnson et al. 1976

Garey, M. R.; Johnson, D. S. & Sethi, R. (1976), „The Complexity of Flowshop and Jobshop Scheduling“, *Mathematics of Operations Research* 1.2, pp. 117–129.

Garey & Johnson 1979

Garey, M. R. & Johnson, D. S. (1979), *Computers and intractability: a guide to the theory of NP-completeness*. New York: W.H. Freeman & Company Publisher.

Geiger & Uzsoy et al. 2006

Geiger, C. D.; Uzsoy, R. & Aytuğ, H. (2006), „Rapid Modeling and Discovery of Priority Dispatching Rules: An Autonomous Learning Approach“, *Journal of Scheduling* 9.1, pp. 7–34.

Gendreau & Potvin 2005

Gendreau, M. & Potvin, J.-Y. (2005), „Metaheuristics in Combinatorial Optimization“, *Annals of Operations Research* 140.1, pp. 189–213.

Glover & Laguna 1998

Glover, F. & Laguna, M. (1998), „Tabu Search“. *Handbook of Combinatorial Optimization*. Ed. by D.-Z. Du & P. Pardalos. Boston, MA: Springer US, pp. 2093–2229.

Glover & Kochenberger 2003

Glover, F. & Kochenberger, G. (2003), *Handbook of metaheuristics*. 1st ed. Boston: Springer.

Goldberg & Holland 1988

Goldberg, D. E. & Holland, J. H. (1988), „Genetic Algorithms and Machine Learning“, *Machine Learning* 3.2, pp. 95–99.

Gomory 1958

Gomory, R. E. (1958), „Outline of an algorithm for integer solutions to linear programs“, *Bulletin of the American Mathematical Society* 64.5, pp. 275–279.

Goodfellow & Bengio et al. 2016

Goodfellow, I.; Bengio, Y. & Courville, A. (2016), *Deep Learning*. 1st ed. Cambridge, MA: MIT Press.

Graham & Lawler et al. 1979

Graham, R. L.; Lawler, E. L.; Lenstra, J. K. & Kan, A. H. (1979), „Optimization and approximation in deterministic sequencing and scheduling: A survey“, *Annals of Discrete Mathematics* 5.C, pp. 287–326.

Greschke & Schönemann et al. 2014

Greschke, P.; Schönemann, M.; Thiede, S. & Herrmann, C. (2014), „Matrix structures for high volumes and flexibility in production systems“, *Procedia CIRP* 17, pp. 160–165.

Greschke 2016

Greschke, P. I. (2016), „Matrix-Produktion als Konzept einer taktunabhängigen Fließfertigung“. Dissertation. Essen: Technische Universität Carolo-Wilhelmina zu Braunschweig.

Grondman & Busoniu et al. 2012

Grondman, I.; Busoniu, L.; Lopes, G. A. D. & Babuska, R. (2012), „A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients“, *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6, pp. 1291–1307.

Gudehus 2010

Gudehus, T. (2010), *Logistik - Grundlagen, Strategien, Anwendungen*. 4th ed. Berlin, Heidelberg: Springer.

Guidotti & Monreale et al. 2018

Guidotti, R.; Monreale, A.; Ruggieri, S.; Turini, F.; Giannotti, F. & Pedreschi, D. (2018), „A survey of methods for explaining black box models“, *ACM Computing Surveys* 51.5.

Günther & Tempelmeier 2012

Günther, H.-O. & Tempelmeier, H. (2012), *Produktion und Logistik*. 9th ed. Berlin, Heidelberg: Springer.

Gupta & Egorov et al. 2017

Gupta, J. K.; Egorov, M. & Kochenderfer, M. (2017), „Cooperative Multi-agent Control Using Deep Reinforcement Learning“. *Autonomous Agents and Multiagent Systems*. Ed. by G. Sukthankar & J. A. Rodriguez-Aguilar. 10642nd ed. Cham: Springer International Publishing, pp. 66–83.

Gutenberg 1951

Gutenberg, E. (1951), *Grundlagen der Betriebswirtschaftslehre*. 1st ed. Berlin, Heidelberg: Springer.

Gutenschwager & Rabe et al. 2017

Gutenschwager, K.; Rabe, M.; Spieckermann, S. & Wenzel, S. (2017), *Simulation in Produktion und Logistik*. 1st ed. Berlin, Heidelberg: Springer.

Hackstein 1984

Hackstein, R. (1984), *Produktionsplanung und -steuerung (PPS): Ein Handbuch für die Betriebspraxis*. Düsseldorf: VDI-Verlag.

Hadfield-Menell & Milli et al. 2017

Hadfield-Menell, D.; Milli, S.; Abbeel, P.; Russell, S. J. & Dragan, A. (2017), „Inverse Reward Design". *Advances in Neural Information Processing Systems 30 (NIPS 2017), Long Beach, 04.12. - 09.12.2017*. Ed. by I. Guyon; U. V. Luxburg; S. Bengio; H. Wallach; R. Fergus; S. Vishwanathan & R. Garnett. Red Hook: Curran Associates Inc., pp. 6765–6774.

Hafner & Riedmiller 2011

Hafner, R. & Riedmiller, M. (2011), „Reinforcement learning in feedback control", *Machine Learning* 84.1-2, pp. 137–169.

Hansen & Mladenović 2001

Hansen, P. & Mladenović, N. (2001), „Variable neighborhood search: Principles and applications", *European Journal of Operational Research* 130.3, pp. 449–467.

Haupt 1989

Haupt, R. (1989), „A survey of priority rule-based scheduling", *OR Spektrum* 11.1, pp. 3–16.

Hazan 2016

Hazan, E. (2016), „Introduction to Online Convex Optimization", *Foundations and Trends in Optimization* 2.3-4, pp. 157–325.

Heger 2014

Heger, J. (2014), „Dynamische Regelselektion in der Reihenfolgeplanung". Dissertation. Bremen: Universität Bremen.

Heger & Branke et al. 2016

Heger, J.; Branke, J.; Hildebrandt, T. & Scholz-Reiter, B. (2016), „Dynamic adjustment of dispatching rule parameters in flow shops with sequence-dependent set-up times", *International Journal of Production Research* 54.22, pp. 6812–6824.

Heger & Voss 2019

Heger, J. & Voss, T. (2019), „Reducing mean tardiness in a flexible job shop containing AGVs with optimized combinations of sequencing and routing rules", *Procedia CIRP* 81, pp. 1136–1141.

Henke & Bughin et al. 2016

Henke, N.; Bughin, J.; Chui, M.; Manyika, J.; Saleh, T.; Wiseman, B. & Sethupathy, G. (2016), *The age of analytics: Competing in a data-driven world*. London: McKinsey Global Institute.

Hilsenbeck 2005

Hilsenbeck, K. (2005), „Optimierungsmodelle in der Halbleiterproduktionstechnik“. Dissertation. München: Technischen Universität München.

Hofstedt & Wolf 2007

Hofstedt, P. & Wolf, A. (2007), *Einführung in die Constraint-Programmierung*. 1st ed. Berlin, Heidelberg: Springer.

Horn 2008

Horn, S. (2008), „Simulationsgestützte Optimierung von Fertigungsabläufen in der Produktion elektronischer Halbleiterspeicher“. Dissertation. Dresden: University of Dresden.

Jeong & Randhawa 2001

Jeong, B. H. & Randhawa, S. U. (2001), „A multi-attribute dispatching rule for automated guided vehicle systems“, *International Journal of Production Research* 39.13, pp. 2817–2832.

Kacem & Hammadi et al. 2002

Kacem, I.; Hammadi, S. & Borne, P. (2002), „Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic“, *Mathematics and Computers in Simulation* 60.3-5, pp. 245–276.

Kaelbling & Littman et al. 1996

Kaelbling, L. P.; Littman, M. L. & Moore, A. W. (1996), „Reinforcement Learning: A Survey“, *Journal of Artificial Intelligence Research* 4, pp. 237–285.

Kagermann & Gaus et al. 2018

Kagermann, H.; Gaus, N.; Euler, K.; Hauck, J.; Beyerer, J.; Wahlster, W. & Brackemann, H. (2018), *Autonome Systeme - Chancen und Risiken für Wirtschaft, Wissenschaft und Gesellschaft*. Berlin: Fachforum Autonome Systeme im Hightech-Forum.

Kakade & Langford 2002

Kakade, S. & Langford, J. (2002), „Approximately Optimal Approximate Reinforcement Learning“. *Proceedings of the Nineteenth International Conference on Machine Learning, Sydney, 08.07. - 12.07.2002*. Ed. by C. Sammut & A. G. Hoffmann. San Francisco: Morgan Kaufmann Publishers Inc., pp. 267–274.

Kallrath 2013

Kallrath, J. (2013), *Gemischt-ganzzahlige Optimierung: Modellierung in der Praxis*. 2nd ed. Wiesbaden: Springer Fachmedien Wiesbaden.

Kellner & Lienland et al. 2018

Kellner, F.; Lienland, B. & Lukesch, M. (2018), *Produktionswirtschaft*. 1st ed. Berlin, Heidelberg: Springer.

Kim & Lee 1998

Kim, G. H. & Lee, C. S. G. (1998), „Heterogeneous Machine Scheduling Problem“, *IEEE Transactions on Robotics and Automation* 14.6, pp. 879–893.

Kingman 1961

Kingman, J. F. C. (1961), „The single server queue in heavy traffic“. *Mathematical Proceedings of the Cambridge Philosophical Society*. Ed. by F. P. White & E. J. H. Corner. Vol. 57. 4. Cambridge University Press, pp. 902–904.

Kirkpatrick & Gelatt et al. 1983

Kirkpatrick, S.; Gelatt, C. D. & Vecchi, M. P. (1983), „Optimization by Simulated Annealing“, *Science* 220.4598, pp. 671–680.

Klein & Scholl 2011

Klein, R. & Scholl, A. (2011), *Planung und Entscheidung Konzepte, Modelle und Methoden einer modernen betriebswirtschaftlichen Entscheidungsanalyse*. 2nd ed. München: Verlag Franz Vahlen.

Klemmt 2012

Klemmt, A. (2012), *Ablaufplanung in der Halbleiter- und Elektronikproduktion*. 1st ed. Wiesbaden: Vieweg+Teubner Verlag.

Knust 2000

Knust, S. (2000), „Shop-scheduling problems with transportation“. Dissertation. University of Osnabrück.

Koren 2010

Koren, Y. (2010), *The global manufacturing revolution : product-process-business integration and reconfigurable systems*. 1st ed. Hoboken: John Wiley & Sons.

Kuhnle & Röhrig et al. 2019

Kuhnle, A.; Röhrig, N. & Lanza, G. (2019), „Autonomous order dispatching in the semiconductor industry using reinforcement learning“, *Procedia CIRP* 79, pp. 391–396.

Kuhnle & Jakubik et al. 2019

Kuhnle, A.; Jakubik, J. & Lanza, G. (2019), „Reinforcement learning for opportunistic maintenance optimization“, *Production Engineering* 13.1, pp. 33–41.

Kuhnle & Kaiser et al. 2020

Kuhnle, A.; Kaiser, J.-P.; Theiß, F.; Stricker, N. & Lanza, G. (2020), „Designing an Adaptive Production Control System Using Reinforcement Learning“, *Journal of Intelligent Manufacturing* , in press.

Kuhnle & Schäfer et al. 2019

Kuhnle, A.; Schäfer, L.; Stricker, N. & Lanza, G. (2019), „Design, Implementation and Evaluation of Reinforcement Learning for an Adaptive Order Dispatching in Job Shop Manufacturing Systems“, *Procedia CIRP* 81, pp. 234–239.

Kulkarni 2012

Kulkarni, P. (2012), *Reinforcement and systemic machine learning for decision making*. 1st ed. Piscataway: John Wiley & Sons.

Küpper & Sieben et al. 2018

Küpper, D.; Sieben, C.; Kuhlmann, K.; Lim, Y. & Ahmad, J. (2018), *Will Flexible-Cell Manufacturing Revolutionize Carmaking?* Cologne: Boston Consulting Group.

Laarhoven & Aarts et al. 1992

Laarhoven, P. J. M. van; Aarts, E. H. L. & Lenstra, J. K. (1992), „Job Shop Scheduling by Simulated Annealing“, *Operations Research* 40.1, pp. 113–125.

Lanza & Ferdows et al. 2019

Lanza, G.; Ferdows, K.; Kara, S.; Mourtzis, D.; Schuh, G.; Váncza, J.; Wang, L. & Wiendahl, H.-P. (2019), „Global production networks: Design and operation“, *CIRP Annals* 68.2, pp. 823–841.

Lanza & Nyhuis et al. 2018

Lanza, G.; Nyhuis, P.; Liebrecht, C. & Hübner, M. (2018), *Industrie 4.0 für die Praxis: Befähigungs- und Einführungsstrategien*. Garbsen: TEWISS Verlag.

Lasi & Fettke et al. 2014

Lasi, H.; Fettke, P.; Kemper, H.-G.; Feld, T. & Hoffmann, M. (2014), „Industry 4.0“, *Business & Information Systems Engineering* 6.4, pp. 239–242.

Law 2014

Law, A. M. (2014), *Simulation modeling and analysis*. 5th ed. New York: McGraw-Hill Education - Europe.

Lin & Wang et al. 2001

Lin, J. T.; Wang, F.-K. & Yen, P.-Y. (2001), „Simulation analysis of dispatching rules for an automated interbay material handling system in wafer fab“, *International Journal of Production Research* 39.6, pp. 1221–1238.

Little 1961

Little, J. D. C. (1961), „A Proof for the Queuing Formula: $L=\lambda W$“, *Operations Research* 9.3, pp. 383–387.

Lödding 2016

Lödding, H. (2016), *Verfahren der Fertigungssteuerung*. Berlin, Heidelberg: Springer Vieweg.

Lustig & Puget 2001

Lustig, I. J. & Puget, J.-F. (2001), „Program Does Not Equal Program: Constraint Programming and Its Relationship to Mathematical Programming“, *Interfaces* 31.6, pp. 29–53.

Mahadevan & Marchalleck et al. 1997

Mahadevan, S.; Marchalleck, N.; Das, T. K. & Gosavi, A. (1997), „Self-improving factory simulation using continuous-time average-reward reinforcement learning“. *Proceedings of the Fourteenth International Conference on Machine Learning, San Francisco, 08.07. - 12.07.1997*. Ed. by D. H. Fisher. San Francisco: Morgan Kaufmann Publishers Inc., pp. 202–210.

Mahadevan & Theocharous 1998

Mahadevan, S. & Theocharous, G. (1998), „Optimizing Production Manufacturing using Reinforcement Learning“. *Proceedings of the 11th International Florida Artificial Intelligence Research Society Conference, Sanibel Island, 18.05. - 20.05.1998*. Ed. by D. J. Cook. Palo Alto: AAAI Press, pp. 372–377.

März & Krug 2011

März, L. & Krug, W. (2011), „Kopplung von Simulation und Optimierung“. *Simulation und Optimierung in Produktion und Logistik*. 1st ed. Berlin, Heidelberg: Springer, pp. 41–45.

Meier 2019

Meier, P. (2019), *Die Wirtschaft als schwingendes System Erfahrungen, Einsichten, Prognosen*. 1st ed. München: Carl Hanser Verlag.

Meissner & Ilsen et al. 2017

Meissner, H.; Ilsen, R. & Aurich, J. C. (2017), „Analysis of Control Architectures in the Context of Industry 4.0“, *Procedia CIRP* 62, pp. 165–169.

Mitchell 2000

Mitchell, J. E. (2000), „Branch-and-Cut Algorithms for Combinatorial Optimization Problems“. *Handbook of Applied Optimization*. Ed. by P. Pardalos & M. Resende. 1st ed. New York: Oxford University Press, pp. 65–77.

Mitchell 1997

Mitchell, T. M. (1997), *Machine Learning*. 1st ed. Boston: McGraw-Hill Education Ltd.

Mnih & Kavukcuoglu et al. 2013

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D. & Riedmiller, M. (2013), „Playing Atari with Deep Reinforcement Learning“, *arXiv preprint* 1312.5602, pp. 1–9.

Mönch 2006

Mönch, L. (2006), „Agentenbasierte Produktionssteuerung komplexer Produktionssysteme“. Dissertation. Ilmenau: Technische Universitat Ilmenau.

Mönch & Fowler et al. 2011

Mönch, L.; Fowler, J. W.; Dauzère-Pérès, S.; Mason, S. J. & Rose, O. (2011), „A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations“, *Journal of Scheduling* 14.6, pp. 583–599.

Mönch & Fowler et al. 2013

Mönch, L.; Fowler, J. W. & Mason, S. J. (2013), *Production Planning and Control for Semiconductor Wafer Fabrication Facilities*. 1st ed. New York: Springer US.

Monostori & Csáji et al. 2004

Monostori, L.; Csáji, B. & Kádár, B. (2004), „Adaptation and Learning in Distributed Production Control“, *CIRP Annals* 53.1, pp. 349–352.

Monostori & Kádár et al. 2016

Monostori, L.; Kádár, B.; Bauernhansl, T.; Kondoh, S.; Kumara, S.; Reinhart, G.; Sauer, O.; Schuh, G.; Sihn, W. & Ueda, K. (2016), „Cyber-physical systems in manufacturing“, *CIRP Annals* 65.2, pp. 621–641.

Monostori & Váncza et al. 2006

Monostori, L.; Váncza, J. & Kumara, S. (2006), „Agent-Based Systems for Manufacturing“, *CIRP Annals* 55.2, pp. 697–720.

Morch & Kjems et al. 1995

Morch, N.; Kjems, U.; Hansen, L.; Svarer, C.; Law, I.; Lautrup, B.; Strother, S. & Rehm, K. (1995), „Visualization of neural networks using saliency maps“. *Proceedings of ICNN'95 - International Conference on Neural Networks, Perth, 27.11. - 01.12.1995*. Piscataway: IEEE, pp. 2085–2090.

Nemhauser & Wolsey 1999

Nemhauser, G. L. & Wolsey, L. A. (1999), *Integer and combinatorial optimization*. 1st ed. Hoboken: John Wiley & Sons.

Ng & Harada et al. 1999

Ng, A.; Harada, D. & Russell, S. (1999), „Policy invariance under reward transformations: Theory and application to reward shaping“. *Proceedings of the Sixteenth International Conference on Machine Learning, Bled, 27.06. - 30.06.1999*. Ed. by I. Bratko & S. Dzeroski. San Francisco: Morgan Kaufmann Publishers Inc., pp. 278–287.

Nickel & Stein et al. 2014

Nickel, S.; Stein, O. & Waldmann, K.-H. (2014), *Operations Research*. 2nd ed. Berlin, Heidelberg: Gabler Verlag.

Niehues 2016

Niehues, M. R. (2016), „Adaptive Produktionssteuerung für Werkstattfertigungssysteme durch fertigungsbegleitende Reihenfolgebildung“. Dissertation. München: Technische Universität München, p. 279.

Nyhuis 2008

Nyhuis, P. (2008), *Beiträge zu einer Theorie der Logistik*. 1st ed. Berlin, Heidelberg: Springer.

Nyhuis & Wiendahl 2012

Nyhuis, P. & Wiendahl, H.-P. (2012), *Logistische Kennlinien*. 3rd ed. Berlin, Heidelberg: Springer.

Ou & Chang et al. 2018

Ou, X.; Chang, Q.; Arinez, J. & Zou, J. (2018), „Gantry Work Cell Scheduling through Reinforcement Learning with Knowledge-guided Reward Setting“, *IEEE Access* 6, pp. 14699–14709.

Ou & Chang et al. 2019

Ou, X.; Chang, Q. & Chakraborty, N. (2019), „Simulation study on reward function of reinforcement learning in gantry work cell scheduling“, *Journal of Manufacturing Systems* 50, pp. 1–8.

Panwalkar & Iskander 1977

Panwalkar, S. S. & Iskander, W. (1977), „A Survey of Scheduling Rules“, *Operations Research* 25.1, pp. 45–61.

Pardo & Tavakoli et al. 2017

Pardo, F.; Tavakoli, A.; Levdik, V. & Kormushev, P. (2017), „Time Limits in Reinforcement Learning“, *arXiv preprint* 1712.00378, pp. 1–10.

Paternina-Arboleda & Das 2001

Paternina-Arboleda, C. D. & Das, T. K. (2001), „Intelligent dynamic control policies for serial production lines“, *IIE Transactions* 33.1, pp. 65–77.

Paternina-Arboleda & Das 2005

Paternina-Arboleda, C. D. & Das, T. K. (2005), „A multi-agent reinforcement learning approach to obtaining dynamic control policies for stochastic lot scheduling problem“, *Simulation Modelling Practice and Theory* 13.5, pp. 389–406.

Pearl 1984

Pearl, J. (1984), *Heuristics - Intelligent search strategies for computer problem solving*. 1st ed. Reading: Addison-Wesley Publishing Company.

Permin & Bertelsmeier et al. 2016

Permin, E.; Bertelsmeier, F.; Blum, M.; Bützler, J.; Haag, S.; Kuz, S.; Özdemir, D.; Stemmler, S.; Thombansen, U.; Schmitt, R.; Brecher, C.; Schlick, C.; Abel, D.; Poprawe, R.; Loosen, P.; Schulz, W. & Schuh, G. (2016), „Self-optimizing Production Systems“, *Procedia CIRP* 41, pp. 417–422.

Pinedo 2016

Pinedo, M. L. (2016), *Scheduling*. 5th ed. Cham: Springer International Publishing.

Prabhu & Duffie 1999

Prabhu, V. & Duffie, N. (1999), „Nonlinear dynamics in distributed arrival time control of heterarchical manufacturing systems“, *IEEE Transactions on Control Systems Technology* 7.6, pp. 724–730.

Puterman 1994
Puterman, M. L. (1994), *Markov decision processes: discrete stochastic dynamic programming*. 1st ed. New York: John Wiley & Sons.

Qu & Wang et al. 2016
Qu, S.; Wang, J.; Govil, S. & Leckie, J. O. (2016), „Optimized Adaptive Scheduling of a Manufacturing Process System with Multi-skill Workforce and Multiple Machine Types: An Ontology-based, Multi-agent Reinforcement Learning Approach“, *Procedia CIRP* 57, pp. 55–60.

Randløv & Alstrøm 1998
Randløv, J. & Alstrøm, P. (1998), „Learning to Drive a Bicycle using Reinforcement Learning and Shaping“. *Proceedings of the Fifteenth International Conference on Machine Learning, Madison, 24.07. - 27.07.1998*. Ed. by J. W. Shavlik. San Francisco: Morgan Kaufmann Publishers Inc., pp. 463–471.

Riedmiller & Riedmiller 1999
Riedmiller, S. & Riedmiller, M. (1999), „A neural reinforcement learning approach to learn local dispatching policies in production scheduling“. *Proceedings of the 16th international joint conference on Artificial intelligence, Stockholm, 31.07. - 06.08.1999*. Ed. by T. Dean. San Francisco: Morgan Kaufmann Publishers Inc., pp. 764–769.

Robinson 2014
Robinson, S. (2014), *Simulation: The practice of model development and use*. 2nd ed. New York: Palgrave Macmillan.

Rothlauf 2006
Rothlauf, F. (2006), „Representations for Genetic and Evolutionary Algorithms“. *Representations for Genetic and Evolutionary Algorithms*. 2nd ed. Berlin, Heidelberg: Springer, pp. 9–32.

Rummery & Niranjan 1994
Rummery, G. & Niranjan, M. (1994), „On-line Q-learning using connectionist systems“. Dissertation. Cambridge: University of Cambridge.

Russell & Norvig 2016
Russell, S. J. & Norvig, P. (2016), *Artificial Intelligence: A Modern Approach*. 3rd ed. Harlow: Pearson Education Limited.

Sarin & Varadarajan et al. 2011
Sarin, S. C.; Varadarajan, A. & Wang, L. (2011), „A survey of dispatching rules for operational control in wafer fabrication“, *Production Planning & Control* 22.1, pp. 4–24.

Schmidhuber 2015

Schmidhuber, J. (2015), „Deep Learning in neural networks: An overview“, *Neural Networks* 61, pp. 85–117.

Scholz-Reiter & Freitag et al. 2005

Scholz-Reiter, B.; Freitag, M.; De Beer, C. & Jagalski, T. (2005), „Modelling and analysis of autonomous shop floor control“. *Proceedings of the 38th CIRP International Seminar on Manufacturing Systems, Florianopolis, 16.05. - 18.05.2005*. Ed. by L. Weingärtner & E. Westkämper. Paris: CIRP, pp. 16–18.

Scholz-Reiter & Görges et al. 2009

Scholz-Reiter, B.; Görges, M. & Philipp, T. (2009), „Autonomously controlled production systems—Influence of autonomous control level on logistic performance“, *CIRP Annals* 58.1, pp. 395–398.

Scholz-Reiter & Hamann 2008

Scholz-Reiter, B. & Hamann, T. (2008), „The behaviour of learning production control“, *CIRP Annals - Manufacturing Technology* 57.1, pp. 459–462.

Scholz-Reiter & Harjes et al. 2010

Scholz-Reiter, B.; Harjes, F. & Hamann, T. (2010), „Automatisierung des Lernens neuronaler Netze in der Produktionssteuerung“, *ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb* 105.1-2, pp. 101–105.

Schönemann & Herrmann et al. 2015

Schönemann, M.; Herrmann, C.; Greschke, P. & Thiede, S. (2015), „Simulation of matrix-structured manufacturing systems“, *Journal of Manufacturing Systems* 37.1, pp. 104–112.

Schuh & Monostori et al. 2008

Schuh, G.; Monostori, L.; Csáji, B. & Döring, S. (2008), „Complexity-based modeling of reconfigurable collaborations in production industry“, *CIRP Annals* 57.1, pp. 445–450.

Schuh & Reuter et al. 2017

Schuh, G.; Reuter, C.; Prote, J.-P.; Brambring, F. & Ays, J. (2017), „Increasing data integrity for improving decision making in production planning and control“, *CIRP Annals* 66.1, pp. 425–428.

Schuh & Stich 2012

Schuh, G. & Stich, V. (2012), *Produktionsplanung und -steuerung 1 - Grundlagen der PPS*. 4th ed. Berlin, Heidelberg: Springer.

Schulman & Levine et al. 2015

Schulman, J.; Levine, S.; Moritz, P.; Jordan, M. I. & Abbeel, P. (2015), „Trust Region Policy Optimization“. *Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 06.07. - 11.07.2015*. Ed. by F. Bach & D. Blei. PMLR, pp. 1889–1897.

SEMI E10-0304 2004

SEMI E10-0304 (2004), *Specification for Definition and Measurement of Equipment Reliability, Availability and Maintainability (RAM)*. San Jose, CA: SEMI - Semiconductor Equipment and Materials International.

Shah & Gosavi et al. 2010

Shah, P.; Gosavi, A. & Nagi, R. (2010), „A machine learning approach to optimise the usage of recycled material in a remanufacturing environment“, *International Journal of Production Research* 48.4, pp. 933–955.

Shahrabi & Adibi et al. 2017

Shahrabi, J.; Adibi, M. A. & Mahootchi, M. (2017), „A reinforcement learning approach to parameter estimation in dynamic job shop scheduling“, *Computers & Industrial Engineering* 110, pp. 75–82.

Shiue & Lee et al. 2018

Shiue, Y.-R.; Lee, K.-C. & Su, C.-T. (2018), „Real-time scheduling for a smart factory using a reinforcement learning approach“, *Computers & Industrial Engineering* 125, pp. 604–614.

Silver & Hubert et al. 2018

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; Lillicrap, T.; Simonyan, K. & Hassabis, D. (2018), „A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play“, *Science* 362.6419, pp. 1140–1144.

Silver & Schrittwieser et al. 2017

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; Chen, Y.; Lillicrap, T.; Hui, F.; Sifre, L.; Driessche, G. van den; Graepel, T. & Hassabis, D. (2017), „Mastering the game of Go without human knowledge“, *Nature* 550.7676, pp. 354–359.

Simonyan & Vedaldi et al. 2014

Simonyan, K.; Vedaldi, A. & Zisserman, A. (2014), „Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps“, *arXiv preprint* 1312.6034, pp. 1–8.

Stegherr 2000

Stegherr, F. (2000), „Reinforcement-Learning zur dispositiven Auftragssteuerung in der Variantenreihenproduktion“. Dissertation. München: Technical University of Munich.

Stricker 2016

Stricker, N. (2016), „Robustheit verketteter Produktionssysteme: Robustheitsevaluation und Selektion des Kennzahlensystems der Robustheit“. Dissertation. Karlsruhe: Karslruhe Institute of Technology.

Stricker & Kuhnle et al. 2018

Stricker, N.; Kuhnle, A.; Sturm, R. & Friess, S. (2018), „Reinforcement learning for adaptive order dispatching in the semiconductor industry“, *CIRP Annals* 67.1, pp. 511–514.

Sturm 2006

Sturm, R. (2006), „Modellbasiertes Verfahren zur Online-Leistungsbewertung von automatisierten Transportsystemen in der Halbleiterfertigung“. Dissertation. Stuttgart: Universität Stuttgart.

Sutton & Barto 2018

Sutton, R. S. & Barto, A. G. (2018), *Reinforcement Learning: An Introduction*. 2nd ed. Cambridge, MA: MIT Press.

Switalski 1989

Switalski, M. (1989), *Hierarchische Produktionsplanung*. Heidelberg: Physica-Verlag Heidelberg.

Tesauro 1992

Tesauro, G. (1992), „Practical Issues in Temporal Difference Learning“, *Machine Learning* 8.3, pp. 257–277.

Thorndike 1911

Thorndike, E. L. (1911), *Animal Intelligence: Experimental Studies*. New York: The Macmillan Co.

Tsakalis & Flores-Godoy et al. 1997

Tsakalis, K. S.; Flores-Godoy, J. J. & Rodriguez, A. A. (1997), „Hierarchical modeling and control for re-entrant semiconductor fabrication lines: A mini-fab benchmark“. *1997 IEEE 6th International Conference on Emerging Technologies and Factory Automation Proceedings, Los Angeles, 09.09. - 12.09.1997*. Ed. by C. J. Kim & R. Zurawski. Piscataway: IEEE, pp. 508–513.

Uzsoy & Lee et al. 1992

Uzsoy, R.; Lee, C. Y. & Martin-Vega, L. A. (1992), „A Review of Production Planning and Scheduling Models in the Semiconductor Industry Part I: System Characteristics, Performance Evaluation and Production Planning“, *IIE Transactions* 24.4, pp. 47–60.

Uzsoy & Lee et al. 1994

Uzsoy, R.; Lee, C. Y. & Martin-Vega, L. A. (1994), „A review of production planning and scheduling models in the semiconductor industry part II: Shop-floor control“, *IIE Transactions (Institute of Industrial Engineers)* 26.5, pp. 44–55.

VDI 3633 Part 1 2014

VDI 3633 Part 1 (2014), *Simulation of systems in materials handling, logistics and production - Fundamentals*. Berlin: Verein Deutscher Ingenieure e.V.

VDI 4400 Part 2 2004

VDI 4400 Part 2 (2004), *Logistic indicators for production*. Berlin: Verein Deutscher Ingenieure e.V.

Vepsalainen & Morton 1987

Vepsalainen, A. P. J. & Morton, T. E. (1987), „Priority Rules for Job Shops with Weighted Tardiness Costs“, *Management Science* 33.8, pp. 1035–1047.

Verma & Murali et al. 2018

Verma, A.; Murali, V.; Singh, R.; Kohli, P. & Chaudhuri, S. (2018), „Programmatically Interpretable Reinforcement Learning“. *Proceedings of the 35th International Conference on Machine Learning, Stockholm, 10.07. - 15.07.2018*. Ed. by J. Dy & A. Krause. PMLR, pp. 8024–8033.

Vieira & Herrmann et al. 2003

Vieira, G. E.; Herrmann, J. W. & Lin, E. (2003), „Rescheduling Manufacturing Systems: A Framework of Strategies, Policies, and Methods“, *Journal of Scheduling* 6.1, pp. 39–62.

Wang & Usher 2004

Wang, Y.-C. & Usher, J. M. (2004), „Learning policies for single machine job dispatching“, *Robotics and Computer-Integrated Manufacturing* 20.6, pp. 553–562.

Wang & Usher 2005

Wang, Y.-C. & Usher, J. M. (2005), „Application of reinforcement learning for agent-based production scheduling“, *Engineering Applications of Artificial Intelligence* 18.1, pp. 73–82.

Wang & Li et al. 2012

Wang, J.; Li, X. & Zhu, X. (2012), „Intelligent dynamic control of stochastic economic lot scheduling by agent-based reinforcement learning“, *International Journal of Production Research* 50.16, pp. 4381–4395.

Wang & Wang et al. 2016

Wang, X.; Wang, H. & Qi, C. (2016), „Multi-agent reinforcement learning based maintenance policy for a resource constrained flow line system“, *Journal of Intelligent Manufacturing* 27.2, pp. 325–333.

Waschneck 2018

Waschneck, B. (2018), „Autonome Entscheidungsfindung in der Produktionssteuerung komplexer Werkstattfertigungen“. Dissertation. Stuttgart: University of Stuttgart.

Waschneck & Altenmüller et al. 2018a

Waschneck, B.; Altenmüller, T.; Bauernhansl, T.; Knapp, A.; Kyek, A.; Reichstaller, A.; Belzner, L.; Altenmuller, T.; Bauernhansl, T.; Knapp, A. & Kyek, A. (2018a), „Deep reinforcement learning for semiconductor production scheduling“. *2018 29th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC 2018), Saratoga Springs, NY, 30.04. - 03.05.2018*. Ed. by A. Roussy & R. Van Roijen. Piscataway: IEEE, pp. 301–306.

Waschneck & Altenmüller et al. 2016

Waschneck, B.; Altenmüller, T.; Bauernhansl, T. & Kyek, A. (2016), „Production Scheduling in Complex Job Shops from an Industrie 4.0 Perspective: A Review and Challenges in the Semiconductor Industry“. *SamI40 workshop at i-KNOW 2016*. Graz, 18.10. - 19.10.2016, pp. 1–12.

Waschneck & Altenmüller et al. 2018b

Waschneck, B.; Altenmüller, T.; Bauernhansl, T. & Kyek, A. (2018b), „Case Study on Operator Compliance to Scheduling Decisions in Semiconductor Manufacturing“. *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE), München, 20.08. - 24.08.2018*. Ed. by B. Vogel-Heuser & B. Lennartson. Piscataway: IEEE, pp. 649–652.

Waschneck & Reichstaller et al. 2018

Waschneck, B.; Reichstaller, A.; Belzner, L.; Altenmüller, T.; Bauernhansl, T.; Knapp, A. & Kyek, A. (2018), „Optimization of global production scheduling with deep reinforcement learning“, *Procedia CIRP* 72, pp. 1264–1269.

Watkins & Dayan 1992

Watkins, C. J. C. H. & Dayan, P. (1992), „Q-learning“, *Machine Learning* 8.3-4, pp. 279–292.

Wauters & Verbeeck et al. 2011

Wauters, T.; Verbeeck, K.; Berghe, G. V. & De Causmaecker, P. (2011), „Learning agents for the multi-mode project scheduling problem“, *Journal of the Operational Research Society* 62.2, pp. 281–290.

Wiendahl & ElMaraghy et al. 2007

Wiendahl, H.-P.; ElMaraghy, H.; Nyhuis, P.; Zäh, M.; Wiendahl, H.-H.; Duffie, N. & Brieke, M. (2007), „Changeable Manufacturing - Classification, Design and Operation“, *CIRP Annals* 56.2, pp. 783–809.

Wiendahl & Scholtissek 1994

Wiendahl, H.-P. & Scholtissek, P. (1994), „Management and Control of Complexity in Manufacturing“, *CIRP Annals* 43.2, pp. 533–540.

Wiendahl 1997

Wiendahl, H.-P. (1997), *Fertigungsregelung: Logistische Beherrschung von Fertigungsabläufen auf Basis des Trichtermodells*. München: Hanser.

Wiendahl & Reichardt et al. 2014

Wiendahl, H.-P.; Reichardt, J. & Nyhuis, P. (2014), *Handbuch Fabrikplanung*. München: Carl Hanser Verlag.

Wiewiora 2011

Wiewiora, E. (2011), „Reward Shaping“. *Encyclopedia of Machine Learning*. Ed. by C. Sammut & G. Webb. Boston, MA: Springer, pp. 863–865.

Witten 1976

Witten, I. H. (1976), „The apparent conflict between estimation and control - a survey of the two-armed bandit problem“, *Journal of the Franklin Institute* 301.1-2, pp. 161–189.

Wolpert & Macready 1997

Wolpert, D. & Macready, W. (1997), „No free lunch theorems for optimization“, *IEEE Transactions on Evolutionary Computation* 1.1, pp. 67–82.

Zeng & Yang et al. 2009

Zeng, Q.; Yang, Z. & Lai, L. (2009), „Models and algorithms for multi-crane oriented scheduling method in container terminals“, *Transport Policy* 16.5, pp. 271–278.

Zhang & Dietterich 1995

Zhang, W. & Dietterich, T. G. (1995), „High-Performance Job-Shop Scheduling With A Time-Delay TD(lambda) Network“. *Advances in Neural Information Processing Systems 8, Denver, 27.11. - 02.12.1995*. Ed. by D. Touretzky; M. Mozer & M. Hasselmo. Cambridge, MA: MIT Press, pp. 1024–1030.

Zhang & Zheng et al. 2011

Zhang, Z.; Zheng, L.; Hou, F. & Li, N. (2011), „Semiconductor final test scheduling with Sarsa(λ, k) algorithm“, *European Journal of Operational Research* 215.2, pp. 446–458.

Ziarnetzky 2017

Ziarnetzky, T. (2017), „Produktionsplanung in der Halbleiterfertigung: Modellierung, Lösungsansätze und simulationsbasierte Leistungsbewertung“. Dissertation. Hagen: FernUniversität in Hagen.

Zimmermann 2008

Zimmermann, H.-J. (2008), *Operations Research*. 2nd ed. Wiesbaden: Vieweg+Teubner Verlag.

List of Figures

List of Tables

List of own publications

Altenmüller, T.; Stüker, T.; Waschneck, B.; Kuhnle, A. & Lanza, G. (2020), „Reinforcement learning for an intelligent and autonomous production control of complex job-shops under time constraints“, *Production Engineering* 14.3, pp. 319–328.

Benfer, M.; Gartner, P.; Treber, S.; Kuhnle, A.; Häfner, B. & Lanza, G. (2020), „Implementierung von unternehmensübergreifender Traceability“, *ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb* 115.5, pp. 304–308.

Greinacher, S.; Overbeck, L.; Kuhnle, A.; Krahe, C. & Lanza, G. (2020), „Multi-objective optimization of lean and resource efficient manufacturing systems“, *Production Engineering* 14.2, pp. 165–176.

Hübner, M.; Liebrecht, C.; Malessa, N.; Kuhnle, A.; Nyhuis, P. & Lanza, G. (2017), „A process model for implementing Industry 4.0 - Introduction of a process model for the individual implementation of Industry 4.0 methods | Vorgehensmodell zur Einführung von Industrie 4.0: Vorstellung eines Vorgehensmodells zur bedarfsgerechten Einführung“, *WT Werkstattstechnik* 107.4, pp. 266–272.

Kuhnle, A.; Röhrig, N. & Lanza, G. (2019), „Autonomous order dispatching in the semiconductor industry using reinforcement learning“, *Procedia CIRP* 79, pp. 391–396.

Kuhnle, A.; Jakubik, J. & Lanza, G. (2019), „Reinforcement learning for opportunistic maintenance optimization“, *Production Engineering* 13.1, pp. 33–41.

Kuhnle, A.; Kaiser, J.-P.; Theiß, F.; Stricker, N. & Lanza, G. (2020), „Designing an Adaptive Production Control System Using Reinforcement Learning“, *Journal of Intelligent Manufacturing*, pp. 1–22.

Kuhnle, A. & Lanza, G. (2019), „Investigation of closed-loop supply chains with product refurbishment as integrated location-inventory problem“, *Production Engineering* 13.3-4, pp. 293–303.

Kuhnle, A.; Schäfer, L.; Stricker, N. & Lanza, G. (2019), „Design, Implementation and Evaluation of Reinforcement Learning for an Adaptive Order Dispatching in Job Shop Manufacturing Systems“, *Procedia CIRP* 81, pp. 234–239.

Liebrecht, C.; Jacob, A.; Kuhnle, A. & Lanza, G. (2017), „Multi-criteria Evaluation of Manufacturing Systems 4.0 under Uncertainty“, *Procedia CIRP* 63, pp. 224–229.

Stricker, N.; Kuhnle, A.; Sturm, R. & Friess, S. (2018), „Reinforcement learning for adaptive order dispatching in the semiconductor industry“, *CIRP Annals* 67.1, pp. 511–514.

Ungermann, F.; Kuhnle, A.; Stricker, N. & Lanza, G. (2019a), „Data analytics for manufacturing systems – A data-driven approach for process optimization“, *Procedia CIRP* 81, pp. 369–374.

Ungermann, F.; Kuhnle, A.; Stricker, N. & Lanza, G. (2019b), „Entscheidungsunterstützungssysteme in der Produktion“, *ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb* 114.1-2, pp. 34–38.

Wagner, R.; Schleich, B.; Haefner, B.; Kuhnle, A.; Wartzack, S. & Lanza, G. (2019), „Challenges and potentials of digital twins and industry 4.0 in product design and production for high performance products“, *Procedia CIRP* 84, pp. 88–93.

Appendix

A1 Explanation of quantitative optimization methods

Linear optimization Linear programs consist of a linear objective function and constraints. All variables are real values. Most basic planning problems, such as production planning or transport planning, fall into this category (Domschke; Drexl et al. 2015). The Simplex algorithm is an efficient solution algorithm for linear optimization problems (Bertsimas & Tsitsiklis 1997).

Mixed-integer optimization More complex, in terms of computational complexity (see Section 2.1.1), are integer or mixed-integer programs for which some decision variables are restricted to integer or binary values (Nemhauser & Wolsey 1999). These variables enable the consideration of logical constraints as well as the modelling of resources that cannot be split. For instance, resource allocation, sequencing, or selection problems require mixed-integer variables (Domschke; Drexl et al. 2015). These problems are mostly combinatorial problems. They are more complex as numerous alternatives have to be evaluated without using the properties of continuous, linear problems that enable efficient mathematical operations. They are commonly solved by cutting-plane methods (Mitchell 2000), such as first introduced by Gomory (1958) and Benders (1962), or Branch-and-Bound methods (Brucker & Knust 2012).

Non-linear optimization Non-linear optimization problems have either a non-linear objective function or at least one non-linear constraint. They are far more computationally complex to solve, as convexity is not ensured. Hence, optimal solutions are not necessarily corner points of the solution space (Zimmermann 2008). However, some specific forms of non-linear problems can be solved efficiently, e.g. second-order cone programs (Alizadeh & Goldfarb 2003). General solution approaches translate the problem with the help of Lagrange multiplier into a problem without constraints that can be solved by gradient methods, such as Newton's method (Zimmermann 2008).

Dynamic optimization Dynamic optimization, also named dynamic programming, assumes a different problem setup and particularly considers problems that range over multiple time steps (Domschke; Drexl et al. 2015). The optimal solution can be determined by recursion, i.e. defining a sub-problem for each time step and solving each separately. However, this is computationally expensive and requires large memory storage for all possible stages (Mönch;

Fowler et al. 2013). Dynamic programming is commonly applied in stochastic decision problems, in particular for MDPs, such as inventory management and lot size planning. They are solved using Bellman's principle of optimality and backward introduction (Bellman 1966).

Constrained optimization Finally, constrained optimization is a rather new field of research that makes use of established methods from computer science that are based on logical programming (Lustig & Puget 2001; Hofstedt & Wolf 2007). Especially, scheduling problems are highly constrained problems, e.g. precedence, sequence, and due date constraints. Hence, they are hard to solve when formulated as mixed-integer program due to the large number of variables that is required (Baptiste; Pape et al. 2001). Logical programming techniques are more efficient in that case.

Online optimization The mathematical optimization methods explained so far assume that the input data is thoroughly known. In practice, however, this is often not the case, as data becomes known over time and requires repeated decisions under uncertainty (Dunke & Nickel 2019). Online optimization assumes that there is none or just limited knowledge about the future and one needs to optimize based on partial knowledge of the present and past. The optimization objective is to ensure not to be too far from a retrospectively optimal solution, even in the worst case. For further details on the foundations and latest research, the reader is referred to Hazan (2016) and Dunke & Nickel (2017).

Heuristics Greedy heuristics can determine the optimal solution but are generally described as too short-sighted (myopic), since the selection of the best alternative at a point in time does not include the following steps (Pearl 1984). Procedures that take this into account randomize the selection of alternatives or apply look-ahead strategies (Mönch; Fowler et al. 2013). Furthermore, heuristics are formulated in a problem-specific way and do not search the solution space of the optimization problem holistically (Zimmermann 2008). Some examples are listed in the following: The k-opt heuristic gives a well-known example in the area of mixed-integer programs for the travelling salesman problem and Kruskal's algorithm for finding a minimum-spanning-tree (Nickel; Stein et al. 2014). For simple scheduling applications, common heuristics are priority rules. For multi-level scheduling problems, e.g. in a job shop manufacturing system, the shifting bottleneck algorithm by Adams; Balas et al. (1988) provides a widespread and superior heuristic (Acker 2011; Heger; Branke et al. 2016). In each iteration it selects the machine that raises the total cycle time the most, i.e. the bottleneck machine, and determines the schedule for this machine. However, the algorithm does not consider, e.g.

breakdowns of machines, which leads to poor performance in complex production systems (Mönch 2006).

Metaheuristics Common metaheuristics are the following: local (neighbourhood) search that is sometimes also called hill climbing algorithm (Hansen & Mladenović 2001), tabu search (Glover & Laguna 1998), simulated annealing (Kirkpatrick; Gelatt et al. 1983), and evolutionary algorithms (Goldberg & Holland 1988; Rothlauf 2006). Search heuristics evaluate the neighbourhood of a solution, wherein the neighbourhood is determined by solutions that are reachable by search operations that, for instance, switch the sequence of two jobs in a schedule. Simple optimization heuristics would just select better or the best solution in the neighbourhood. However, in order to reduce the chance of getting stuck in a local optimum, randomized search procedures show a better performance. An example is given by the randomized-greedy procedure introduced by Feo & Resende (1995) or the tabu search, which ignores solutions within a tabu list of last visited solutions. Simulated annealing imitates the natural processes of metal cooling and, for instance, the degree of randomness or the probability of selecting a worse solution is decreasing over time, i.e. cooling down, to still enforce a state of convergence to the end of the cooling process. An example of job shop scheduling is found in the work of Laarhoven; Aarts et al. (1992).

Unsupervised learning For unsupervised learning, only unstructured input data is provided. The number of reasonable clusters into which the database can be divided is unknown. Therefore, the aim of clustering is to determine the number of clusters and their boundaries. Unsupervised learning is particularly useful when data need to be understood and explored, as there is no prior knowledge required. However, an application in a domain where an optimization objective needs to be achieved, such as order dispatching, does not match the unsupervised nature. Hence, it is not considered in the remainder of this research.

Supervised learning In supervised learning, the input and output data are explicitly defined. In contrast to unsupervised learning, the range of values is already determined before the learning process. The learning process aims to learn the relation between input and output data. For a continuous numeric output, the relationship between an input and the correct associated output is represented by a regression function and for discrete output values via a classification function. The trained model can be applied to data, previously not used in the training process, for quality assessment. A disadvantage of supervised learning is the amount of data that needs to be gathered, which might cause a considerable manual data labelling effort. In principle, supervised learning would match the requirements of the order dispatching problem, but the entire training data is not known upfront due to the dynamic nature of the production environment. Hence, it does not sufficiently address the adaptivity requirement.

Model-free Reinforcement Learning methods

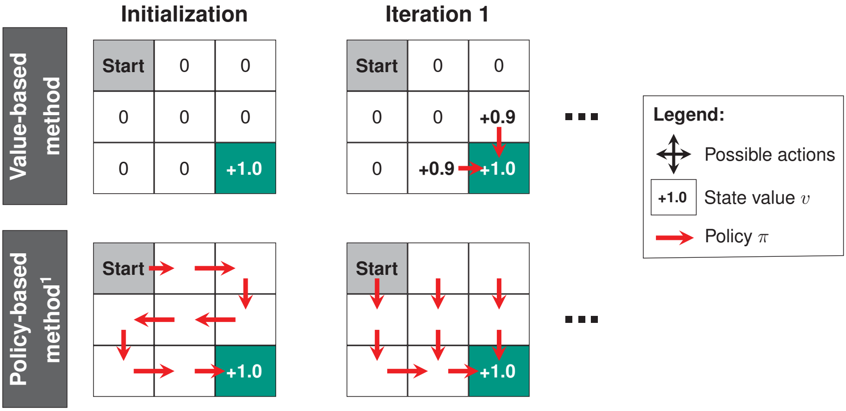


Figure A1.1: Comparison of value- and policy-based methods for one optimization iteration. The goal of the simplified example is to find the shortest path from the start in the upper left to the goal in the lower right corner.

[1] Although the initialization of the policy-based method is, in general, randomized, too, in the example an artificial initialization is displayed to depict the update more precisely.

A2 List of simulation parameters

Table A2.1: List of parametrizable order- and resource-related simulation parameters.

Sources	**Value range and unit**
Capacity outbound buffer	$\mathbb{N}^{\geq 0}$
Product variants	List of machines, subset of M
Mean arrival time	$\mathbb{R}^{\geq 0}$, $[\mathrm{TU}]$
Machines	**Value range and unit**
Capacity inbound buffer	$\mathbb{N}^{\geq 0}$
Capacity outbound buffer	$\mathbb{N}^{\geq 0}$
MTBF	$\mathbb{R}^{\geq 0}$, $[\mathrm{TU}]$
MTOL, repair time	$\mathbb{R}^{\geq 0}$, $[\mathrm{TU}]$
Machine group	$\mathbb{N}^{\geq 0}$
Internal order sequencing rule	FIFO
Sinks	**Value range and unit**
Capacity	$\mathbb{N}^{\geq 0}$
Transport	**Value range and unit**
Capacity	$\mathbb{N}^{\geq 0}$
Movement speed	$\mathbb{R}^{\geq 0}$, $\left[\frac{\mathrm{DU}}{\mathrm{TU}}\right]$
Time to load machine	$\mathbb{R}^{\geq 0}$, $[\mathrm{TU}]$
Time to unload machine	$\mathbb{R}^{\geq 0}$, $[\mathrm{TU}]$
Time to load sink	$\mathbb{R}^{\geq 0}$, $[\mathrm{TU}]$
Time to unload source	$\mathbb{R}^{\geq 0}$, $[\mathrm{TU}]$
Responsible work area	List of resources, subset of $\mathcal{R}$
Orders	**Value range and unit**
List of processes	List of machines, subset of M
Mean processing time	$\mathbb{R}^{\geq 0}$, $[\mathrm{TU}]$
Min. processing time	$\mathbb{R}^{\geq 0}$, $[\mathrm{TU}]$
Max. processing time	$\mathbb{R}^{\geq 0}$, $[\mathrm{TU}]$
Demand per variant	$\mathbb{R}^{\geq 0}$, $\left[\frac{\mathrm{orders}}{\mathrm{day}}\right]$

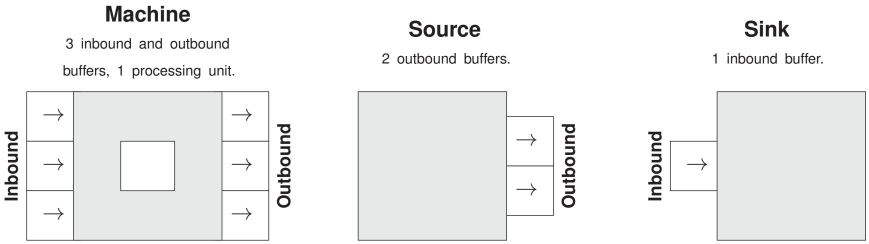


Figure A2.1: Example illustration of resource types with inbound and outbound buffers (load ports).

A3 Methodological approach of modelling RL-based adaptive order dispatching

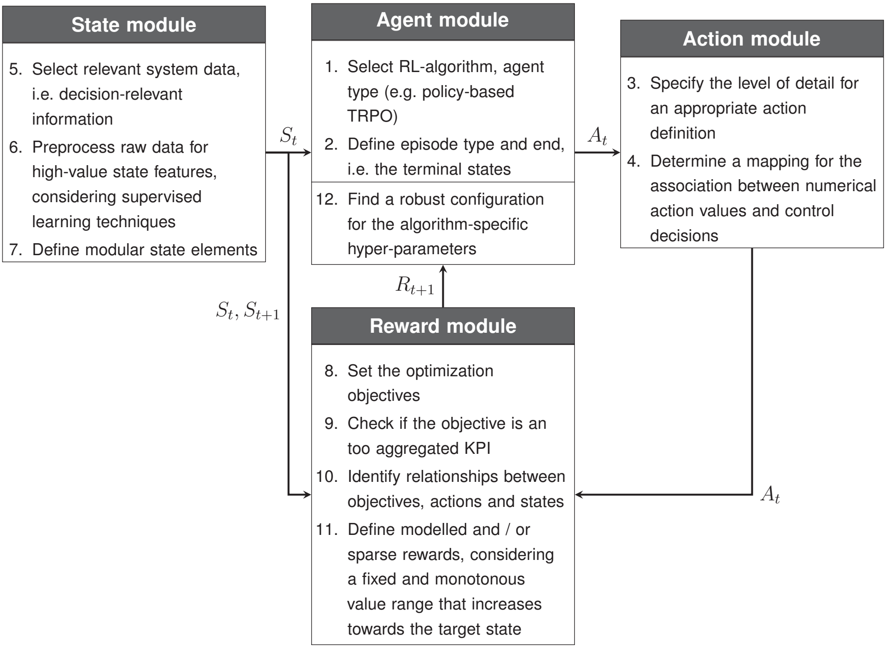

State module
5. Select relevant system data, i.e. decision-relevant information
6. Preprocess raw data for high-value state features, considering supervised learning techniques
7. Define modular state elements
Agent module
1. Select RL-algorithm, agent type (e.g. policy-based TRPO)
2. Define episode type and end, i.e. the terminal states
12. Find a robust configuration for the algorithm-specific hyper-parameters
Action module
3. Specify the level of detail for an appropriate action definition
4. Determine a mapping for the association between numerical action values and control decisions
Reward module
8. Set the optimization objectives
9. Check if the objective is an too aggregated KPI
10. Identify relationships between objectives, actions and states
11. Define modelled and / or sparse rewards, considering a fixed and monotonous value range that increases towards the target state
S_t
A_t
R_{t+1}
S_t, S_{t+1}
A_t

Figure A3.1: Overview of step-by-step approach for modelling the RL-based adaptive order dispatching introduced in Figure 4.7.

A4 Computational results for varying hyper-parameter settings

The computational results in Figure A4.1 are based on Scenario 2 (see Section 5.1.2 for more details) as this scenario is less restricted and, hence, easier to learn for the RL-agent. However, the results are transferable to Scenario 1. The figure shows the mean episode reward values as well as the moving mean standard deviation, aggregated from three repetitive simulation runs. The moving mean window is set to 100.

The state representation includes the S_{VA} feature and a constant reward r_{const} with the constant reward value of 0.2 is given for any valid action $A_{valid,t}$. This represents a basic RL-agent, which allows the analysis of the hyper-parameter influences as any other factors are excluded, such as complex state features or reward functions.

The default values of the present work are an ANN configuration of 128×128, $\alpha = 0.001$, $\gamma = 0.9$, and ϵ turned off. Apparently, the ANN configuration does not have a significant influence on the RL-agent's performance. Next, one can see that for an increasing learning rate α the agent performs less stable, in particular for $\alpha = 0.1$. On the other hand, a small α slows down the learning process, what can be seen for $\alpha = 0.0001$. Furthermore, a higher γ-value of 0.95 shows a slower learning process comparing to the default setting. In this simple RL-agent configuration, a smaller γ even further increased the learning speed, however this could not be confirmed for more complex agent configurations as analysed in Chapter 5. Finally, using ϵ-decay from $0.9 \rightarrow 0.001$ in 4 Mio. iterations does result in a longer learning phase, too.

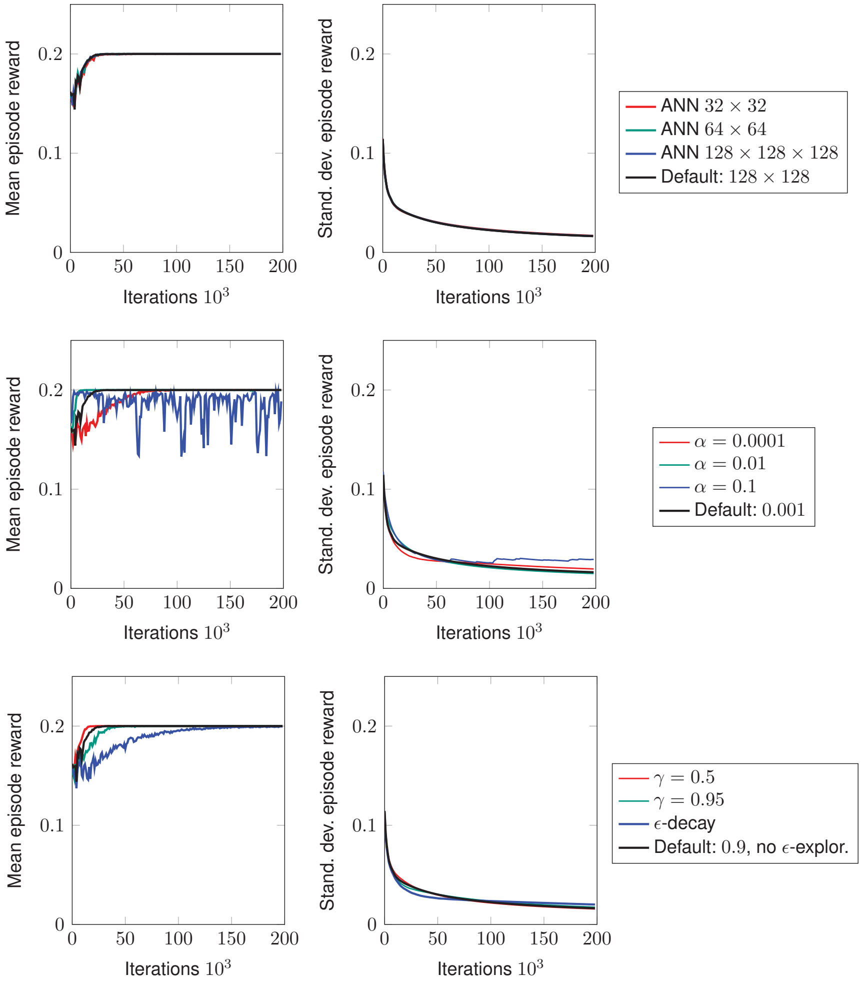


Figure A4.1: Varying hyper-parameter settings.

A5 Theoretical computations of the system's capacity dimensions

Table A5.1: Analytical calculation of machine capacities.

Machine	Work load $[\mathrm{TU}]$	Utilization* (excluding breakdowns) $[\%]$	Work load + breakdowns $[\mathrm{TU}]$	Utilization* $[\%]$
	A	B	C	D
M_1	1247.0	86.6	1463.4	101.6
M_2	1230.8	85.5	1395.0	96.9
M_3	1256.1	87.2	1405.3	97.6
M_4	1121.1	77.9	1430.9	99.4
M_5	1267.5	88.0	1393.5	96.8
M_6	1266.5	88.0	1417.6	98.4
M_7	1255.8	87.2	1405.6	97.6
M_8	1155.0	80.2	1417.7	98.5
Average			**1416.1**	**98.3**

Calculation rule per column:

Column A: $\text{Demand rate} \cdot \text{Mean process time}$

Column B: $\frac{\text{Column A}}{1440}$

Column C: $\text{Column A} \cdot \frac{1440}{\text{MTBF}} \cdot \text{MTOL}$

Column D: $\frac{\text{Column C}}{1440}$

* The utilization values are calculated with the basis of 1440, i.e. the number of minutes per day.

Table A5.2: Analytical calculation of dispatching operator capacities for varying transportation speed values $v = 0.3$.

Machine	**Dispatching operator work load** $[\mathrm{TU}]$ (average, $v = 1.0$)	**Dispatching operator work load** $[\mathrm{TU}]$ (average, $v = 0.3$)	**Dispatching operator work load** $[\mathrm{TU}]$ (minimal, $v = 0.3$)
	A	B	C
M_1	138.8	302.6	135.4
M_2	85.0	172.7	93.7
M_3	87.7	212.0	129.5
M_4	95.5	226.2	140.1
M_5	158.9	403.0	244.1
M_6	156.5	325.3	167.9
M_7	161.9	336.1	174.0
M_8	261.5	606.3	324.9
Total $\sum$	1145.8 (79.6%)*	2584.1 (179.5%)*	1409.6 (97.9%)*

Calculation assumption per column:

The work load is computed based on the demand rate per machine multiplied by the total transport and handling time.

Columns A and B: Assuming average transport distances for the destination machine out of the set of feasible machines as well as for distances to the order's origin to pick it up, i.e. assuming the probability of the location of the operator relative to the pick-up location is evenly distributed.

Column C: Assuming minimal transport distances for the destination machine out of the set of feasible machines as well as for distances to the order's origin to pick it up, i.e. assuming that the operator is always directly at the pick-up location.

* The percentage utilization values in brackets are calculated with the basis of 1440, i.e. the number of minutes per day.

A6 Recursion limit, forced idle time, and episode length parameter analysis

The computational results in Figure A6.1 are based on Scenario 2 (see Section 5.1.2 for more details) as this scenario is less restricted and, hence, easier to learn for the RL-agent. However, the results are transferable to Scenario 1. The figure shows the moving mean episode reward values as well as the moving mean standard deviation, aggregated from three repetitive simulation runs. The moving mean window is set to 100. The state representation includes the features $S_{VA}, S_{AT}, S_{BEN}, S_{BEX}$ and the utilization reward r_{util} is used for any valid action $A_{valid,t}$.

The default values are a recursion limit of 5, a forced idle time of 2, and an episode length of 100. The influence of forced idle time is less significant. A long idle time slows down the training process as the agent is inactive for a longer time. Although the learning is even slightly faster than in the default setting for an idle time of 0 it is not chosen due to the fact that no time passes and, hence, the agent observes the same state over and over again, which showed in other experiments with more complex RL-agents a deteriorating performance.

Furthermore, one can see that for an increasing recursion limit the learning process is faster, as less forced idle actions are executed. However, the performance also shows a higher standard deviation. If the recursion is turned off, i.e. the value is set to 0, the training is clearly negatively influenced. For the episode length, a higher value obviously inhibits the training process, too. But it also reduces the variance due to the larger episode window that averages out short-term changes. On the other hand, a shorter episode length speeds up the training, but it is less robust. Hence, the default episode length is set to 100. Comparing the performance of an episode length of 10 and a recursion limit of 10, it can be concluded that both have a similar effect. This is due to the fact that limiting the episodes also acts somehow as recursion limit, because at every episode end, i.e. upon a terminal state, the policy update is performed.

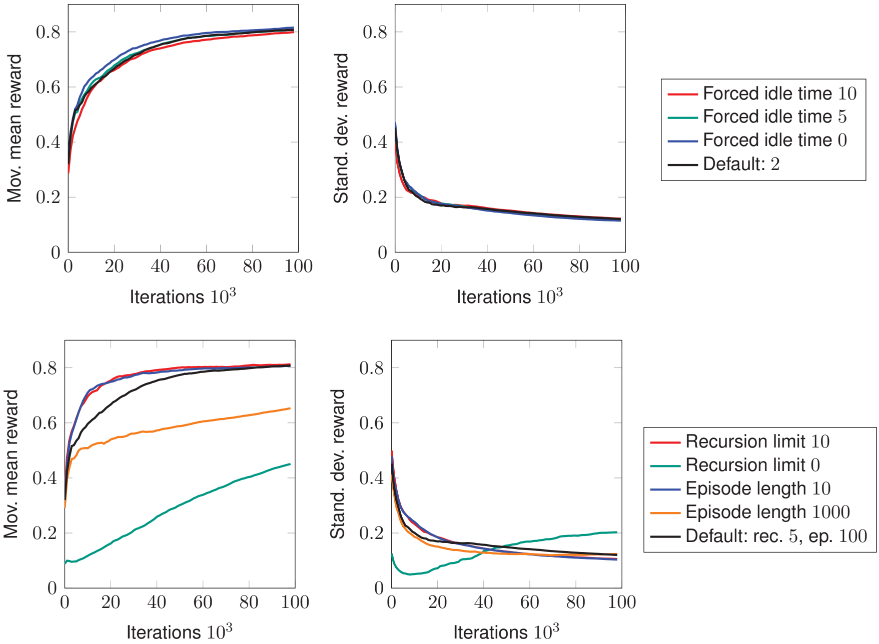


Figure A6.1: Varying recursion parameter, forced idle time, and episode length settings.

A7 Overview of feasible actions

Table A7.1: Numbering of 20 feasible actions for the action subsets $A_{S \rightarrow M}$ and $A_{M \rightarrow S}$.

	SO_1	SO_2	SO_3	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	SI_1	SI_2	SI_3
SO_1				1	2	3	4	5						
SO_2					6	7	8	9						
SO_3									10	11	12			
M_1												13		
M_2												14		
M_3													15	
M_4													16	
M_5													17	
M_6														18
M_7														19
M_8														20
SI_1														
SI_2														
SI_3														

A8 Correlation of performance indicators

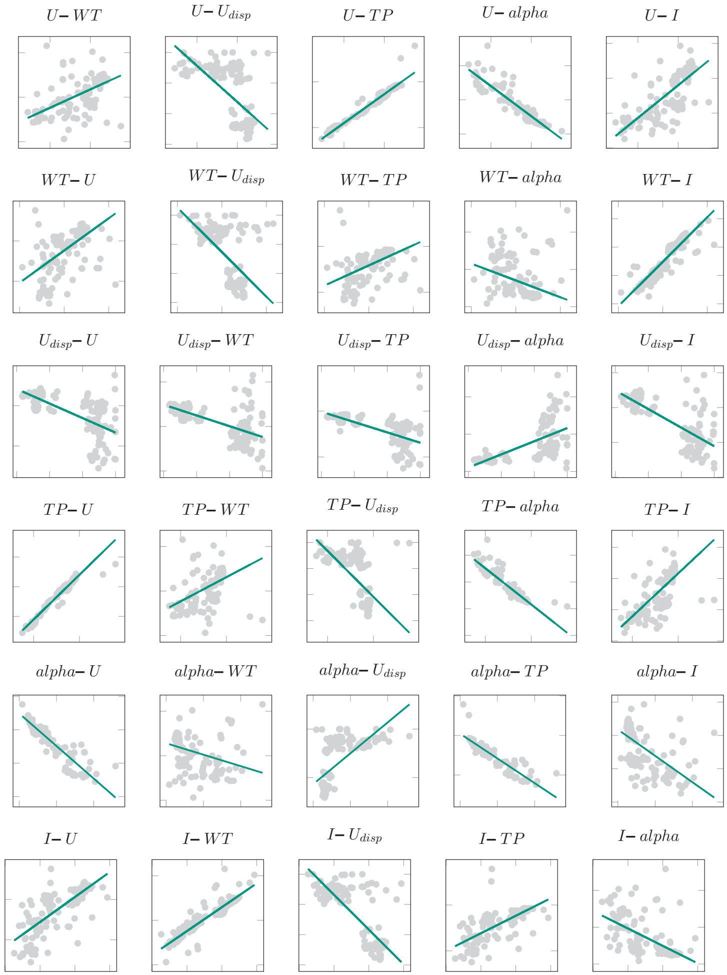


Figure A8.1: Correlation of performance indicators for all heuristics and RL-agents. The first term of the title corresponds to the values displayed on the x-axis and the second to the y-axis.

A9 Computational results of RL-agents for Scenario 2

Table A9.1: Mean performance results for RL-agents receiving state information S_{VA} and reward signal r_{const} with varying factors ω.

Agent	Reward and weight factor	Scenario 2				
	$(\omega_1 \widehat{=} A_{S\rightarrow M}, \omega_2 \widehat{=} A_{M\rightarrow S})$	$U[\%]$	$WT[\text{TU}]$	I	Share of $A_{invalid,t}[\%]$	Iteration of convergence
1	$r_{const}, \omega_1 = 1, \omega_2 = 0$	**85.4**	124.1	22.1	0.25	95,000
2	$r_{const}, \omega_1 = 0.5, \omega_2 = 0.5$	83.7	119.2	22.3	**0.07**	**33,667**
3	$r_{const}, \omega_1 = 0, \omega_2 = 1$	78.8	**107.3**	**17.6**	0.21	58,667

Table A9.2: Mean performance results for RL-agents receiving state information S_{VA} and sparse reward signal $r_{const,ep}$.

Agent	Scenario	Reward	$U[\%]$	$WT[\text{TU}]$	I	Iteration of convergence
39	1	$r_{const,ep}$	61.7	102.6	11.0	2,626,333
40	2	$r_{const,ep}$	84.1	126.1	23.6	204,667

Table A9.3: Mean performance results for RL-agents with varying state information and reward signals, aiming to optimize specific production performance indicators.

Agent	State	Reward	Scenario 2						
			$U[\%]$	σ_U	$WT[\text{TU}]$	σ_{WT}	I	σ_I	α-value
2	S_{VA}	r_{const}	**83.7**	**25.2**	119.2	145.2	20.3	**4.9**	**0.69**
4	S_{VA}, S_{AT}	r_{const}	82.6	25.7	119.1	143.5	20.2	5.1	**0.69**
5	S_{VA}, S_{WT}	r_{const}	78.7	27.5	**108.1**	**133.5**	**17.8**	5.2	0.83
6	S_{VA}	r_{util}	86.2	23.2	122.6	148.7	21.3	**5.0**	0.58
7	S_{VA}, S_{WT}	r_{util}	83.9	25.2	119.9	**143.7**	**20.5**	5.3	0.65
8	S_{VA}, S_{AT}	r_{util}	86.3	**22.9**	122.3	145.1	21.2	**5.0**	0.59
9	S_{VA}, S_{BEN}, S_{BEX}	r_{util}	**86.7**	23.5	**119.6**	145.8	20.8	**5.0**	**0.51**
10	S_{VA}	r_{wt}	**79.2**	**27.2**	111.0	135.9	18.7	**5.2**	**0.78**
11	S_{VA}, S_{WT}	r_{wt}	78.7	27.3	**106.6**	**133.5**	**17.6**	**5.2**	0.83

Table A9.4: Mean performance results for RL-agents with fixed state information and reward functions r_{util} and r_{wt} when varying the episode design.

Agent	State	Reward	Episode design	Scenario 2			
				$U[\%]$	$WT[\mathrm{TU}]$	I	α-*value*
12	$S_{VA}, S_{AT}, S_{BEN}, S_{BEX}$	r_{util}	100 $A_{valid,t}$	**87.1**	120.3	21.4	0.53
13			100 $A_{S\to M}$	86.6	**119.7**	**21.0**	**0.52**
14			100 $A_{M\to S}$	87.0	123.4	22.0	**0.52**
15			100 time steps	86.9	120.2	21.3	0.54
16	S_{VA}, S_{AT}, S_{WT}	r_{wt}	100 $A_{valid,t}$	77.6	**106.3**	17.5	**0.83**
17			100 $A_{S\to M}$	77.5	107.2	**17.3**	**0.83**
18			100 $A_{M\to S}$	78.2	106.7	17.9	**0.83**
19			100 time steps	**79.4**	109.3	18.2	**0.83**

Table A9.5: Mean performance results for RL-agents with fixed state information and reward signal when varying the action mapping as well as the action subsets the agent can execute.

Agent	State	Reward	Mapping	Action subsets	Scenario 2			
					$U[\%]$	$WT[\mathrm{TU}]$	I	α-*value*
12	$S_{VA}, S_{AT}, S_{BEN}, S_{BEX}$	r_{util}	direct	$A_{S\to M}, A_{M\to S}$	**87.1**	**120.3**	21.4	**0.53**
20			direct	$A_{S\to M}, A_{M\to S}, A_{empty}, A_{idle}$	87.0	120.5	**21.2**	0.54
21			resource	$A_{S\to M}, A_{M\to S}$	**85.0**	**113.0**	**18.9**	0.62
22			resource	$A_{S\to M}, A_{M\to S}, A_{empty}, A_{idle}$	**85.0**	115.4	19.2	**0.60**
16	S_{VA}, S_{AT}, S_{WT}	r_{wt}	direct	$A_{S\to M}, A_{M\to S}$	**77.6**	**106.3**	17.5	**0.83**
23			direct	$A_{S\to M}, A_{M\to S}, A_{empty}, A_{idle}$	77.1	106.4	**17.4**	0.89
24			resource	$A_{S\to M}, A_{M\to S}$	**78.6**	108.8	17.4	**0.89**
25			resource	$A_{S\to M}, A_{M\to S}, A_{empty}, A_{idle}$	77.5	**105.9**	**17.0**	0.94

Table A9.6: Mean performance results for RL-agents with the tree-based reward calculation and varying the tree evaluation function.

Agent	**State**	**Evaluation**	**Tree size**	**Scenario 2**			
				$U[\%]$	$WT[\text{TU}]$	I	α*-value*
26	$S_{VA}, S_{AT}, S_{BEN}, S_{BEX}$	Distance	1500	86.5	123.5	21.5	0.55
27		Utilization	1500	82.8	**115.6**	**19.6**	0.63
12		*Reference agent*		**87.1**	120.3	21.4	**0.53**

Table A9.7: Mean performance results for RL-agents with fixed state information and the reward functions $r_{w,util}$ and $r_{w,wt}$ while varying weight factors of the action subsets.

Agent	**State**	**Reward**	**Weight factor**	**Scenario 2**			
			$(\omega_1 \hat{=} A_{S\rightarrow M}, \omega_2 \hat{=} A_{M\rightarrow S})$	$U[\%]$	$WT[\text{TU}]$	I	α*-value*
28	$S_{VA}, S_{AT}, S_{BEN}, S_{BEX}, S_L, S_{MF}, S_{BPT}, S_{RPT}$	$r_{w,util}$	$\omega_1 = 0.25, \omega_2 = 0.75$	84.8	**109.5**	**18.0**	0.57
29			$\omega_1 = 0.5, \omega_2 = 0.5$	**86.9**	119.7	20.6	**0.47**
30			$\omega_1 = 0.75, \omega_2 = 0.25$	86.1	123.7	21.9	0.51
31	$S_{VA}, S_{AT}, S_{BEN}, S_{BEX}, S_L, S_{MF}, S_{WT}, S_{BPT}, S_{RPT}$	$r_{w,wt}$	$\omega_1 = 0.25, \omega_2 = 0.75$	78.7	**106.7**	**17.4**	0.86
32			$\omega_1 = 0.5, \omega_2 = 0.5$	78.1	108.1	17.8	0.76
33			$\omega_1 = 0.75, \omega_2 = 0.25$	**86.6**	125.9	22.6	**0.56**

Table A9.8: Mean performance results for RL-agents with fixed state information and a multi-objective reward function when varying weight factors of the multi-objective rewards.

Agent	**State**	**Reward**	**Scenario 2**			
			$U[\%]$	$WT[\text{TU}]$	I	α*-value*
34	$S_{VA}, S_{AT}, S_{BEN}, S_{BEX}, S_L, S_{MF}, S_{WT}, S_{BPT}, S_{RPT}$	$0.75 \cdot r_{util} + 0.25 \cdot r_{wt}$	**86.3**	118.8	20.5	**0.54**
35		$0.5 \cdot r_{util} + 0.5 \cdot r_{wt}$	86.0	118.6	20.3	0.58
36		$0.25 \cdot r_{util} + 0.75 \cdot r_{wt}$	82.5	**114.1**	**19.0**	0.72

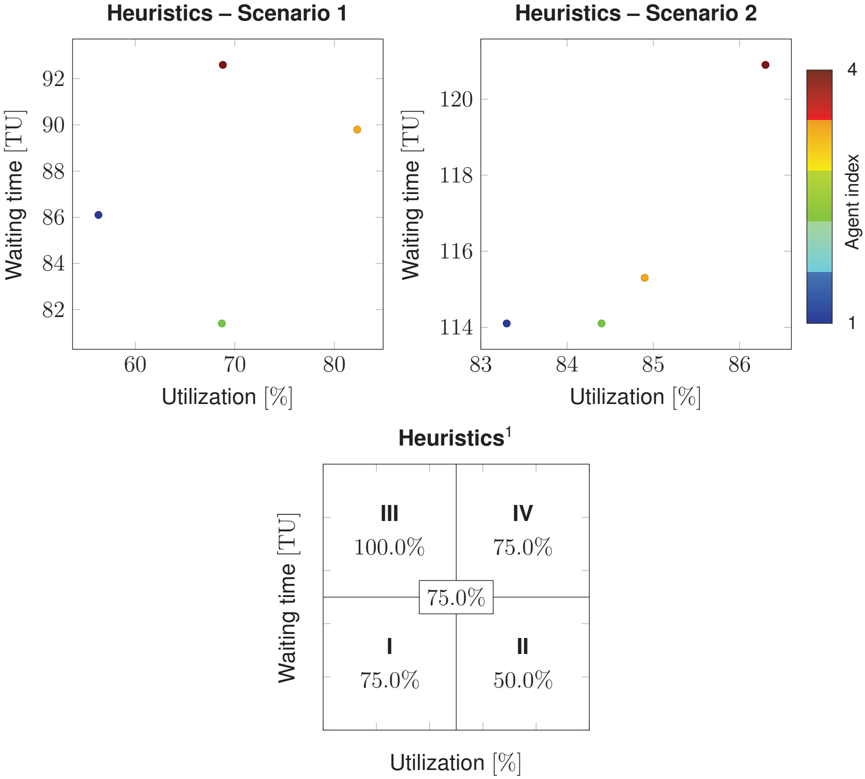


Figure A9.1: Comparison of benchmark heuristics in Scenario 1 and 2 (top). The colouring indicates the solution index to capture the association between one solution in both scenarios. The quadrants (bottom) depict the probability that a solution that is in a specific quadrant in the solution space of Scenario 1 is also in that same quadrant in Scenario 2. The limits of the quadrants are defined by the minimum and maximum value of the respective solution space.

[1] Due to the limited number of data points, i.e. just four benchmark heuristics, these results need to be interpreted with caution.

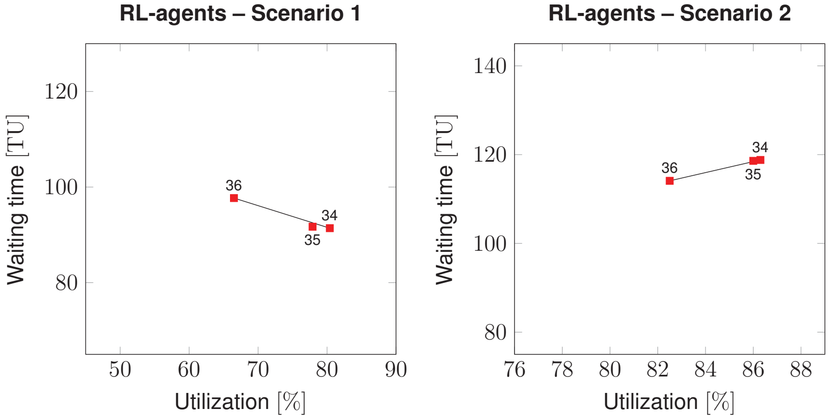


Figure A9.2: Straight line connection between Agents 34, 35, and 36 with multi-objective reward functions and varying weight factors for the rewards r_{util} and r_{wt}.

A10 Additional computational results

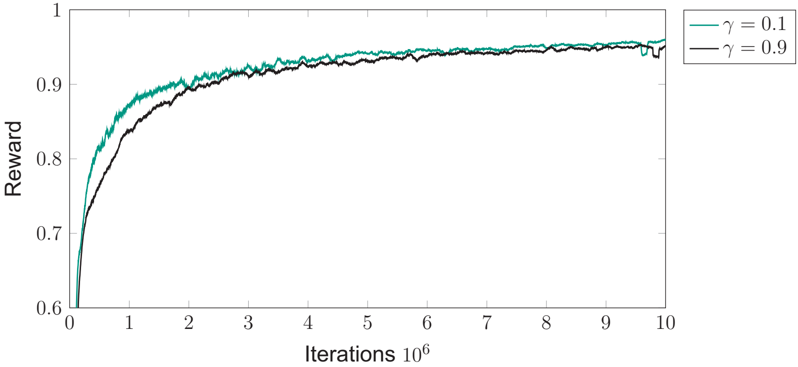


Figure A10.1: Comparing moving mean reward for Agents 37 and 38 imitating the NJF performance for two γ-values in Scenario 1.*

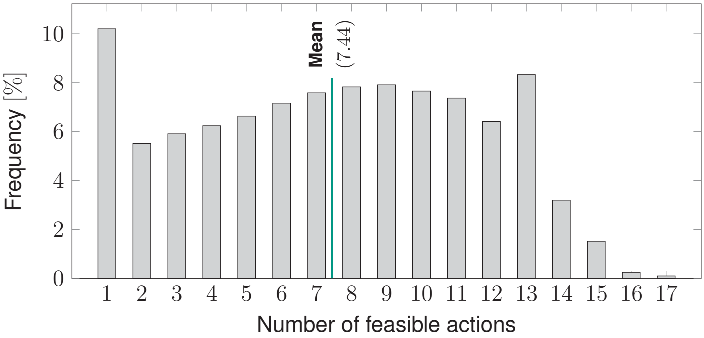


Figure A10.2: Frequency of the number of feasible actions at the decision point for a single simulation run of Agent 12 in Scenario 1, based on the last 500,000 actions.

Forschungsberichte aus dem wbk
Institut für Produktionstechnik
Karlsruher Institut für Technologie (KIT)

Bisher erschienene Bände:

Band 0
Dr.-Ing. Wu Hong-qi

Adaptive Volumenstromregelung mit Hilfe von drehzahlgeregelten Elektroantrieben

Band 1
Dr.-Ing. Heinrich Weiß

Fräsen mit Schneidkeramik - Verhalten des System Werkzeugmaschine-Werkzeug-Werkstück und Prozessanalyse

Band 2
Dr.-Ing. Hans-Jürgen Stierle

Entwicklung und Untersuchung hydrostatischer Lager für die Axialkolbenmaschine

Band 3
Dr.-Ing. Herbert Hörner

Untersuchung des Geräuschverhaltens druckgeregelter Axialkolbenpumpen

Band 4
Dr.-Ing. Rolf-Dieter Brückbauer

Digitale Drehzahlregelung unter der besonderen Berücksichtigung von Quantisierungseffekten

Band 5
Dr.-Ing. Gerhard Staiger

Graphisch interaktive NC-Programmierung von Drehteilen im Werkstattbereich

Band 6
Dr.-Ing. Karl Peters

Ein Beitrag zur Berechnung und Kompensation von Positionierfehlern an Industrierobotern

Band 7
Dr.-Ing. Paul Stauss

Automatisierte Inbetriebnahme und Sicherung der Zuverlässigkeit und Verfügbarkeit numerisch gesteuerter Fertigungseinrichtungen

Band 8
Dr.-Ing. Günter Möckesch

Konzeption und Realisierung eines strategischen, integrierten Gesamtplanungs- und -bearbeitungssystems zur Optimierung der Drehteilorganisation für auftragsbezogene Drehereien

Band 9
Dr.-Ing. Thomas Oestreicher

Rechnergestützte Projektierung von Steuerungen

Band 10
Dr.-Ing. Thomas Selinger

Teilautomatisierte werkstattnahe NC-Programmerstellung im Umfeld einer integrierten Informationsverarbeitung

Band 11
Dr.-Ing. Thomas Buchholz

Prozessmodell Fräsen, Rechnerunterstützte Analyse, Optimierung und Überwachung

Band 12
Dr.-Ing. Bernhard Reichling

Lasergestützte Positions- und Bahnvermessung von Industrierobotern

Band 13
Dr.-Ing. Hans-Jürgen Lesser

Rechnergestützte Methoden zur Auswahl anforderungsgerechter Verbindungselemente

Band 14
Dr.-Ing. Hans-Jürgen Lauffer

Einsatz von Prozessmodellen zur rechnerunterstützten Auslegung von Räumwerkzeugen

Band 15
Dr.-Ing. Michael C. Wilhelm

Rechnergestützte Prüfplanung im Informationsverbund moderner Produktionssysteme

Band 16
Dr.-Ing. Martin Ochs

Entwurf eines Programmsystems zur wissensbasierten Planung und Konfigurierung

Band 17
Dr.-Ing. Heinz-Joachim Schneider

Erhöhung der Verfügbarkeit von hochautomatisierten Produktionseinrichtungen mit Hilfe der Fertigungsleittechnik

Band 18
Dr.-Ing. Hans-Reiner Ludwig

Beanspruchungsanalyse der Werkzeugschneiden beim Stirnplanfräsen

Band 19
Dr.-Ing. Rudolf Wieser

Methoden zur rechnergestützten Konfigurierung von Fertigungsanlagen

Band 20
Dr.-Ing. Edgar Schmitt

Werkstattsteuerung bei wechselnder Auftragsstruktur

Band 21
Dr.-Ing. Wilhelm Enderle

Verfügbarkeitssteigerung automatisierter Montagesysteme durch selbsttätige Behebung prozessbedingter Störungen

Band 22
Dr.-Ing. Dieter Buchberger

Rechnergestützte Strukturplanung von Produktionssystemen

Band 23
Prof. Dr.-Ing. Jürgen Fleischer

Rechnerunterstützte Technologieplanung für die flexibel automatisierte Fertigung von Abkantteilen

Band 24
Dr.-Ing. Lukas Loeffler

Adaptierbare und adaptive Benutzerschnittstellen

Band 25
Dr.-Ing. Thomas Friedmann

Integration von Produktentwicklung und Montageplanung durch neue rechnergestützte Verfahren

Band 26
Dr.-Ing. Robert Zurrin

Variables Formhonen durch rechnergestützte Hornprozesssteuerung

Band 27
Dr.-Ing. Karl-Heinz Bergen

Langhub-Innenrundhonen von Grauguss und Stahl mit einem elektromechanischem Vorschubsystem

Band 28
Dr.-Ing. Andreas Liebisch

Einflüsse des Festwalzens auf die Eigenspannungsverteilung und die Dauerfestigkeit einsatzgehärteter Zahnräder

Band 29
Dr.-Ing. Rolf Ziegler

Auslegung und Optimierung schneller Servopumpen

Band 30
Dr.-Ing. Rainer Bartl

Datenmodellgestützte Wissensverarbeitung zur Diagnose und Informationsunterstützung in technischen Systemen

Band 31
Dr.-Ing. Ulrich Golz

Analyse, Modellbildung und Optimierung des Betriebsverhaltens von Kugelgewindetrieben

Band 32
Dr.-Ing. Stephan Timmermann

Automatisierung der Feinbearbeitung in der Fertigung von Hohlformwerkzeugen

Band 33
Dr.-Ing. Thomas Noe

Rechnergestützter Wissenserwerb zur Erstellung von Überwachungs- und Diagnoseexpertensystemen für hydraulische Anlagen

Band 34
Dr.-Ing. Ralf Lenschow

Rechnerintegrierte Erstellung und Verifikation von Steuerungsprogrammen als Komponente einer durchgängigen Planungsmethodik

Band 35
Dr.-Ing. Matthias Kallabis

Räumen gehärteter Werkstoffe mit kristallinen Hartstoffen

Band 36
Dr.-Ing. Heiner-Michael Honeck

Rückführung von Fertigungsdaten zur Unterstützung einer fertigungsgerechten Konstruktion

Band 37
Dr.-Ing. Manfred Rohr

Automatisierte Technologieplanung am Beispiel der Komplettbearbeitung auf Dreh-/Fräszellen

Band 38
Dr.-Ing. Martin Steuer

Entwicklung von Softwarewerkzeugen zur wissensbasierten Inbetriebnahme von komplexen Serienmaschinen

Band 39
Dr.-Ing. Siegfried Beichter

Rechnergestützte technische Problemlösung bei der Angebotserstellung von flexiblen Drehzellen

Band 40
Dr.-Ing. Thomas Steitz

Methodik zur marktorientierten Entwicklung von Werkzeugmaschinen mit Integration von funktionsbasierter Strukturierung und Kostenschätzung

Band 41
Dr.-Ing. Michael Richter

Wissensbasierte Projektierung elektrohydraulischer Regelungen

Band 42
Dr.-Ing. Roman Kuhn

Technologieplanungssystem Fräsen. Wissensbasierte Auswahl von Werkzeugen, Schneidkörpern und Schnittbedingungen für das Fertigingsverfahren Fräsen

Band 43
Dr.-Ing. Hubert Klein

Rechnerunterstützte Qualitätssicherung bei der Produktion von Bauteilen mit frei geformten Oberflächen

Band 44
Dr.-Ing. Christian Hoffmann

Konzeption und Realisierung eines fertigungsintegrierten Koordinatenmessgerätes

Band 45
Dr.-Ing. Volker Frey

Planung der Leittechnik für flexible Fertigungsanlagen

Band 46
Dr.-Ing. Achim Feller

Kalkulation in der Angebotsphase mit dem selbsttätig abgeleiteten Erfahrungswissen der Arbeitsplanung

Band 47
Dr.-Ing. Markus Klaiber

Produktivitätssteigerung durch rechnerunterstütztes Einfahren von NC-Programmen

Band 48
Dr.-Ing. Roland Minges

Verbesserung der Genauigkeit beim fünfachsigen Fräsen von Freiformflächen

Band 49
Dr.-Ing. Wolfgang Bernhart

Beitrag zur Bewertung von Montagevarianten: Rechnergestützte Hilfsmittel zur kostenorientierten, parallelen Entwicklung von Produkt und Montagesystem

Band 50
Dr.-Ing. Peter Ganghoff

Wissensbasierte Unterstützung der Planung technischer Systeme: Konzeption eines Planungswerkzeuges und exemplarische Anwendung im Bereich der Montagesystemplanung

Band 51
Dr.-Ing. Frank Maier

Rechnergestützte Prozessregelung beim flexiblen Gesenkbiegen durch Rückführung von Qualitätsinformationen

Band 52
Dr.-Ing. Frank Debus

Ansatz eines rechnerunterstützten Planungsmanagements für die Planung in verteilten Strukturen

Band 53
Dr.-Ing. Joachim Weinbrecht

Ein Verfahren zur zielorientierten Reaktion auf Planabweichungen in der Werkstattregelung

Band 54
Dr.-Ing. Gerd Herrmann

Reduzierung des Entwicklungsaufwandes für anwendungsspezifische Zellenrechnersoftware durch Rechnerunterstützung

Band 55
Dr.-Ing. Robert Wassmer

Verschleissentwicklung im tribologischen System Fräsen: Beiträge zur Methodik der Prozessmodellierung auf der Basis tribologisher Untersuchungen beim Fräsen

Band 56
Dr.-Ing. Peter Uebelhoer

Inprocess-Geometriemessung beim Honen

Band 57
Dr.-Ing. Hans-Joachim Schelberg

Objektorientierte Projektierung von SPS-Software

Band 58
Dr.-Ing. Klaus Boes

Integration der Qualitätsentwicklung in featurebasierte CAD/CAM-Prozessketten

Band 59
Dr.-Ing. Martin Schreiber

Wirtschaftliche Investitionsbewertung komplexer Produktionssysteme unter Berücksichtigung von Unsicherheit

Band 60
Dr.-Ing. Ralf Steuernagel

Offenes adaptives Engineering-Werkzeug zur automatisierten Erstellung von entscheidungsunterstützenden Informationssystemen

Band 62
Dr.-Ing. Uwe Schauer

Qualitätsorientierte Feinbearbeitung mit Industrierobotern: Regelungsansatz für die Freiformflächenfertigung des Werkzeug- und Formenbaus

Band 63
Dr.-Ing. Simone Loeper

Kennzahlengestütztes Beratungssystem zur Verbesserung der Logistikleistung in der Werkstattfertigung

Band 64
Dr.-Ing. Achim Raab

Räumen mit hartstoffbeschichteten HSS-Werkzeugen

Band 65,
Dr.-Ing. Jan Erik Burghardt

Unterstützung der NC-Verfahrenskette durch ein bearbeitungselementorientiertes, lernfähiges Technologieplanungssystem

Band 66
Dr.-Ing. Christian Tritsch

Flexible Demontage technischer Gebrauchsgüter: Ansatz zur Planung und (teil-)automatisierten Durchführung industireller Demontageprozesse

Band 67
Dr.-Ing. Oliver Eitrich

Prozessorientiertes Kostenmodell für die entwicklungsbegleitende Vorkalkulation

Band 68
Dr.-Ing. Oliver Wilke

Optimierte Antriebskonzepte für Räummaschinen - Potentiale zur Leistungssteigerung

Band 69
Dr.-Ing. Thilo Sieth

Rechnergestützte Modellierungsmethodik zerspantechnologischer Prozesse

Band 70
Dr.-Ing. Jan Linnenbuerger

Entwicklung neuer Verfahren zur automatisierten Erfassung der geometrischen Abweichungen an Linearachsen und Drehschwenkköpfen

Band 71
Dr.-Ing. Mathias Klimmek

Fraktionierung technischer Produkte mittels eines frei beweglichen Wasserstrahlwerkzeuges

Band 72
Dr.-Ing. Marko Hartel

Kennzahlenbasiertes Bewertungssystem zur Beurteilung der Demontage- und Recyclingeignung von Produkten

Band 73
Dr.-Ing. Jörg Schaupp

Wechselwirkung zwischen der Maschinen- und Hauptspindelantriebsdynamik und dem Zerspanprozess beim Fräsen

Band 74
Dr.-Ing. Bernhard Neisius

Konzeption und Realisierung eines experimentellen Telemanipulators für die Laparoskopie

Band 75
Dr.-Ing. Wolfgang Walter

Erfolgsversprechende Muster für betriebliche Ideenfindungsprozesse. Ein Beitrag zur Steigerung der Innovationsfähigkeit

Band 76
Dr.-Ing. Julian Weber

Ein Ansatz zur Bewertung von Entwicklungsergebnissen in virtuellen Szenarien

Band 77
Dr.-Ing. Dipl. Wirtsch.-Ing. Markus Posur

Unterstützung der Auftragsdurchsetzung in der Fertigung durch Kommunikation über mobile Rechner

Band 78
Dr.-Ing. Frank Fleissner

Prozessorientierte Prüfplanung auf Basis von Bearbeitungsobjekten für die Kleinserienfertigung am Beispiel der Bohr- und Fräsbearbeitung

Band 79
Dr.-Ing. Anton Haberkern

Leistungsfähigere Kugelgewindetriebe durch Beschichtung

Band 80
Dr.-Ing. Dominik Matt

Objektorientierte Prozess- und Strukturinnovation (OPUS)

Band 81
Dr.-Ing. Jürgen Andres

Robotersysteme für den Wohnungsbau: Beitrag zur Automatisierung des Mauerwerkabaus und der Elektroinstallation auf Baustellen

Band 82
Dr.-Ing. Dipl.Wirtschaftsing. Simone Riedmiller

Der Prozesskalender - Eine Methodik zur marktorientierten Entwicklung von Prozessen

Band 83
Dr.-Ing. Dietmar Tilch

Analyse der Geometrieparameter von Präzisionsgewinden auf der Basis einer Least-Squares-Estimation

Band 84
Dr.-Ing. Dipl.-Kfm. Oliver Stiefbold

Konzeption eines reaktionsschnellen Planungssystems für Logistikketten auf Basis von Software-Agenten

Band 85
Dr.-Ing. Ulrich Walter

Einfluss von Kühlschmierstoff auf den Zerspanprozess beim Fräsen: Beitrag zum Prozessverständniss auf Basis von zerspantechnischen Untersuchungen

Band 86
Dr.-Ing. Bernd Werner

Konzeption von teilautonomer Gruppenarbeit unter Berücksichtigung kultureller Einflüsse

Band 87
Dr.-Ing. Ulf Osmers

Projektieren Speicherprogrammierbarer Steuerungen mit Virtual Reality

Band 88
Dr.-Ing. Oliver Doerfel

Optimierung der Zerspantechnik beim Fertigungsverfahren Wälzstossen: Analyse des Potentials zur Trockenbearbeitung

Band 89
Dr.-Ing. Peter Baumgartner

Stufenmethode zur Schnittstellengestaltung in der internationalen Produktion

Band 90
Dr.-Ing. Dirk Vossmann

Wissensmanagement in der Produktentwicklung durch Qualitätsmethodenverbund und Qualitätsmethodenintegration

Band 91
Dr.-Ing. Martin Plass

Beitrag zur Optimierung des Honprozesses durch den Aufbau einer Honprozessregelung

Band 92
Dr.-Ing. Titus Konold

Optimierung der Fünfachsfräsbearbeitung durch eine kennzahlenunterstützte CAM-Umgebung

Band 93
Dr.-Ing. Jürgen Brath

Unterstützung der Produktionsplanung in der Halbleiterfertigung durch risikoberücksichtigende Betriebskennlinien

Band 94
Dr.-Ing. Dirk Geisinger

Ein Konzept zur marktorientierten Produktentwicklung

Band 95
Dr.-Ing. Marco Lanza

Entwurf der Systemunterstützung des verteilten Engineering mit Axiomatic Design

Band 96
Dr.-Ing. Volker Hüntrup

Untersuchungen zur Mikrostrukturierbarkeit von Stählen durch das Fertigungsverfahren Fräsen

Band 97
Dr.-Ing. Frank Reinboth

Interne Stützung zur Genauigkeitsverbesserung in der Inertialmesstechnik: Beitrag zur Senkung der Anforderungen an Inertialsensoren

Band 98
Dr.-Ing. Lutz Trender

Entwicklungsintegrierte Kalkulation von Produktlebenszykluskosten auf Basis der ressourcenorientierten Prozesskostenrechnung

Band 99
Dr.-Ing. Cornelia Kafka

Konzeption und Umsetzung eines Leitfadens zum industriellen Einsatz von Data-Mining

Band 100
Dr.-Ing. Gebhard Selinger

Rechnerunterstützung der informellen Kommunikation in verteilten Unternehmensstrukturen

Band 101
Dr.-Ing. Thomas Windmüller

Verbesserung bestehender Geschäftsprozesse durch eine mitarbeiterorientierte Informationsversorgung

Band 102
Dr.-Ing. Knud Lembke

Theoretische und experimentelle Untersuchung eines bistabilen elektrohydraulischen Linearantriebs

Band 103
Dr.-Ing. Ulrich Thies

Methode zur Unterstützung der variantengerechten Konstruktion von industriell eingesetzten Kleingeräten

Band 104
Dr.-Ing. Andreas Schmälzle

Bewertungssystem für die Generalüberholung von Montageanlagen –Ein Beitrag zur wirtschaftlichen Gestaltung geschlossener Facility- Managment-Systeme im Anlagenbau

Band 105
Dr.-Ing. Thorsten Frank

Vergleichende Untersuchungen schneller elektromechanischer Vorschubachsen mit Kugelgewindetrieb

Band 106
Dr.-Ing. Achim Agostini

Reihenfolgeplanung unter Berücksichtigung von Interaktionen: Beitrag zur ganzheitlichen Strukturierung und Verarbeitung von Interaktionen von Bearbeitungsobjekten

Band 107
Dr.-Ing. Thomas Barrho

Flexible, zeitfenstergesteuerte Auftragseinplanung in segmentierten Fertigungsstrukturen

Band 108
Dr.-Ing. Michael Scharer

Quality Gate-Ansatz mit integriertem Risikomanagement

Band 109
Dr.-Ing. Ulrich Suchy

Entwicklung und Untersuchung eines neuartigen Mischkopfes für das Wasser Abrasivstrahlschneiden

Band 110
Dr.-Ing. Sellal Mussa

Aktive Korrektur von Verlagerungsfehlern in Werkzeugmaschinen

Band 111
Dr.-Ing. Andreas Hühsam

Modellbildung und experimentelle Untersuchung des Wälzschälprozesses

Band 112
Dr.-Ing. Axel Plutowsky

Charakterisierung eines optischen Messsystems und den Bedingungen des Arbeitsraums einer Werkzeugmaschine

Band 113
Dr.-Ing. Robert Landwehr

Konsequent dezentralisierte Steuerung mit Industrial Ethernet und offenen Applikationsprotokollen

Band 114
Dr.-Ing. Christoph Dill

Turbulenzreaktionsprozesse

Band 115
Dr.-Ing. Michael Baumeister

Fabrikplanung im turbulenten Umfeld

Band 116
Dr.-Ing. Christoph Gönnheimer

Konzept zur Verbesserung der Elektromagnetischen Verträglichkeit (EMV) in Produktionssystemen durch intelligente Sensor/Aktor-Anbindung

Band 117
Dr.-Ing. Lutz Demuß

Ein Reifemodell für die Bewertung und Entwicklung von Dienstleistungsorganisationen: Das Service Management Maturity Modell (SMMM)

Band 118
Dr.-Ing. Jörg Söhner

Beitrag zur Simulation zerspanungstechnologischer Vorgänge mit Hilfe der Finite-Element-Methode

Band 119
Dr.-Ing. Judith Elsner

Informationsmanagement für mehrstufige Mikro-Fertigungsprozesse

Band 120
Dr.-Ing. Lijing Xie

Estimation Of Two-dimension Tool Wear Based On Finite Element Method

Band 121
Dr.-Ing. Ansgar Blessing

Geometrischer Entwurf mikromechatronischer Systeme

Band 122
Dr.-Ing. Rainer Ebner

Steigerung der Effizienz mehrachsiger Fräsprozesse durch neue Planungsmethoden mit hoher Benutzerunterstützung

Band 123
Dr.-Ing. Silja Klinkel

Multikriterielle Feinplanung in teilautonomen Produktionsbereichen – Ein Beitrag zur produkt- und prozessorientierten Planung und Steuerung

Band 124
Dr.-Ing. Wolfgang Neithardt

Methodik zur Simulation und Optimierung von Werkzeugmaschinen in der Konzept- und Entwurfsphase auf Basis der Mehrkörpersimulation

Band 125
Dr.-Ing. Andreas Mehr

Hartfeinbearbeitung von Verzahnungen mit kristallinen diamantbeschichteten Werkzeugen beim Fertigungsverfahren Wälzstoßen

Band 126
Dr.-Ing. Martin Gutmann

Entwicklung einer methodischen Vorgehensweise zur Diagnose von hydraulischen Produktionsmaschinen

Band 127
Dr.-Ing. Gisela Lanza

Simulative Anlaufunterstützung auf Basis der Qualitätsfähigkeiten von Produktionsprozessen

Band 128
Dr.-Ing. Ulf Dambacher

Kugelgewindetrieb mit hohem Druckwinkel

Band 129
Dr.-Ing. Carsten Buchholz

Systematische Konzeption und Aufbau einer automatisierten Produktionszelle für pulverspritzgegossene Mikrobauteile

Band 130
Dr.-Ing. Heiner Lang

Trocken-Räumen mit hohen Schnittgeschwindigkeiten

Band 131
Dr.-Ing. Daniel Nesges

Prognose operationeller Verfügbarkeiten von Werkzeugmaschinen unter Berücksichtigung von Serviceleistungen

Im Shaker Verlag erschienene Bände:

Band 132
Dr.-Ing. Andreas Bechle

Beitrag zur prozesssicheren Bearbeitung beim Hochleistungsfertigungsverfahren Wälzschälen

Band 133
Dr.-Ing. Markus Herm

Konfiguration globaler Wertschöpfungsnetzwerke auf Basis von Business Capabilities

Band 134
Dr.-Ing. Hanno Tritschler

Werkzeug- und Zerspanprozessoptimierung beim Hartfräsen von Mikrostrukturen in Stahl

Band 135
Dr.-Ing. Christian Munzinger

Adaptronische Strebe zur Steifigkeitssteigerung von Werkzeugmaschinen

Band 136
Dr.-Ing. Andreas Stepping

Fabrikplanung im Umfeld von Wertschöpfungsnetzwerken und ganzheitlichen Produktionssystemen

Band 137
Dr.-Ing. Martin Dyck

Beitrag zur Analyse thermische bedingter Werkstückdeformationen in Trockenbearbeitungsprozessen

Band 138
Dr.-Ing. Siegfried Schmalzried

Dreidimensionales optisches Messsystem für eine effizientere geometrische Maschinenbeurteilung

Band 139
Dr.-Ing. Marc Wawerla

Risikomanagement von Garantieleistungen

Band 140
Dr.-Ing. Ivesa Buchholz

Strategien zur Qualitätssicherung mikromechanischer Bauteile mittels multisensorieller Koordinatenmesstechnik

Band 141
Dr.-Ing. Jan Kotschenreuther

Empirische Erweiterung von Modellen der Makrozerspanung auf den Bereich der Mikrobearbeitung

Band 142
Dr.-Ing. Andreas Knödel

Adaptronische hydrostatische Drucktascheneinheit

Band 143
Dr.-Ing. Gregor Stengel

Fliegendes Abtrennen räumlich gekrümmter Strangpressprofile mittels Industrierobotern

Band 144
Dr.-Ing. Udo Weismann

Lebenszyklusorientiertes interorganisationelles Anlagencontrolling

Band 145
Dr.-Ing. Rüdiger Pabst

Mathematische Modellierung der Wärmestromdichte zur Simulation des thermischen Bauteilverhaltens bei der Trockenbearbeitung

Band 146
Dr.-Ing. Jan Wieser

Intelligente Instandhaltung zur Verfügbarkeitssteigerung von Werkzeugmaschinen

Band 147
Dr.-Ing. Sebastian Haupt

Effiziente und kostenoptimale Herstellung von Mikrostrukturen durch eine Verfahrenskombination von Bahnerosion und Laserablation

Band 148
Dr.-Ing. Matthias Schlipf

Statistische Prozessregelung von Fertigungs- und Messprozess zur Erreichung einer variabilitätsarmen Produktion mikromechanischer Bauteile

Band 149
Dr.-Ing. Jan Philipp Schmidt-Ewig

Methodische Erarbeitung und Umsetzung eines neuartigen Maschinenkonzeptes zur produktflexiblen Bearbeitung räumlich gekrümmter Strangpressprofile

Band 150
Dr.-Ing. Thomas Ender

Prognose von Personalbedarfen im Produktionsanlauf unter Berücksichtigung dynamischer Planungsgrößen

Band 151
Dr.-Ing. Kathrin Peter

Bewertung und Optimierung der Effektivität von Lean Methoden in der Kleinserienproduktion

Band 152
Dr.-Ing. Matthias Schopp

Sensorbasierte Zustandsdiagnose und -prognose von Kugelgewindetrieben

Band 153
Dr.-Ing. Martin Kipfmüller

Aufwandsoptimierte Simulation von Werkzeugmaschinen

Band 154
Dr.-Ing. Carsten Schmidt

Development of a database to consider multi wear mechanisms within chip forming simulation

Band 155
Dr.-Ing. Stephan Niggeschmidt

Ausfallgerechte Ersatzteilbereitstellung im Maschinen- und Anlagenbau mittels lastabhängiger Lebensdauerprognose

Band 156
Dr.-Ing. Jochen Conrad Peters

Bewertung des Einflusses von Formabweichungen in der Mikro-Koordinatenmesstechnik

Band 157
Dr.-Ing. Jörg Ude

Entscheidungsunterstützung für die Konfiguration globaler Wertschöpfungsnetzwerke

Band 158
Dr.-Ing. Stefan Weiler

Strategien zur wirtschaftlichen Gestaltung der globalen Beschaffung

Band 159
Dr.-Ing. Jan Rühl

Monetäre Flexibilitäts- und Risikobewertung

Band 160
Dr.-Ing. Daniel Ruch

Positions- und Konturerfassung räumlich gekrümmter Profile auf Basis bauteilimmanenter Markierungen

Band 161
Dr.-Ing. Manuel Tröndle

Flexible Zuführung von Mikrobauteilen mit piezoelektrischen Schwingförderern

Band 162
Dr.-Ing. Benjamin Viering

Mikroverzahnungsnormal

Band 163
Dr.-Ing. Chris Becke

Prozesskraftrichtungsangepasste Frässtrategien zur schädigungsarmen Bohrungsbearbeitung an faserverstärkten Kunststoffen

Band 164
Dr.-Ing. Patrick Werner

Dynamische Optimierung und Unsicherheitsbewertung der lastabhängigen präventiven Instandhaltung von Maschinenkomponenten

Band 165
Dr.-Ing. Martin Weis

Kompensation systematischer Fehler bei Werkzeugmaschinen durch self-sensing Aktoren

Band 166
Dr.-Ing. Markus Schneider

Kompensation von Konturabweichungen bei gerundeten Strangpressprofilen durch robotergestützte Führungswerkzeuge

Band 167
Dr.-Ing. Ester M. R. Ruprecht

Prozesskette zur Herstellung schichtbasierter Systeme mit integrierten Kavitäten

Band 168
Dr.-Ing. Alexander Broos

Simulationsgestützte Ermittlung der Komponentenbelastung für die Lebensdauerprognose an Werkzeugmaschinen

Band 169
Dr.-Ing. Frederik Zanger

Segmentspanbildung, Werkzeugverschleiß, Randschichtzustand und Bauteileigenschaften: Numerische Analysen zur Optimierung des Zerspanungsprozesses am Beispiel von Ti-6Al-4V

Band 170
Dr.-Ing. Benjamin Behmann

Servicefähigkeit

Band 171
Dr.-Ing. Annabel Gabriele Jondral

Simulationsgestützte Optimierung und Wirtschaftlichkeitsbewertung des Lean-Methodeneinsatzes

Band 172
Dr.-Ing. Christoph Ruhs

Automatisierte Prozessabfolge zur qualitätssicheren Herstellung von Kavitäten mittels Mikrobahnerosion

Band 173
Dr.-Ing. Steven Peters

Markoffsche Entscheidungsprozesse zur Kapazitäts- und Investitionsplanung von Produktionssystemen

Band 174
Dr.-Ing. Christoph Kühlewein

Untersuchung und Optimierung des Wälzschälverfahrens mit Hilfe von 3D-FEM-Simulation – 3D-FEM Kinematik- und Spanbildungssimulation

Band 175
Dr.-Ing. Adam-Mwanga Dieckmann

Auslegung und Fertigungsprozessgestaltung sintergefügter Verbindungen für µMIM-Bauteile

Band 176
Dr.-Ing. Heiko Hennrich

Aufbau eines kombinierten belastungs- und zustandsorientierten Diagnose- und Prognosesystems für Kugelgewindetriebe

Band 177
Dr.-Ing. Stefan Herder

Piezoelektrischer Self-Sensing-Aktor zur Vorspannungsregelung in adaptronischen Kugelgewindetrieben

Band 178
Dr.-Ing. Alexander Ochs

Ultraschall-Strömungsgreifer für die Handhabung textiler Halbzeuge bei der automatisierten Fertigung von RTM-Bauteilen

Band 179
Dr.-Ing. Jürgen Michna

Numerische und experimentelle Untersuchung zerspanungsbedingter Gefügeumwandlungen und Modellierung des thermo-mechanischen Lastkollektivs beim Bohren von 42CrMo4

Band 180
Dr.-Ing. Jörg Elser

Vorrichtungsfreie räumliche Anordnung von Fügepartnern auf Basis von Bauteilmarkierungen

Band 181
Dr.-Ing. Katharina Klimscha

Einfluss des Fügespalts auf die erreichbare Verbindungsqualität beim Sinterfügen

Band 182
Dr.-Ing. Patricia Weber

Steigerung der Prozesswiederholbarkeit mittels Analyse akustischer Emissionen bei der Mikrolaserablation mit UV-Pikosekundenlasern

Band 183
Dr.-Ing. Jochen Schädel

Automatisiertes Fügen von Tragprofilen mittels Faserwickeln

Band 184
Dr.-Ing. Martin Krauße

Aufwandsoptimierte Simulation von Produktionsanlagen durch Vergrößerung der Geltungsbereiche von Teilmodellen

Band 185
Dr.-Ing. Raphael Moser

Strategische Planung globaler Produktionsnetzwerke
Bestimmung von Wandlungsbedarf und Wandlungszeitpunkt mittels multikriterieller Optimierung

Band 186
Dr.-Ing. Martin Otter

Methode zur Kompensation fertigungsbedingter Gestaltabweichungen für die Montage von Aluminium Space-Frame-Strukturen

Band 187
Dr.-Ing. Urs Leberle

Produktive und flexible Gleitförderung kleiner Bauteile auf phasenflexiblen Schwingförderern mit piezoelektrischen 2D-Antriebselementen

Band 188
Dr.-Ing. Johannes Book

Modellierung und Bewertung von Qualitätsmanagementstrategien in globalen Wertschöpfungsnetzwerken

Band 189
Dr.-Ing. Florian Ambrosy

Optimierung von Zerspanungsprozessen zur prozesssicheren Fertigung nanokristalliner Randschichten am Beispiel von 42CrMo4

Band 190
Dr.-Ing. Adrian Kölmel

Integrierte Messtechnik für Prozessketten unreifer Technologien am Beispiel der Batterieproduktion für Elektrofahrzeuge

Band 191
Dr.-Ing. Henning Wagner

Featurebasierte Technologieplanung zum Preforming von textilen Halbzeugen

Band 192
Dr.-Ing. Johannes Gebhardt

Strukturoptimierung von in FVK eingebetteten metallischen Lasteinleitungselementen

Band 193
Dr.-Ing. Jörg Bauer

Hochintegriertes hydraulisches Vorschubsystem für die Bearbeitung kleiner Werkstücke mit hohen Fertigungsanforderungen

Band 194
Dr.-Ing. Nicole Stricker

Robustheit verketteter Produktionssysteme
Robustheitsevaluation und Selektion des Kennzahlensystems der Robustheit

Band 195
Dr.-Ing. Anna Sauer

Konfiguration von Montagelinien unreifer Produkttechnologien am Beispiel der Batteriemontage für Elektrofahrzeuge

Band 196
Dr.-Ing. Florian Sell-Le Blanc

Prozessmodell für das Linearwickeln unrunder Zahnspulen
Ein Beitrag zur orthozyklischen Spulenwickeltechnik

Band 197
Dr.-Ing. Frederic Förster

Geregeltes Handhabungssystem zum zuverlässigen und energieeffizienten Handling textiler Kohlenstofffaserzuschnitte

Band 198
Dr.-Ing. Nikolay Boev

Numerische Beschreibung von Wechselwirkungen zwischen Zerspanprozess und Maschine am Beispiel Räumen

Band 199
Dr.-Ing. Sebastian Greinacher

Simulationsgestützte Mehrzieloptimierung schlanker und ressourceneffizienter Produktionssysteme

Band 200
Dr.-Ing. Benjamin Häfner

Lebensdauerprognose in Abhängigkeit der Fertigungsabweichungen bei Mikroverzahnungen

Band 201
Dr.-Ing. Stefan Klotz

Dynamische Parameteranpassung bei der Bohrungsherstellung in faserverstärkten Kunststoffen unter zusätzlicher Berücksichtigung der Einspannsituation

Band 202
Dr.-Ing. Johannes Stoll

Bewertung konkurrierender Fertigungsfolgen mittels Kostensimulation und stochastischer Mehrzieloptimierung
Anwendung am Beispiel der Blechpaketfertigung für automobile Elektromotoren

Band 203
Dr.-Ing. Simon-Frederik Koch

Fügen von Metall-Faserverbund-Hybridwellen im Schleuderverfahren
ein Beitrag zur fertigungsgerechten intrinsischen Hybridisierung

Band 204
Dr.-Ing. Julius Ficht

Numerische Untersuchung der Eigenspannungsentwicklung für sequenzielle Zerspanungsprozesse

Band 205
Dr.-Ing. Manuel Baumeister

Automatisierte Fertigung von Einzelblattstapeln in der Lithium-Ionen-Zellproduktion

Band 206
Dr.-Ing. Daniel Bertsch

Optimierung der Werkzeug- und Prozessauslegung für das Wälzschälen von Innenverzahnungen

Band 207
Dr.-Ing. Kyle James Kippenbrock

Deconvolution of Industrial Measurement and Manufacturing Processes for Improved Process Capability Assessments

Band 208
Dr.-Ing. Farboud Bejnoud

Experimentelle Prozesskettenbetrachtung für Räumbauteile am Beispiel einer einsatzgehärteten PKW-Schiebemuffe

Band 209
Dr.-Ing. Steffen Dosch

Herstellungsübergreifende Informationsübertragung zur effizienten Produktion von Werkzeugmaschinen am Beispiel von Kugelgewindetrieben

Band 210
Dr.-Ing. Emanuel Moser

Migrationsplanung globaler Produktionsnetzwerke
Bestimmung robuster Migrationspfade und risiko-effizienter Wandlungsbefähiger

Band 211
Dr.-Ing. Jan Hochdörffer

Integrierte Produktallokationsstrategie und Konfigurationssequenz in globalen Produktionsnetzwerken

Band 212
Dr.-Ing. Tobias Arndt

Bewertung und Steigerung der Prozessqualität in globalen Produktionsnetzwerken

Band 213
Dr.-Ing. Manuel Peter

Unwuchtminimale Montage von Permanentmagnetrotoren durch modellbasierte Online-Optimierung

Band 214
Dr.-Ing. Robin Kopf

Kostenorientierte Planung von Fertigungsfolgen additiver Technologien

Band 215
Dr.-Ing. Harald Meier

Einfluss des Räumens auf den Bauteilzustand in der Prozesskette Weichbearbeitung – Wärmebehandllung – Hartbearbeitung

Band 216
Dr.-Ing. Daniel Brabandt

Qualitätssicherung von textilen Kohlenstofffaser-Preforms mittels optischer Messtechnik

Band 217
Dr.-Ing. Alexandra Schabunow

Einstellung von Aufnahmeparametern mittels projektionsbasierter Qualitätskenngrößen in der industriellen Röntgen-Computertomographie

Band 218
Dr.-Ing. Jens Bürgin

Robuste Auftragsplanung in Produktionsnetzwerken
Mittelfristige Planung der variantenreichen Serienproduktion unter Unsicherheit der Kundenauftragskonfigurationen

Band 219
Dr.-Ing. Michael Gerstenmeyer

Entwicklung und Analyse eines mechanischen Oberflächenbehandlungsverfahrens unter Verwendung des Zerspanungswerkzeuges

Band 220
Dr.-Ing. Jacques Burtscher

Erhöhung der Bearbeitungsstabilität von Werkzeugmaschinen durch semi-passive masseneinstellbare Dämpfungssysteme

Band 221
Dr.-Ing. Dietrich Berger

Qualitätssicherung von textilen Kohlenstofffaser-Preforms mittels prozessintegrierter Wirbelstromsensor-Arrays

Band 222
Dr.-Ing. Fabian Johannes Ballier

Systematic gripper arrangement for a handling device in lightweight production processes

Band 223
Dr.-Ing. Marielouise Schäferling, geb. Zaiß

Development of a Data Fusion-Based Multi-Sensor System for Hybrid Sheet Molding Compound

Band 224
Dr.-Ing. Quirin Spiller

Additive Herstellung von Metallbauteilen mit dem ARBURG Kunststoff-Freiformen

Band 225
Dr.-Ing. Andreas Spohrer

Steigerung der Ressourceneffizienz und Verfügbarkeit von Kugelgewindetrieben durch adaptive Schmierung

Band 226
Dr.-Ing. Johannes Fisel

Veränderungsfähigkeit getakteter Fließmontagesysteme
Planung der Fließbandabstimmung am Beispiel der Automobilmontage

Band 227
Dr.-Ing. Patrick Bollig

Numerische Entwicklung von Strategien zur Kompensation thermisch bedingter Verzüge beim Bohren von 42CrMo4

Band 228
Dr.-Ing. Ramona Pfeiffer, geb. Singer

Untersuchung der prozessbestimmenden Größen für die anforderungsgerechte Gestaltung von Pouchzellen-Verpackungen

Band 229
Dr.-Ing. Florian Baumann

Additive Fertigung von endlosfaserverstärkten Kunststoffen mit dem ARBURG Kunststoff-Freiform Verfahren

Band 230
Dr.-Ing. Tom Stähr

Methodik zur Planung und Konfigurationsauswahl skalierbarer Montagesysteme – Ein Beitrag zur skalierbaren Automatisierung

Band 231
Dr.-Ing. Jan Schwennen

Einbringung und Gestaltung von Lasteinleitungsstrukturen für im RTM-Verfahren hergestellte FVK-Sandwichbauteile

Band 232
Dr.-Ing. Sven Coutandin

Prozessstrategien für das automatisierte Preforming von bebinderten textilen Halbzeugen mit einem segmentierten Werkzeugsystem

Band 233
Dr.-Ing. Christoph Liebrecht

Entscheidungsunterstützung für den Industrie 4.0-Methodeneinsatz
Strukturierung, Bewertung und Ableitung von Implementierungsreihenfolgen

Band 234
Dr.-Ing. Stefan Treber

Transparenzsteigerung in Produktionsnetzwerken
Verbesserung des Störungsmanagements durch verstärkten Informationsaustausch

Band 235
Dr.-Ing. Marius Dackweiler

Modellierung des Fügewickelprozesses zur Herstellung von leichten Fachwerkstrukturen

Band 236
Dr.-Ing. Fabio Echsler Minguillon

Prädiktiv-reaktives Scheduling zur Steigerung der Robustheit in der Matrix-Produktion

Band 237
Dr.-Ing. Sebastian Haag

Entwicklung eines Verfahrensablaufes zur Herstellung von Batteriezellstapeln mit großformatigem, rechteckigem Stapelformat und kontinuierlichen Materialbahnen

Band 238
Dr.-Ing. Raphael Wagner

Strategien zur funktionsorientierten Qualitätsregelung in der Serienproduktion

Band 239
Dr.-Ing. Christopher Ehrmann

Ausfallfrüherkennung von Ritzel-Zahnstangen- Trieben mittels Acoustic Emission

Band 240
Dr.-Ing. Janna Hofmann

Prozessmodellierung des Fünf-Achs-Nadelwickelns zur Implementierung einer trajektoriebasierten Drahtzugkraftregelung

Band 241
Dr.-Ing. Andreas Kuhnle

Adaptive Order Dispatching based on Reinforcement Learning
Application in a Complex Job Shop in the Semiconductor Industry