Distributed Planning for Self-Organizing Production Systems

Zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte
Dissertation
von

Dipl.-Wirt.-Ing. Julius Pfrommer
aus Ostfildern

Tag der mündlichen Prüfung:	22. Juli 2019
Referent:	Prof. Dr.-Ing. habil. Jürgen Beyerer
Korreferent:	Prof. Dr.-Ing. Michael Weyrich

Kurzfassung

Für automatisierte Produktionsanlagen gibt es einen fundamentalen Tradeoff zwischen Effizienz und Flexibilität. In den meisten Fällen sind die Abläufe nicht nur durch den physischen Aufbau der Produktionsanlage, sondern auch durch die spezielle zugeschnittene Programmierung der Anlagensteuerung fest vorgegeben. Änderungen müssen aufwändig in einer Vielzahl von Systemen nachgezogen werden. Das macht die Herstellung kleiner Stückzahlen unrentabel.

In dieser Dissertation wird ein Ansatz entwickelt, um eine automatische Anpassung des Verhaltens von Produktionsanlagen an wechselnde Aufträge und Rahmenbedingungen zu erreichen. Dabei kommt das Prinzip der Selbstorganisation durch verteilte Planung zum Einsatz. Die aufeinander aufbauenden Ergebnisse der Dissertation sind wie folgt:

1. Es wird ein Modell von Produktionsanlagen entwickelt, dass nahtlos von der detaillierten Betrachtung physikalischer Produktionsprozesse bis hin zu Lieferbeziehungen zwischen Unternehmen skaliert. Im Vergleich zu existierenden Modellen von Produktionsanlagen werden weniger limitierende Annahmen gestellt. In diesem Sinne ist der Modellierungsansatz ein Kandidat für eine häufig geforderte “Theorie der Produktion”.

2. Für die so modellierten Szenarien wird ein Algorithmus zur Optimierung der nebenläufigen Abläufe entwickelt. Der Algorithmus verbindet Techniken für die kombinatorische und die kontinuierliche Optimierung: Je nach Detailgrad und Ausgestaltung des modellierten Szenarios kann der identische Algorithmus kombinatorische Fertigungsfeinplanung (Scheduling) vornehmen, weltweite Lieferbeziehungen unter Einbezug von Unsicherheiten und Risiko optimieren und physikalische Prozesse prädiktiv regeln. Dafür werden Techniken der Monte-Carlo Baumsuche (die auch bei Deepminds Alpha Go zum Einsatz kommen) weiterentwickelt. Durch Ausnutzung zusätzlicher Struktur in den Modellen skaliert der Ansatz auch auf große Szenarien.

3. Der Planungsalgorithmus wird auf die verteilte Optimierung durch unabhängige Agenten übertragen. Dafür wird die sogenannte “Nutzen-Propagation” als Koordinations-Mechanismus entwickelt. Diese ist von der Belief-Propagation zur Inferenz in Probabilistischen Graphischen Modellen inspiriert. Jeder teilnehmende Agent hat einen lokalen Handlungsraum, in dem er den Systemzustand beobachten und handelnd eingreifen kann. Die Agenten sind an der Maximierung der Gesamtwohlfahrt über alle Agenten hinweg interessiert. Die dafür notwendige Kooperation entsteht über den Austausch von Nachrichten zwischen benachbarten Agenten. Die Nachrichten beschreiben den erwarteten Nutzen für ein angenommenes Verhalten im Handlungsraum beider Agenten.

4. Es wird eine Beschreibung der wiederverwendbaren Fähigkeiten von Maschinen und Anlagen auf Basis formaler Beschreibungslogiken entwickelt. Ausgehend von den beschriebenen Fähigkeiten, sowie der vorliegenden Aufträge mit ihren notwendigen Produktionsschritten, werden ausführbare Aktionen abgeleitet. Die ausführbaren Aktionen, mit wohldefinierten Vorbedingungen und Effekten, kapseln benötigte Parametrierungen, programmierte Abläufe und die Synchronisation von Maschinen zur Laufzeit.

Die Ergebnisse zusammenfassend werden Grundlagen für flexible automatisierte Produktionssysteme geschaffen – in einer Werkshalle, aber auch über Standorte und Organisationen verteilt – welche die ihnen innewohnenden Freiheitsgrade durch Planung zur Laufzeit und agentenbasierte Koordination gezielt einsetzen können. Der Bezug zur Praxis wird durch Anwendungsbeispiele hergestellt. Die Machbarkeit des Ansatzes wurde mit realen Maschinen im Rahmen des EU-Projekts SkillPro und in einer Simulationsumgebung mit weiteren Szenarien demonstriert.

Abstract

There is a fundamental tradeoff between automation and flexibility in production systems. Large lot sizes can be produced efficiently with automated production systems. Many machines and equipment, like a 5-axis CNC mill, are in principle capable of producing many different kinds of parts. Similarly, (intra-) logistics systems exist for the automated transport and warehousing. Integrating these flexible components to an overall production system that is equally flexible has been prevented by the limits of automation technology in dealing with the ensuing complexity. Most production processes are rigid not only by way of the physical layout of machines and their integration, but also by the custom programming of the control logic for the integration of components to a production systems. Changes are time- and resource-expensive. This makes the production of small lot sizes of customized products economically challenging.

This thesis develops solutions for the automated adaptation of production systems based on self-organisation and distributed planning. The main results are the following:

1. A model of production systems that scales seamlessly from detailed physical process dynamics up to more abstract descriptions of entire supply chains. Compared to existing models of production systems, the proposed approach requires less limiting assumptions and also includes a treatment of concurrency—many productions on many machines in parallel and their interaction. In the sense, the proposed model is a candidate for a "theory of production".

2. Based on the model, an algorithm for the optimization of concurrent production scenarios is proposed. The algorithm combines techniques for combinatorial and continuous optimization. Depending on the level fidelity of of the model, the same algorithm can solve combinatorial scheduling problems, minimize risk in global supply-chains and control physical production processes. For this, the technique of Monte-Carlo

Tree Search is extended. By exploiting algebraic structure in the models, the approach can be scaled to large scenarios.

3. The algorithm is further extended to decentralized optimization by independent agents. For the coordination between agents, the technique of “utility propagation” is developed. Utility propagation is inspired by belief propagation, a well-known technique for inference in probabilistic graphical models. Every agent has a local scope of visibility where can further influence the actions taking place. For the coordination, it is expected that the scope of the different agents is overlapping. The agents are further expected to cooperate, i.e. they are interested to maximize the overall welfare that is generated. The coordination between agents is based on the exchange of messages between neighboring agents. The messages describe the expected generated welfare conditional to the actions that are in the scope of both the sending and the receiving agent.

4. A formal description of reusable skills of production resources (machines and tools) in production systems is developed. By combining the modeled skills with a description of the production steps, executable actions are generated that are described by preconditions and effects. Internally, the actions encapsulate all required program logic for the automation and the runtime-synchronization between different machines and tools that participate in the action.

In summary, the results of this work enable future production systems that are both efficient and adaptive. For this the components of the production system are enabled to use the degrees of freedom that are available to them. By the use of self-organization for the coordination, components of the overall system can react to changes in the system topology and external conditions. The approach was tested in application scenarios—in simulation and in physical production systems. For example as part of the EU-project SkillPro.

Acknowledgments

Let me first thank Prof. Dr.-Ing. habil. Jürgen Beyerer for his mentorship, guidance and shared passion for agent-based and distributed systems throughout the development of this thesis. The time at the IES chair and the interactions were formative and invaluable. Furthermore I want to thank Prof. Dr.-Ing. Michael Weyrich for the discussions and his role as a reviewer of this dissertation.

Fraunhofer IOSB, and particularly the ILT department led by Thomas Usländer, was a great environment to conduct both scientific research leading up this thesis and to pursue high-impact engineering projects in the automation environment. The colleagues at IOSB and especially my former group leaders Miriam Schleipen and Ljiljana Stojanovic taught me the ropes of the craft of applied research. This was an experience from which I still profit immensely, now that I am also in the position of group leader at Fraunhofer IOSB.

Besides the results on distributed planning for production control, a "byproduct" of this thesis was that it enabled me to contribute to the open62541 open source implementation of the OPC UA standard for industrial communication. This background ensures a clear technological path from the theoretical work to the application scenarios. The open62541 development team proved very inspirational and productive, so let me thank Florian Palm and Sten Grüner, Prof. Leon Urbas, Chris Iatrou, Stefan Profanter, and Andreas Ebner.

Another thank you goes to Joseph Warrington and Georg Schildbach for their guidance during my first steps into research during my time at the Automatic Control Laboratory at ETH Zurich.

Most importantly, this thesis would not have been possible without the encouragement and support of my family. *Ce thèse n'aurait pas été possible sans tout ton soutien, Ingrid!*

Karlsruhe, July 2019 *Julius Pfrommer*

Table of Contents

List of Figures

List of Tables

List of Symbols

General Notation

$a, \dots, z$	Scalar (including tuples) and function mapping to a scalar
$A, \dots, Z$	Set
$\mathcal{A}, \dots, \mathcal{Z}$	Graph represented by a tuple (V, E) with nodes V and edges $E \subseteq V \times V$
$\boldsymbol{a}, \dots, \boldsymbol{z}$	Vector or function mapping to a vector. Column vectors are constructed as $\boldsymbol{a} = (a_1, a_2, \dots)^\top$.
$\boldsymbol{A}, \dots, \boldsymbol{Z}$	Matrix. Constructed as $\boldsymbol{A} = [a_{11}\ a_{12}; a_{21}\ a_{22}]$ or from column vectors as $\boldsymbol{A} = [\boldsymbol{a}\boldsymbol{b}]$.
$\mathbf{0}, \mathbf{1}$	Zero and one column vectors. The size is clear from the context or explicitly mentioned.
$\boldsymbol{\nu}_i$	Basis-vector with zeros and a single one-entry at index i

General Sets

$\mathbb{Z}$	Set of integers
$\mathbb{N}$	Set of natural numbers (without zero)
$\mathbb{N}_0$	Set of natural numbers (including zero)
$\mathbb{R}$	Set of real numbers
$\mathbb{R}_+$	Set of real positive numbers (including zero)

Probability

$\mathbb{P}$, $\mathbb{E}$	Probability, Expectation
$\mathcal{N}(\mu, \sigma)$	Normal distribution with mean μ and standard deviation σ
$\mathcal{U}(X)$	Uniform distribution on the support of set X
$\sim$	Distributed according to
$\underline{a}, \dots, \underline{z}$	Random variable (scalar)
$\underline{\boldsymbol{a}}, \dots, \underline{\boldsymbol{z}}$	Random variable (vectorial)

Production System Model

ρ, P	Product type, Set of product types
$\boldsymbol{p} \in \mathbb{N}_0^{\lvert P \rvert}$	Collection of products, represented by a vector with the number of products for each product type
c, C	System component, Set of components
s, S	Component state, Set of component states
σ, Σ	Time-indexed resource state, Set of time-indexed resource states
a, A	Action, Set of actions
C_a	Set of components that participate in the action a
Σ_a	Feasible initial states for the action a
θ, Θ_a, Θ	Action parameter, Parameter space of action a, Parameter space across all actions
o, O	Observation, Set of observations
$\mathfrak{r} : \Sigma \times A \times \Theta \times \Sigma \to \mathbb{R}$	Reward function

Distributed Production System Model

i, I	Agent, Set of agents
$j \in N(i)$	Neighbours of agent i
$a_i \in A_i$	Action where agent i participates projected to the scope of agent i
$C_{a,i}$	Set of components that participate in action a that are also in the scope of agent i
$\Sigma_{a,i}$	Feasible initial states for the action a for the components that are in the scope of agent i

Trace Theory

$[\boldsymbol{w}]$	The trace of equivalent action sequences generated by the action sequence $\boldsymbol{w}$
$\boldsymbol{w} \simeq \boldsymbol{v}$	Action sequences $\boldsymbol{w}$ and $\boldsymbol{v}$ are equivalent
$a \perp b$	Actions a and b are independent
$w_k \prec w_l$	Partial order relation between the elements with indices k and l of the sequence $\boldsymbol{w}$
τ	Permutation of sequence element indices

Planning

$\boldsymbol{w} \in A^*$	Action sequence with elements w^k
ε	Empty action sequence
$W, W^{\boldsymbol{\sigma}}$	Set of action sequences; Set of possible action sequences starting at state $\boldsymbol{\sigma}$.
$W_i, W_i^{\boldsymbol{\sigma}}$	Set of action sequences for agent i; Set of possible action sequences for agent i starting at state $\boldsymbol{\sigma}_i$.

$W_i \mid \boldsymbol{v}_{ij}$ Set of possible action sequences for agent i starting at state $\boldsymbol{\sigma}_i$ that project to the sequence $\boldsymbol{v}_{ij}$ for the shared scope of the agents i and j

$\boldsymbol{h}$ History of (parameterized) actions and observations. The elements are (a^k, o^k) or (a^k, θ^k, o^k) respectively.

H Set of possible histories with (parameterized) actions and observations

$\boldsymbol{h}_i, \boldsymbol{h}_{ij}$ History projected to the scope of agent i; History projected to shared scope of the neighboring agents i and j

H_i Set of possible histories with (parameterized) actions and observations for agent i

$\pi : \Sigma \to A \times \Theta$ Policy for observable settings. Maps from the current state to the next action.

$\pi : H \to A \times \Theta$ Policy for partially-observable settings. Maps from observed histories to the next action.

$\mathfrak{v}, \mathfrak{q}$ V-value and Q-value for the expected reward under optimal decision making

$\mathfrak{m}_{i \to j}$ Message send from agent i to a neighbor $j \in N(i)$

Miscellaneous

$1_{[\psi]}$ Indicator function (one if ψ is true, zero otherwise)

$n[k], \boldsymbol{n}[k], N[k]$ Access to a hash-map under the key k. The shown notation is for a scalar, vector or set value respectively. If no value was stored for k prior, n returns a (scalar or vectorial) zero or the empty set.

$\succeq$ Element-wise comparison of two column-vectors

List of Acronyms

AMS	Automated Manufacturing System
B&B	Branch and Bound
BP	Belief Propagation
DL	Description Logics
ERP	Enterprise Resource Planning
EWO	Enterprise-Wide Optimization
FMS	Flexible Manufacturing System
GDL	Generalized Distributive Law
JSP	Job-Shop Problem
LNF	Lexicographical Normal Form
MARL	Multi-Agent Reinforcement Learning
MCTS	Monte-Carlo Tree Search
MDP	Markov Decision Process
MES	Manufacturing Execution System
MILP	Mixed-Integer Linear Program
MPC	Model Predictive Control
ODE	Ordinary Differential Equation
OEM	Original Equipment Manufacturer
OO	Optimistic Optimization
OPC UA	OPC Unified Architecture

POMDP	Partially-Observable Markov Decision Process
PGM	Probabilistic Graphical Models
PLC	Programmable Logic Controller
RL	Reinforcement Learning
SC	Situation Calculus
SCADA	Supervisory Control and Data Akquisition
TT	Trace Theory

1 Introduction

> *There has been a great deal of talk, much of it well founded, that the effect of science on economics and on the economy has not only been very large but that something like a second industrial revolution is impending. Illustrating this are the enormous advances in communications – physical and informational –, advances in automatization and in the domain of information and control, and finally, atomic energy.*
>
> *John von Neumann [Neu55]*

1.1 Production and Logistics in a Global Economy

After introducing the assembly line for the production of the Model T automobile in 1908, Henry Ford sought to make his company self-sufficient. For the supply of raw material, his Ford Motor Company bought 700,000 acres of forest, rubber plantations, iron and coal mines, and so on. The Ford River Rouge Complex near Detroit was designed to transform the incoming raw material into fully assembled cars. The manufacturing operations performed at River Rouge included coal coking, steel forging, sheet metal stamping, engine casting, lumber milling, tire making, the production of sheet glass from molten sand, and many more. All leading up to the final production step: the final car assembly in Ford's assembly line [Bri03]. Since Ford's only product at the time was the Model T, all production processes were highly specialized to maximize efficiency and reduce costs. This made the Model T the most affordable car at the time. Ford's competitor Chevrolet had a different approach. They used generic manufacturing equipment that could produce parts for several models at the same time. This also enabled frequent updates of the car models. Innovations of the 1920s, such as motors with electric starters that require no hand crank, let the Model T appear increasingly outdated. In 1927, Ford finally introduced a successor: the Model A. However, since the production processes were tailored towards the Model T, the changeover proved difficult. The River Rouge site came to standstill for a duration of six months until production could be slowly resumed [Hou85]. In the following years, Ford gave up the model of the highly integrated manufacturing site. Today, like all automotive companies, Ford operates a range of production sites that are specialized on a range of parts (e.g. the internal combustion engine) that is used for several car models. And also the final assembly lines can switch between car models to adjust to changing customer demands. All production sites are connected with logistics networks and rely heavily on external material and component suppliers.

Ford's change from an integration manufacturing site to a network of interconnected production sites is exemplary for the evolution of many manufacturing industries. Initially, customized craft production is replaced by mass production, resulting in large efficiency gains and opening up new markets. When the markets are divided up, companies diversify their product portfolio to cater for individual customer groups. This leads to smaller order sizes, reducing the efficiency of the mass production approach. New methods have been developed to enable customization without loosing all the efficiency gains of mass production. Among the most popular ones are Lean Production [Ohn88] (also known as the Toyota Production System) and the Just in Sequence (JIS) inventory strategy [WS11]. They enable production sites to reduce the minimum order size that is still economical to produce. Sometimes lot sizes are reduced to the absolute minimum: a single customized product.

While automotive brands rely on a network of suppliers, these relationships are relatively stable. Building up the capacity to produce a specific part in high quantities in the expected quality takes time and investments in automated processes. This only makes economical sense when years of high demand are expected. For many retail goods, a similar division of labour happens in a complex supply-chain. But, these relationships can be established and dissolved literally overnight. Cheap long-range shipping, the reduction of tariffs and easy communication has enabled world-wide supply chains. We will take the example of global supply chains in the apparel industry [Ger99]. The supply-chain for a cotton shirt comprises the provider of raw cotton, spinning of the yarn, color dyeing of the yarn, weaving of the textile, cutting and sewing to make the shirt, design printing and stitching of brand logos. All of these production steps are typically executed by different companies in different countries. And the supply-chain is dynamically reconfigured for individual orders based on availability and price. A shirt bought today may have taken a wholly different way around the world than the same shirt bought a week earlier in the same store [Chr00]. Many apparel retailers even forego central warehousing for their stores. Instead, products are delivered directly from the last link in the supply chain to the store. So the retailers do not bind capital in stock for the entire season. And they can react within weeks to data showing good or bad sales of a specific product [CM15]. On the downside, most western apparel brands do not control their supply-chain and rely on sourcing agents from overseas.

The biggest sourcing agent is Li&Fung Limited. Operating out of Hong Kong, Li&Fung self-describes it's core business as "managing the supply chain for high volume, time sensitive goods" [FFW07]. Li&Fung owns no factory, warehouse or inventory and can still source nearly any retail good. Its database contains factories throughout Asia with their support for different manufacturing processes, available capacity and logistics options, as well as the availability and prices of raw material commodity components. Orchestrating the supply chain is a profitable business. In 2015, Li&Fung achieved a turnover of $18.8 billion and a healthy $2.2 billion profit [Lim15]. The leverage the sourcing agent has over the supply chain is seen increasingly critical by companies who rely on their services. In 2015, Wal-Mart announced a plan to reduce their reliance specifically on Li&Fung. Relying too much on a single provider had become a strategic weakness for the world's biggest retailer [WS15].

So why are dynamic reconfigurations of the supply-chain possible for the apparel industry, but not for automotive? In the apparel supply-chain, suppliers provide generic access to manufacturing processes that can be used for many different customers with little changeover costs and fast production ramp-up.

- Standardized commodity goods enable a high degree of automation. For example, the objective of textile plants is to run their power looms with as little downtime as possible. The automated equipment allows the configuration of different product types. For example, a Jacquard loom can configure different weaving patterns. The difference between product types can be entirely handled by the automated production system.

- On the other end of the spectrum, cutting and sewing of apparel is highly dependent on human labour. Lot-sizes are generally smaller and the type of product can change drastically between orders (e.g. switching from jeans to dress shirts). Handling of pliable textile material is difficult for automated equipment and requires custom machines and long changeover times. This makes automated equipment ineconomical as long as cheap human labour is available in overseas countries.

Globalization and the decentralization of supply chains lead to increased requirements for flexibility in production [Abe+06]. In practice, however, flexibility in production is a conflicting goal with efficiency. Automated systems provide the increased efficiency required for mass production. But they require considerable investment for the initial setup and the changeover between products. In recent years, many countries with a large industrial base have set up research programmes to renew industrial production with the increased use of information processing and communication technology. Among these programmes are “Industrie 4.0” in Germany [KWH13], “Made in China 2025” [Ken15] in China and “Industrie du Futur” [FD16] in France. A major part of these efforts is the creation of new automation technology that improves on the tradeoff between efficiency and flexibility in production.

1.2 The Structure of Automated Production Systems

The vast majority of control systems for production systems is organized as a hierarchy. This is true both for discrete manufacturing and continuous production processes (e.g. chemicals, pharmaceuticals, beverages). Figure 1.1 shows the automation pyramid, a frame of reference for the hierarchical design of most automated systems in discrete manufacturing. Figure 1.2 depicts the typical automation hierarchy from the process industry (e.g. chemicals and pharmaceuticals). Decisions are made hierarchically and data is aggregated more and more as it is forwarded to the upper levels of the automation hierarchy. Hierarchical control follows the principle of subsidiarity, where upper levels make high-level decisions that are gradually refined as they are forwarded down the automation hierarchy. Subsidiarity is a necessary consequence of the fact that information is aggregated when it moves up the automation hierarchy.

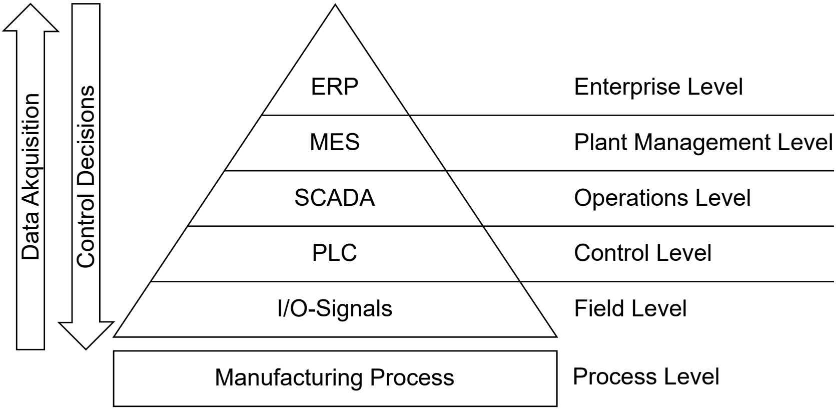


Figure 1.1: The automation hierarchy in discrete manufacturing. The IEC 62264 / ISA-95 standard does not include the enterprise level. It was added here to include interfaces to systems outside the shopfloor.

The models used for decision making in the upper levels are more and more coarse and abstract away low-level details. But the low-level details have to be considered eventually. The lower levels in the automation hierarchy takes decisions from one level above and “fill the gaps”.

In many systems, the control levels are tightly coupled. Changes to a component of a production system usually requires changes in many adjacent systems both vertically and horizontally. The subsystems are interwoven and implicit assumptions about adjacent systems are represented only in custom control code. This makes modifications to automated systems costly and time-intensive. Reducing this effort is the focus of an entire research community.

The remainder of this section gives an overview on the most important planning and optimization methods for decision-making on the different levels of the automation hierarchy. Whilst it is not possible to provide a complete enumeration, the examples from this chapter will set the frame for the modeling and planning techniques that are introduced later on.

The Control Level

Modern control theory and their implementation on computers can be traced back to work done at MIT in the 1940s. There, Norbert Wiener first coined the term

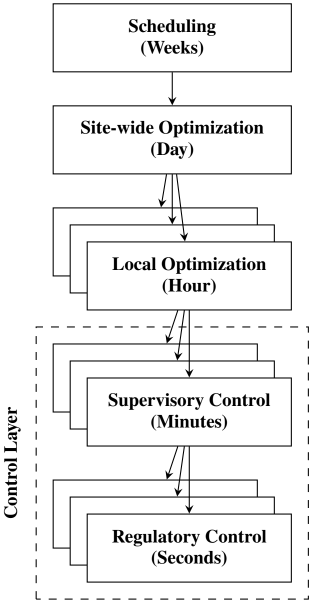


Figure 1.2: Typical automation hierarchy of a chemical plant [Sko04].

"cybernetics" as the conjunction of control and communication [Wie48]. At the same time, project Whirlwind, conducted at Jay Forrester's Servomechanisms Laboratory, developed digital "feedback control" for numerically controlled (NC) manufacturing processes [Rei91]. Both modern Programmable Logic Controllers (PLCs) and the application of feedback control methods on digital computers are descendant from this work in a direct lineage. The fundamental difference between the two lies in the type of decisions they have to make. Programmable Logic Controllers (PLC) generally are driven by a state machine with discrete transitions or events. Feedback (optimal) control is mostly concerned with physical systems with continuous dynamics.

Programmable Logic Controllers Programmable Logic Controllers [Joh87; Wal12] (PLC) are directly interfaced with a physical process via sensors and actuators. In the automation of discrete manufacturing, PLC couple the physical system with digital control and communication. This coupling requires custom program code to accomodate for the specific details of the

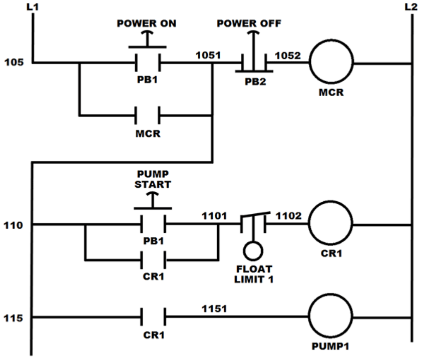


Figure 1.3: Example for a PLC program in ladder logic

physical system and its intended functionality. As the control level often deals with safety-critical functionality, hard bounds on realtime reactiveness have to be guaranteed. The IEC-61131 languages [Com93] are standard for PLC programming and mandate a programming style that is idiomatic to industrial controllers. Figure 1.3 shows an example for one of the IEC-61131 languages, ladder logic, which is directly descendent from the analog circuits that were originally used for industrial control before the PLC.

The logic coded into a PLC is mostly reactive. Sensor inputs are read and translated into actuation commands. This loop is repeated at a fast pace (many hundred Hertz) for realtime operations. Lengthy planning procedures do not usually fit into the constraints of the control loop in a PLC. The IEC-61499 standard is intended as a modernization of IEC-61131 [Vya11]. It adds an event-based control flow to the strictly cyclic operations of previous PLC generations. But even with IC-61499, PLC-based control is mostly reactive. Computationally expensive planning is generally not performed in a safety-critical control environment.

Automated production systems typically are integrated from components that come with their own control hardware. The integration requires communication between individual controllers and custom control software to react to cyclically transmitted status messages and events. On the control level, traditional fieldbuses are still common today [Zur14]. Fieldbuses are even mandatory to use if safety-critical functionality relies on digital communication between controllers or between a PLCs and field devices with sensors and actuators. The programming of PLCs is often finished on-site as part of the integration of system components. As many automation systems are custom solutions, code reuse for PLCs is difficult even if systems are built from standard components. This leads to a tight coupling that also increases the time required to make changes in an existing system.

Feedback Control The canonical definition of an optimal control problem is as follows [Lib11]:

$$\dot{\boldsymbol{x}} = f(t, \boldsymbol{x}, \boldsymbol{u}), \quad \boldsymbol{x}(t_0) = \boldsymbol{x}_0 \tag{1.1}$$

The system dynamics is described by an ordinary differential equation (ODE) f. The system state at time t is $\boldsymbol{x}(t) \in \mathbb{R}^n$ and its evolution depends on the control input $\boldsymbol{u}(t) \in \mathbb{R}^m$. The initial condition is given by $\boldsymbol{x}_0$. A cost functional for the state evolution assigns costs to the state and control effort between times t_0 and t_f and an additional terminal cost on the final state $\boldsymbol{x}(t_f)$.

$$C(\boldsymbol{u}) = \int_{t_0}^{t_f} L(t, \boldsymbol{x}(t), \boldsymbol{u}(t))dt + K(t_f, x_f) \tag{1.2}$$

The problem of optimal control is to compute $\boldsymbol{u}^* = \arg\min_{\boldsymbol{u}} C(\boldsymbol{u})$. In this general framework, computing $\boldsymbol{u}^*$ is a variational problem as $\boldsymbol{u}$ is a (vectorial) function over time [Lue69]. A popular approach is to discretize the time domain $T = \{t_0, t_1, \ldots, t_f\}$ so that $\boldsymbol{u}^*$ problem of finding the sequence of $\boldsymbol{u}$ that minimizes C. Applying the resulting $\boldsymbol{u}$ blindly until t_f is called *open-loop* control. Repeating the optimization after every time period with updated state information is called Model Predictive Control (MPC) or Receding Horizon Control [ML99; Mac02].

Traditionally, feedback control has be implemented as analog electrical circuits. But these are restricted to relatively simplistic solutions, such as PID controllers [Ben93]. With the increase in computational power available in control devices, Model Predictive Control (MPC) has become possible for many application. Here, the control problem is stated as an optimization problem that is solved repetitively at a high frequency to incorporate sensor measurements for “feedback” control. Optimal control as an optimization problem originates from the association of dynamical system with control input with

The Operations Level

Supervisory Control and Data Acquisition On the operations level, decisions are being made with respect to a horizon of minutes or hours. So-called SCADA systems (Supervisory Control and Data Acquisition) collect data from the control level, aggregate it and present it to a higher-level decision making system or a human operator. Some aspects of the lower-level control layer, such as reactions to safety critical conditions are typically abstracted away. SCADA systems are often interfaced directly with the PLC that control the process. So the possible choices of communication technology are reduced to the capabilities of the PLC. In addition to classical fieldbuses, Ethernet-based protocols are making inroads into factories. The most popular protocol for non-realtime communication on the shopfloor today is OPC UA [MLD09].

Performance and Quality Control The supervision of the process performance and resulting product quality is performed on the operations level [Jel06]. The performance and quality of production processes is generally varying over time. The reasons for this are the following: a) an inherent stochasticity of the process, b) changes to the input material and semi-finished goods, c) effects from changing ambient conditions, such as temperature and humidity, d) gradual degradation of the equipment and tools and their maintenance, and e) the evolution of the dynamic system state. To illustrate e), take the example of a stamping press. The evolution of the dynamic system state could refer to an increase in the tool temperature during long uninterrupted production runs, or a buildup of residual oil from the metal coating in the stamp tool. It is often up to a skilled process expert to adjust the process parameters at runtime to ensure the required performance.

The Plant Management Level

On the plant management level, an entire shopfloor is considered. Typically the planning horizon is between several hours and several days of operation [She03].

Scheduling Scheduling theory [Pin08] is concerned with the distribution of production steps to machines in order to maximize the overall efficiency and to reduce costs. The field was active since the early 1960s [GT60] and many important breakthroughs have been made. Scheduling functionality is often sold under names such as Manufacturing Resource Planning (MRP, [Wig81]) or Advanced Planning and Scheduling (APS). Traditionally, due to the long runtime of schedule optimization, updates were being computed at night. If the original plan is disrupted by an unforeseen event, such as a delay or a machine breakdown, the scheduling procedure is restarted or the original plan is repaired with appropriate heuristics. Repairing or iterative refinement of plans has a long history [HL05]. Modern systems can also perform a full rescheduling even during a running shift [VHL03; Dim15].

Material Handling Material handling with uncertain arrival and processing times is usually modeled using stochastic processes and queuing theory [Gro08; Fur18]. Based on such a stochastic queuing system model the system behaviour can be described. One approach is to compute a steady state occupancy of the queues, for example based on the landmark BCMP theorem [Bas+75]. The discipline of queuing theory is concerned not only with computing steady state occupancy, but also to apply queuing algorithms / network schedulers for customer routing such that the network throughput is optimized.

The Enterprise Level

Enterprise Resource Planning Enterprise Resource Planning (ERP) describes a class of software systems to assist enterprise-wide management [Jac+07]. The ERP products with the highest market share are SAP ERP (previously SAP/R3) and the Oracle E-Business Suite [Gar18]. In many aspects, ERP functionality mirrors tasks performed at the plant management level. But the timeframes are generally much longer and several production and warehousing sites are jointly considered. Enterprise-wide optimization (EWO) aims at optimizing the operations of supply, manufacturing and distribution activities of a company to reduce overall costs and inventories [Gro05].

Supply-Chain Management Supply-Chain Management (SCM) [Ali05; GF08] is concerned with production scenarios that include several layers of suppliers. SCM is most common in industries where suppliers are not delivering specialized parts instead of commodity products. The integration with suppliers is often very tight. This enables the reduction of buffer storage at the production site by the use of just-in-time and just-in-sequence delivery. The so-called bullwhip effect [LPW97] describes how small fluctuations in customer demand lead to large fluctuations in demands at suppliers that are several tiers removed. A big motivator for digitalisation and information sharing in the supply-chain is the reduction of the bullwhip effect by enabling better forecasts for the suppliers.

1.3 Approaches for Flexible Production Systems

This section discusses approaches to render automated production system flexible. The possibility of flexible production is one of the driving motivations for Industrie 4.0 [Wey+14]. This thesis has a scope on automated production. Organizational methods that focus on the human element in production, such as Lean Manufacturing [Ohn88], are therefore not considered in depth. Several authors have developed frameworks to characterize flexibility in production systems [Bro+84; BS88; GG89; SS90; Ger93; DT98; Wie+07]

The scientific literature uses specific terms to describe flexibility in a production context. See Figure 1.4 for a common nomenclature by Wiendahl [Wie+07]. Even though specific terms exist, the term flexibility is deliberately used with its colloquial meaning in this thesis: The models and planning algorithms developed in this thesis apply to all level in the automation hierarchy. The specific terms for flexibility from the scientific literature are mostly tied to one level of the automation hierarchy. We aim to avoid misunderstandings by the specific terms outside of their commonly understood definition.

Service-Oriented Production Systems The principle of service-orientation is used in computer science to develop system architectures where components are loosely coupled [Mac+06]. A specific service provider can be exchanged as long as the interfaces for interaction remain identical and the underlying functionality

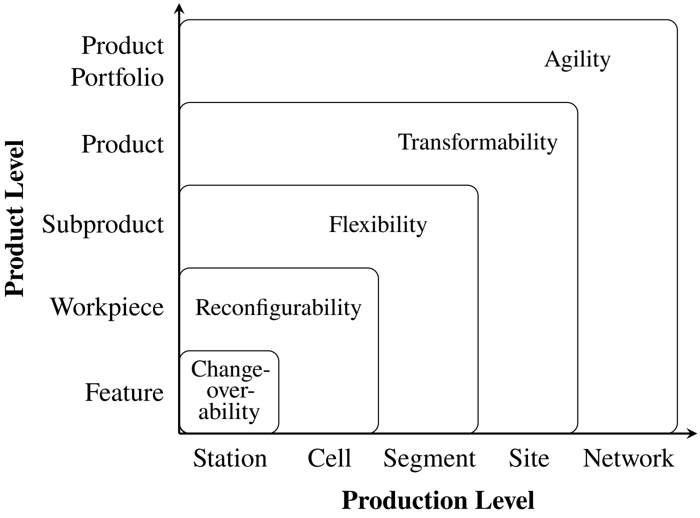


Figure 1.4: Classes of manufacturing changeability from [Wie+07].

is still provided. Discovery mechanisms are used to find and select appropriate service providers. In the context of Industrie 4.0, service-orientation is regarded as an enabler for future control system architectures that dissolve the classical automation hierarchy. See Figure 1.5 for a popular depiction. The DIN SPEC 16593-1 standard [DIN18] defines a reference model with basic principles for service-based architectures in the context of Industrie 4.0. This is the common basis for technical realisations to the vision from Figure 1.5.

A range of research projects has translated service-orientation to production control. The authors from the SOCRADES project [JS05; De +08; Cân+11] and Shen et al. [She+07] develop a service-oriented manufacturing system architecture where semantic technologies are used to match possible providers of functionality in a manufacturing system. Loskyll et al. [Los+11; Los+12] expand the concept of semantic service discovery to the parameterization and orchestration of services. For this, they develop a domain-specific ontology for the use in semantic reasoning tools. Puttonen et al. [PLM13] describe a set of specialized web services for composing and invoking semantically enriched automated procedures in a manufacturing setting. They also present an algorithm

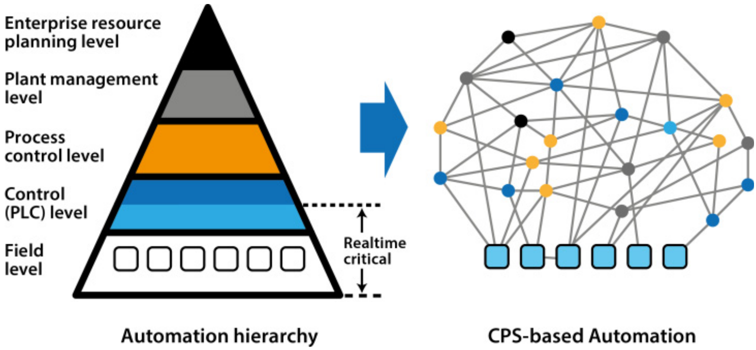


Figure 1.5: Decomposition of the automation hierarchy with distributed services [Mes13; Mon14].

to identify the steps required to reach a predefined goal state. Dürkop et al. [Dür+14] discuss the use of service-oriented architectures in reconfigurable manufacturing systems (see Figure 1.4 and the technical challenges that need to be overcome. [SZW17] use model-based approach for the service development and a modular architecture to reduce the complexity of service-based production systems. [LV15] combine service-oriented manufacturing control with a multi-agent architecture. The Smart Factory Web testbed in the Industrial Internet Consortium (IIC) uses web services for planning and control in global supply chains [Jun+17].

Agent-Based Production Systems An even more radical departure from the classical automation hierarchy is investigated with agent-based distributed control of production systems. Agent-based systems in manufacturing and logistics are the topic of a dedicated research community that has been active since the 1980s [DP87; LS92]. The survey papers [MVK06; LK08; Lei09; LMV13; LK15] give an account on the history of agent-based control and an overview on the focus of current work. Notably, the IEEE-IES Technical Committee on Industrial Agents (TC-IA[1] brings together researchers on an international level.

[1] https://tcia.ieee-ies.org/

Software frameworks have been developed to assist the development of agent-based systems. For example the well-known JADE project [BCG07]. The frameworks for software agents, however, do not provide abstractions specifically for the production domain and are used for the development of distributed software systems in general.

The core challenge of agent-based control is the coordination of individual decision making across agents. The remainder of this paragraph discusses the most common approaches. A common coordination mechanisms for agent-based control is negotiation [ZR89]. The Contract Net Protocol (CNP) [Smi80] replaces the market with a negotiation scheme in order to decompose and distribute tasks between agents. The CNP has been applied for agent-based manufacturing systems in a range of research projects and industrial installations [Par87; LL94; SKB97; Oue+99]. While the CNP is mostly used for greedy decision making, other authors have integrated scheduling theory with agent-based control [SWH06; Agn+14; Bad11] Other coordination mechanisms are nature-inspired and derived from the behavior of animals [XL08].

Holonic production control is a special case of agent-based control. The term *holon*, originally coined in [Koe68], refers to systems made up from components that encapsulates both physical assets and virtual functionality [GLK98; Fis99; MB00]. The key idea is that the system components are themselves holons. This goes beyond the usual system-of-systems approach, as holons are self-similar in the sense that the structure and functionality of the constituent parts is governed by the same principles as their parent. Taken to its extreme, this self-similarity in manufacturing systems has led to the concept of the fractal factory [War93]. The PROSA project has proposed a architecture reference architecture for holonic manufacturing systems [Van+98]. See Figure 1.6 for the building blocks defined by PROSA.

What is currently lacking in the field of agent-based and holonic manufacturing control are widely used benchmark scenarios to quantitatively compare the proposed coordination mechanisms. Compared to other scientific fields, this has led to many competing approaches without a clear winner and uncertainty on how well the different approaches can cope with aspects outside of their original scope. For example if unforeseen events are introduced in a stochastic environment. For practitioners, this has led to an overwhelming range of choices.

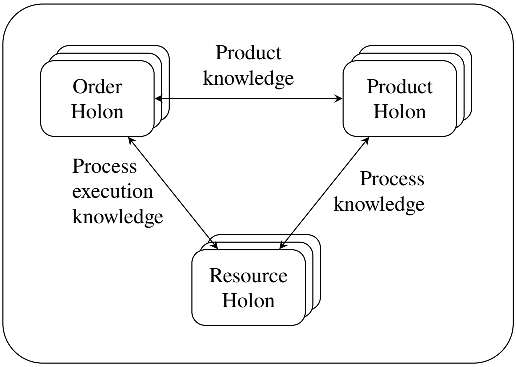


Figure 1.6: Building blocks of a holonic manufacturing system according to the PROSA reference architecture [Van+98].

For researchers, years of effort have so far not amalgamated into a unified theory of agent-based production control.

Plug and Produce

The idea of Plug & Produce is derived from plug-and-play functionality known from the USB interface for computer hardware. There, well-known device classes with standardized functionality remove the need for custom software drivers for the hardware integration. Arai et al. [Ara+00] first translated plug-an-play to the production domain and coined the term Plug & Produce. Onori et al. [Ono+12] use the concept for a self-configuring assembly system at the shop-floor level.

The integration of machines and equipment with Plug & Produce encompasses the following aspects: First of all, basic connectivity is established for an existing (industrial) communication infrastructure [Dür+12; Rei+10]. Second, the new component announces its presence to a central controller or directly to the adjacent components with a discovery mechanism [Pro+17]. Third, in production, there exists a wide range of machines and equipment. This heterogeneity cannot be reduced to a small number of devices classes. A way to enable Plug & Work scenarios in the face of device heterogeneity is the use of self-descriptions languages for the integration [OHN14; Sch+15a]. The fourth and most challenging aspect is the functional integration. Lepuschitz et al. [Lep+11] show reconfiguration of manufacturing resources based on distributed IEC 61499 function blocks and a semantic description of the manufacturing setting and the expected behavior.

Many of the published Plug & Produce implementations use a dedicated interconnector module that acts as a facade for manufacturing equipment and provides a uniform interface and that generates low-level commands for the underlying device [NWS07; Dor+17].

One approach for the functional integration in Plug & Produce is the modeling of the skills of technical equipment. For this, see the review of the state of the art in Chapter 5.

1.4 The Missing Hierarchy of Production Theories

Scientists and engineers use models on a level of abstraction that is the most useful for the phenomena under investigation [Gie04]. It often occurs that a more accurate model is available in principle. But working with an increased level of accuracy would overburden the analysis with unnecessary complexity. For example, an electrical engineer laying out the power grid of a city will not use Maxwell's Equation for a power-flow study. Many technical fields have arranged these model approaches (theories) in a hierarchy. This hierarchy has evolved over time and – in the natural sciences – its development is closely related to the process of scientific discovery [Kuh62; Car84]. As an example, Figure 1.7 shows how the model hierarchy established in the field of optics . If some phenomena cannot be explained one can resort to a more detailed (and computationally or analytically more expensive) model until the first principles from Maxwell's Equations and quantum physics are reached [MW59].

Different academic fields have produced "Theories of Production". For example economics [Sch34; Dan66; She71], business administration [Sch86; Fär88] and production management [Dyc06]. Around the year 2000, prominent authors have called for a unified theory of production that provides a common foundation that integrates existing results [Dyc03; Sch04; WNH10]. Recent years have seen a range of proposals to fulfill this need [NW10; Sch+11; Sch+15b]. The authors of [Sch+17] provide a comprehensive review and

	Reflection	Refraction	Color	Diffraction	Polarization	Kerr Effect	Faraday Effect	Coherence Lengths
Ray Optics	✓	✓	✓					
Huygen's Waves	✓	✓	✓	✓				
Transverse Waves	✓	✓	✓	✓	✓	✓	✓	
Maxwell's Equations	✓	✓	✓	✓	✓	✓	✓	
Quantum Mechanics	✓	✓	✓	✓	✓	✓	✓	✓

Figure 1.7: The model hierarchy in optics. (Reproduced from `http://www.argmin.net/2018/01/25/optics`.)

classification of such production theories. But the cited work remains mostly conceptual and does not model the detailed control of automated systems on the lower levels of the control hierarchy.

By contrasting the model hierarchies in other academic fields with the control hierarchy in automation, one could aim for a high-fidelity model at the bottom layers that is abstracted more and more as we go up in the control hierarchy. While that is the case, we want to stress that the underlying modeling principles should stay the same for all levels of the control hierarchy.

The work on a theory of production is not only of academic interest and also relevant to practitioners that have to cope with the increasing complexity of production systems. Currently, each layer of the control hierarchy works with dedicated system models that are tailored for the tasks at hand. But the models are so task-specific that they become mutually incompatible. This is problematic at vertical as well as horizontal interfaces of the control hierarchy. Assumptions about the behavior of adjacent components are implicit in the custom rules for the interaction between subsystems and encoded in custom control program code. The lack of a common core to translate between subsystems leads to constant manual effort for system integrators and the loss of flexibility due to the task-specific hard coupling of components.

1.5 Scientific Contributions and Thesis Organization

The original title at the very beginning of the work leading to this thesis was "Agent-based production control using the paradigm of self-organisation in the context of Industrie 4.0". The path to the final thesis led through scientific fields that are not directly associated with that original title. For example Probabilistic Graphical Models, Convex Optimisation, Description Logics, Reinforcement Learning, and many more. Some of these detours did not pay off as intended. But many did. While not always visible, the thesis has stayed true to the original topic and the results and techniques from different fields finally tie to together into one body of work.

The thesis is organized into six chapters. See Figure 1.8 for an overview and recommended reading order. The scientific contributions are closely related to the thesis structure. In general, prior results are summarized in dedicated "background" sections. Beyond the background sections, all results with an explicit reference to the literature were developed as part of the thesis development. It follows a summary of the key developments.

Chapter 2: A Model of Concurrent Production Systems A novel model representation for production systems is introduced. The goal of the model is to provide a uniform representation of both discrete and continuous production processes on all levels of the automation hierarchy. To achieve this, the model combines the following aspects:

- Seamless scaling from the dynamics of cyber-physical components within machines and equipment up to the orchestration of global supply chains.
- Integration of discrete manufacturing (where individual work-pieces are considered) and process manufacturing (for the production of chemicals, beverages).
- Representation of concurrency, i.e. parallelism and the synchronization of operations across multiple system components.

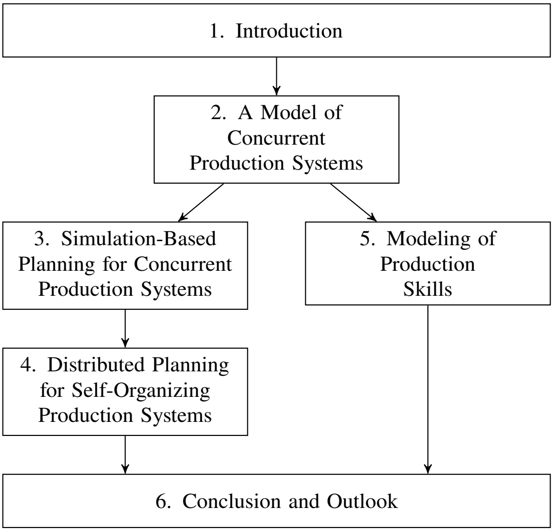

1. Introduction
2. A Model of Concurrent Production Systems
3. Simulation-Based Planning for Concurrent Production Systems
5. Modeling of Production Skills
4. Distributed Planning for Self-Organizing Production Systems
6. Conclusion and Outlook

Figure 1.8: Structure and recommended reading order of the thesis.

- Representation of both deterministic and stochastic scenarios.

The following four chapters aim to show that the model representation not only presents a common core for higher-level models but is practical to use in flexible production systems. For this, a tailored planning algorithm is developed, further scaling of planning is achieved by exploiting additional model structure, the planning algorithm is extended to distributed planning by independent agents, and finally models of the skills of machines and equipment are used as high-level abstraction from which executable low-level representations are derived for runtime control.

Chapter 3: Simulation-Based Planning for Concurrent Production Systems The production system model implies a planning problem: maximizing the expected reward that is generated. Existing planning algorithms from the production domain do not apply to the model from Chapter 2. They make limiting assumptions on the planning problem structure that no longer apply. To solve the planning problem, an algorithm based on Monte-Carlo Tree Search (MCTS) is developed.

- Combines MCTS for discrete decision making with Optimistic Optimization (OO) for decision-making on continuous domains.
- Requires only forward-simulation (rollout) of the planning scenario.
- Additional knowledge about the planning problem can be integrated via so-called Rollout Policies. A large class of planning problems is identified where a relaxation of the planning problem can be solved as a Mixed Integer Linear Program (MILP). The solution to the MILP is then used for the Rollout Policy.
- The algorithm is shown to solve a wide range of standard benchmark problems after casting them into the model of Chapter 2. The range of benchmark problems considered includes combinatorial decision-making in the Jopshop Scheduling Problem (JSP) and optimal control for the swing-up of an inverted pendulum.
- A large class of models is identified that can be relaxed to a Mixed-Integer Linear Program (MILP). This leads to a large improvements for planning in scenarios where actions are repeated many times in a row.

Chapter 4: Distributed Planning for Self-Organizing Production Systems The planning complexity depends on the number of individual components that partake in the system. By reducing the planning scope to a subset of the system components, it is much easier to explore the solution space and converge to good solutions. We develop an algorithm that combines Monte-Carlo Tree Search with Message Passing approaches known from Belief Propagation. Thereby, the system can be compartmentalized into individual agents who are coordinating their actions. The agent coordination mechanism is shown to improve the planning solution quality compared to uncoordinated individual planning. The same implementation of our novel planning algorithm can be used the solve hitherto separate planning and optimization concerns from all levels of the automation hierarchy.

- Combines MCTS with Message-Passing algorithms originally developed for Belief Propagation in Probabilistic Graphical Models.

- The agents need to simulate only a small portion of the overall system.

- The considered benchmark problems include distributed supply chain orchestration by independent agents and the distributed maneuver planning of autonomous vehicles.

Chapter 5: Modeling of Production Skills In order to achieve flexibility in automation with runtime planning, an accurate system model is required. It can be quite resource-intensive to keep the physical system and its model representation synchronized. Especially if new manufacturing operations are frequently introduced in a flexible production environment. To reduce this effort, higher-level descriptions are developed from which low-level representations for runtime control are generated. We introduce a formal model for the technical skills of components in an automated system based on semantic modeling and deductive inference based on second-order logic.

The modeled skills are used to generate executable action-representations for specific operations. The generation of executable actions take in as input the capabilities of the system components, the topological layout of the system and a description of the requested operation. The capability model is therefore used as a high-level descriptive language that can be compiled to executable actions for the participating system components. The generated action representations contain all preconditions and effects required for detailed planning.

Already published results that were created in preparation for this thesis are:

- Julius Pfrommer, Miriam Schleipen, and Jürgen Beyerer. “Fähigkeiten adaptiver Produktionsanlagen”. In: *atp-edition* 55 (11) (2013)

- Julius Pfrommer, Miriam Schleipen, and Jürgen Beyerer. “PPRS: Production skills and their relation to product, process, and resource”. In: *Proceedings of the 2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*. IEEE. Cagliari, Italy, 2013

- Julius Pfrommer, Denis Stogl, Kiril Aleksandrov, Viktor Schubert, and Björn Hein. "Modelling and Orchestration of Service-Based Manufacturing Systems via Skills". In: *Emerging Technologies & Factory Automation (ETFA), 2014 IEEE 19th Conference on*. Barcelona, Spain, Sept. 2014

- Julius Pfrommer, Denis Stogl, Kiril Aleksandrov, Stefan Escaida Navarro, Björn Hein, and Jürgen Beyerer. "Plug & produce by modelling skills and service-oriented orchestration of reconfigurable manufacturing systems". In: *at-Automatisierungstechnik* 63.10 (2015), pp. 790–800

- Selma Azaiez, Michael Boc, Loic Cudennec, Max Da Silva Simoes, Jens Haupert, Selma Kchir, Xenia Klinge, Wael Labidi, Karima Nahhal, Julius Pfrommer, Miriam Schleipen, Christian Schulz, and Thibaud Tortech. "Towards Flexibility in Future Industrial Manufacturing: A Global Framework for Self-organization of Production Cells". In: *Procedia Computer Science* 83 (2016), pp. 1268–1273

- Julius Pfrommer, Sten Grüner, Thomas Goldschmidt, and Dirk Schulz. "A common core for information modeling in the Industrial Internet of Things". In: *at-Automatisierungstechnik* 64.9 (2016), pp. 729–741

- Sten Grüner, Julius Pfrommer, and Florian Palm. "RESTful Industrial Communication With OPC UA". in: *IEEE Transactions on Industrial Informatics* 12.5 (2016), pp. 1832–1841

- Julius Pfrommer, Miriam Schleipen, Selma Azaiez, Michael Boc, and Xenia Kling. "Deploying software functionality to manufacturing resources safely at runtime". In: *Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference on*. Berlin, Germany: IEEE, Sept. 2016, pp. 1–7

- Julius Pfrommer. "Graphical Partially Observable Monte-Carlo Planning". In: *Workshop on Learning, Inference and Control of Multi-Agent Systems, Conference on Neural Information Processing Systems (NIPS)*. Dec. 2016

The following publications are our technical reports or publications in adjacent fields. They contributed to the thesis indirectly by using similar techniques or by working on implementation technologies in industrial automation (e.g. using technologies such as OPC UA and AutomationML) that are relevant for bringing the results of this thesis into practice.

- Julius Pfrommer. *Towards Graphical Partially Observable Monte-Carlo Planning*. Tech. rep. 2016, pp. 113–125
- Julius Pfrommer. *Distributed Constraint Optimization over Constrained Communication Topologies*. Tech. rep. 2015, pp. 77–87
- Julius Pfrommer. *Information and Control in Cyber-Physical Production Systems*. Tech. rep. 2014, pp. 61–74
- Julius Pfrommer, Clemens Zimmerling, Jinzhao Liu, Luise Kärger, Frank Henning, and Jürgen Beyerer. "Optimisation of manufacturing process parameters using deep neural networks as surrogate models". In: *Proceedings of the 51st CIRP Conference on Manufacturing Systems*. Stockholm: CIRP, 2018
- Julius Pfrommer. "Semantic Interoperability at Big-Data Scale with the open62541 OPC UA Implementation". In: *2nd International Workshop on Interoperability and Open-Source Solutions for the Internet of Things (InterOSS-IoT)*. Stuttgart, Germany, Nov. 2016
- Julius Pfrommer, Sten Grüner, and Florian Palm. "Hybrid OPC UA and DDS: Combining architectural styles for the industrial internet". In: *Factory Communication Systems (WFCS), 2016 IEEE World Conference on*. Aveiro, Portugal: IEEE, May 2016, pp. 1–7
- Thomas Usländer, Julius Pfrommer, and Miriam Schleipen. "Das Internet der Dinge in der Automation - Anforderungen und Technologien". In: *5. Jahreskolloquium "Kommunikation in der Automation" (KommA 2014)*. Lemgo, 2014
- Julius Pfrommer, Miriam Schleipen, Thomas Usländer, Ulrich Epple, Roland Heidel, Leon Urbas, Olaf Sauer, and Jürgen Beyerer. "Begrifflichkeiten um Industrie 4.0 – Ordnung im Sprachwirrwarr". In: *Tagungsband zu Entwurf komplexer Automatisierungssysteme (EKA) 2014*. Ed. by Ulrich Jumar and Christian Diedrich. Magdeburg, May 2014

- Julius Pfrommer, Joseph Warrington, Georg Schildbach, and Manfred Morari. “Dynamic vehicle redistribution and online price incentives in shared mobility systems”. In: *IEEE Transactions on Intelligent Transportation Systems* 15.4 (2014), pp. 1567–1578

2 A Model of Concurrent Production Systems

> *The value of [Lagrange's book "Mécanique Analytique"] consists in the exposition of a general method by which every mechanical question may be stated in a single algebraic equation. The entire history of any mechanical system, as for example, the solar system, may thus be condensed into a single sentence.*
>
> *Robert S. Woodward [Woo95]*

The chapter introduces a model for production systems that combines continuous and discrete system dynamics with concurrency, i.e. parallel operations and the synchronization of system components. The model is intended as the basis for a "theory of production systems". For this, the model needs to able to represent the system dynamics on all levels of the control hierarchy. Examples from different levels of the control hierarchy are used for demonstration and to substantiate this claim. The core postulate of this chapter is the following:

> *The same set of modeling principles can represent the relevant properties of production systems on all levels of the control hierarchy.*

2.1 State, Actions and Action Sequences

Definition 2.1. *A system is a set of components $c \in C$.*

Examples for components on a manufacturing shopfloor are machines and logistics equipment, such as forklifts. Depending on the level of abstraction of the model, components can also represent parts inside a machine, as well as entire production plants and warehouses. Assume for now that components do not contain other components in a hierarchy.

Definition 2.2. *Each component c has a state $s \in S_c$. The set S_c contains all possible states of the component c.*

The global state space $S = \times_{c \in C} S_c$ is the cartesian product of the possible states for every component. A global state is a vector $\boldsymbol{s} \in S$ with elements s_c. The state space of a subset of the components $Q \subseteq C$ is $S_Q = \times_{c \in Q} S_c$. The projection $\Pi_Q(\boldsymbol{s}) = (s_c : c \in Q)$ extracts the state of the components Q from the global state. In general, a subscript denoting a set of components indicates projection and $\boldsymbol{s}_Q = \Pi_Q(\boldsymbol{s})$. The inverse projection is $\Pi_Q^{-1}(\boldsymbol{s}_Q) = \{\boldsymbol{u} \in S : \Pi_Q(\boldsymbol{u}) = \boldsymbol{s}_Q\}$.

Example 2.1. Consider the ABB IRB140 robotic manipulator from Figure 2.1. The green sphere represents the Tool Center Point (TCP). The TCP has six degrees of freedom (three each for translation and rotation). But some positions are not reachable due to the physical constraints of the manipulator. The constraints are encoded in the set of reachable positions $\Psi \subset \mathbb{R}^6$. In addition, the IRB140 can be fitted with different tools. In this example, the possible tools are $\Upsilon = \{\texttt{gripper}, \texttt{welder}, \texttt{drill}, \texttt{none}\}$. The overall state space of the manipulator is $S_{\texttt{irb140}} = \Psi \times \Upsilon$.

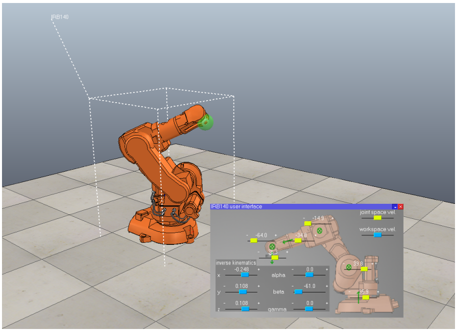


Figure 2.1: IRB140 robotic manipulator with inverse kinematics control. The simulation environment V-Rep and the model of the IRB140 robotic manipulator is courtesy of Coppelia Robotics (`http://www.coppeliarobotics.com/`).

Many manufacturing operations define trajectories where the manipulator starts and finishes at fixed locations. As an alternative to modeling a continuous state space, one can limit the possible positions to a discrete set of predefined positions. A continuous configuration space for the position is however better suited to model the physical movement dynamics of the robot.

Components can change their state over time. The following nomenclature is taken from [Fuj98]: *Physical time* refers to time in the physical system that is represented by the model. *Simulation time* refers to a time representation, for example a real value that corresponds to a physical clock by scaling and an epoch-date for the origin. *Wallclock time* refers to the time when the simulation is executed. In distributed systems, it is generally impossible to assign an absolute order to events [Lam78]. We make the simplifying assumption that clocks are synchronized to absolute precision. So the physical time of the system components is always identical. In the model representation, however, the simulation time can differ between components. That is, components can evolve their state independently from one another until synchronization forces their simulation time to coincide again. If not stated otherwise, time refers to simulation time in the remainder of the text.

Definition 2.3. *The time-indexed state of a component c is*

$$\sigma = (s, t) \in \Sigma_c, \quad \Sigma_c = S_c \times \mathbb{R} \,. \tag{2.1}$$

It indicates the state of the component s at time t. The time t represents the offset from some epoch-date in seconds.

The time-indexed state of the entire system is $\boldsymbol{\sigma} \in \Sigma$ for $\Sigma = \times_{c \in C} \Sigma_c$. In the context of a system state $\boldsymbol{\sigma}$, the time-indexed state of a component c is referred to as $\sigma_c = (s_c, t_c)$. Again, subscript-based notation for projection refers to the definitions from the surrounding context: Σ_Q denotes the joint time-indexed state-space for a subset of the components $Q \subseteq C$ and $\boldsymbol{\sigma}_Q = \Pi_Q(\boldsymbol{\sigma})$ is the projection of a global time-indexed state into Σ_Q.

Components change their time-indexed state by executing actions. In the model, actions skip the time-indexed state ahead to the time after the execution. Components have no well-defined state during the execution of an action. If the model is linked to a physical instance of the system, then the relation between simulation time and physical time is as follows. If the time t_c of some component c is earlier or equal to the physical time, then the component has been idle since t_c and is immediately available. If t_c is later than the physical time, then the component is occupied with the execution of one or more actions until t_c.

Definition 2.4. *Actions are possible state transitions, represented as a tuple*

$$a = \big(C_a, \bar{\Sigma}_a, \boldsymbol{e}_a, d_a\big) \ . \tag{2.2}$$

The tuple describes the action via its participating components, preconditions and effects. It consists of

- *the participating components $C_a \subseteq C$,*
- *the feasible initial time-indexed states of the participating components $\bar{\Sigma}_a \subseteq \Sigma_{C_a} = (\times_{c \in C_a} \Sigma_c)$,*
- *the action effect $\boldsymbol{e}_a : \bar{\Sigma}_a \to S_{C_a}$, where $e_{a,c}$ is the effect for just the participating component $c \in C_a$, and*
- *the action duration $d_a : \bar{\Sigma}_a \to \mathbb{R}_+$.*

The set of feasible time-indexed initial states $\bar{\Sigma}_a$ encodes the preconditions of the action a for the participating components. The shorthand notation for the state of the participating components is $\boldsymbol{\sigma}_a = \Pi_{C_a}(\boldsymbol{\sigma})$. The feasible global initial states for the action a are $\Sigma_a = \{\boldsymbol{\sigma} \in \Sigma : \boldsymbol{\sigma}_a \in \bar{\Sigma}_a\}$. Every action is an operator on the time-indexed global state $a : \Sigma_a \to \Sigma$.

From the initial global state $\boldsymbol{\sigma}$ it follows that the earliest possible starting time is $t_a^{\text{start}}(\boldsymbol{\sigma}_a) = \max_{c \in C_a} t_c$ for the action a. Let $\boldsymbol{\sigma}' = a(\boldsymbol{\sigma})$ denote the global state following the execution. The new global time-indexed state is then comprised of elements

$$\sigma'_c = \begin{cases} \Big(e_{a,c}(\boldsymbol{\sigma}_a),\ t_a^{\text{start}}(\boldsymbol{\sigma}_a) + d_a(\boldsymbol{\sigma}_a)\Big), & \text{if } c \in C_a \\ \sigma_c, & \text{else}\,. \end{cases} \tag{2.3}$$

Equation 2.3 defines the operational semantics of all actions. It implies the *Markov assumption*: The outcome of an action depends only on the previous system state. Components that are not participating in an action do not partake in the preconditions and their state is left unchanged as well.

Equation 2.3 defines how components with different simulation times are synchronized: The participating component with the highest simulation time defines the starting time of the action. The other participating components idle until they join the execution.[1] The feasible initial states Σ_a can also encode preconditions for the component timing. Consider a physical system whose dynamics is described by a differential equation. Such systems typically cannot idle without any effect on their state. To adequately cover components that cannot idle in the sense of Equation 2.3, the preconditions of Σ_a can require identical simulation times for all participating components $\forall \boldsymbol{\sigma}_a \in \Sigma_a, \forall (c, c') \in C_a \times C_a,\, t_c = t_{c'}$. Alternatively, as the effect function $\boldsymbol{e}_a$ takes the time-indexed state of all participating components as input, the state evolution of the individual components up until the starting time of a can be modeled as part of the effect function.

In discrete manufacturing, the major concern is the movement and transformation of products within the system. Many modeling approaches represent products as objects with individual lifecycles [Sal+10] or even as independent agents who negotiate and make independent decisions [KBT17]. To model products as individual objects, they could be represented as system components c. But, in this text, we follow a different approach that enables algorithmic improvements for planning later on.

Let $\rho \in P$ denote the set of different product types. A product type not only represents marketable output, but also raw material and semi-finished work-pieces that occur between production steps. Products are not considered individually. So products of the same product type are indistinguishable. Instead of representing each component individually, it suffices to track the number of products of each product type present at every component. A product is always contained in some component. Take for example a workpiece mounted inside a machine, a crate of material sitting on a forklift, or a finished product stored in a high bay warehouse. The following definition is a special case of Definition 2.2 for the component state.

[1] This corresponds to the use of the popular Max-Plus algebra to describe the time-evolution of discrete event systems [Bac+92].

Definition 2.5. *Some components can physically contain products. In that case, the component state decomposes into the component configuration $\xi \in \Xi_c$ and a vector $\boldsymbol{p}$ for the number of contained products for each product type.*

$$s = (\xi, \boldsymbol{p}) \in S_c, \quad S_c \subseteq \left(\Xi_c \times \mathbb{N}_0^{|P|}\right).$$

For convenience, we denote a vector with just a single product of type ρ as the basis-vector $\boldsymbol{\nu}_\rho$. The number of products of type ρ in component c is $(p_c)_\rho$. A component c with no contained products has $\boldsymbol{p}_c = \mathbf{0}$, where $\mathbf{0}$ denotes the null-vector of appropriate dimensionality.

Example 2.2. This example introduces a minimal manufacturing scenario that will be used throughout this text. The scenario, shown in Figure 2.2, is comprised of a machine tool (`mt`) that mills piston rods out of steel bars, a lattice box (`box`) and a robotic manipulator that packages the piston rods for transport (`manip`).

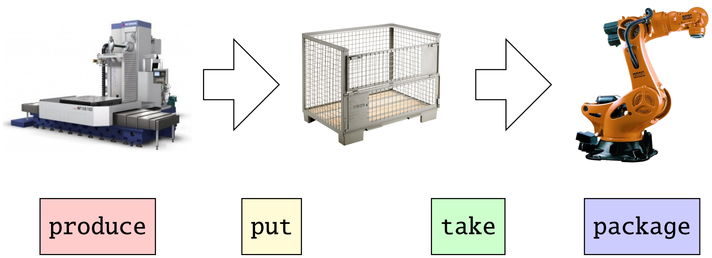


Figure 2.2: Minimal scenario for discrete manufacturing

Four actions have been defined for the scenario. The actions `produce` and `package` have only one participating components, the machine tool and the manipulator respectively. The actions `put` and `take` require the participation of two components to model the transition of a product between them. We do not consider single piston rods, but only orders of 100 parts with the product type `order`. Every component can only hold a single order-product at once. Therefore, once the machine tool has completed an order, the parts need to be put into the lattice box before the next order can be produced. The manipulator then has to take the parts of the previous order out of the lattice box before the machine tool can put in the next. Products "appear" and "disappear" when they leave the scope of the modeled system.

The complete definition of the action `put` is as follows:

- $C_{\texttt{put}} = \{\texttt{mt}, \texttt{box}\}$
- $\bar{\Sigma}_{\texttt{put}} = \{\boldsymbol{\sigma}_{\texttt{put}} \in \Sigma_{C_{\texttt{put}}} : \boldsymbol{p}_{\texttt{mt}} = \boldsymbol{\nu}_{\texttt{order}},\ \boldsymbol{p}_{\texttt{box}} = \mathbf{0}\}$
- $\boldsymbol{e}_{\texttt{put}}(\boldsymbol{\sigma}_{\texttt{put}}) = ((\xi_{\texttt{mt}}, \mathbf{0}), (\xi_{\texttt{box}}, \boldsymbol{\nu}_{\texttt{order}}))$
- $\boldsymbol{d}_{\texttt{put}}(\boldsymbol{\sigma}_{\texttt{put}}) = (5s, 5s)$

Actions can be chained to form action sequences. The set of actions A is the alphabet for the free monoid (Kleene Star, [HU79]) A^* which contains all words (sequences) of finite length. The empty sequence is written as ε. The sequence elements w^k are indexed according to their position in the sequence $\boldsymbol{w} = w^1 w^2 \ldots w^{|\boldsymbol{w}|}$.

Definition 2.6. *An action sequence $\boldsymbol{w} \in A^*$ is itself an action resulting from the composition of the constituent actions.*

For a valid initial state $\boldsymbol{\sigma} \in \Sigma_{\boldsymbol{w}}$ (defined in the next paragraph), the resulting state after the execution of $\boldsymbol{w}$ is $\boldsymbol{w}(\boldsymbol{\sigma}) = (w^{|\boldsymbol{w}|} \circ \cdots \circ w^1)(\boldsymbol{\sigma})$ according to the dynamics of action execution from Equation (2.3). The multiplication notation for sequences is preferred to the $\circ$-notation for function composition to keep the notation light, to write the sequence elements in-order, and because we will take on an algebraic perspective on composition later on. Note that, due to the way concurrency is represented with the time-indexing of component states, an action at a later sequence index may actually start at an earlier simulation time than one of its predecessors in the sequence. But this is possible only if the two actions do not share participating components.

The subsequence of the first k elements is $\boldsymbol{w}^{:k} = \prod_{l=1}^{k} w^l$. The subsequence starting at the kth element is $\boldsymbol{w}^{k:} = \prod_{l=k}^{|\boldsymbol{w}|} w^l$. The domain of $\boldsymbol{w}$ as an operator is $\Sigma_{\boldsymbol{w}} = \{\boldsymbol{\sigma} \in \Sigma : \forall k \in \{1, \ldots, |\boldsymbol{w}|\},\ \boldsymbol{w}^{:k-1}(\boldsymbol{\sigma}) \in \Sigma_{w^k}\}$. This ensures that the preconditions of all actions are satisfied when the sequence is executed in order. The composition of action operators to a sequence is always possible. But the resulting sequence might be infeasible with an empty domain $\Sigma_{\boldsymbol{w}} = \varnothing$.

The set of all action sequences, from now on denoted as $W = A^*$, implies a tree-graph. In the context of an initial system state $\boldsymbol{\sigma}$, infeasible sequences are removed from the tree. The feasible sequences starting at $\boldsymbol{\sigma}$ are $W^{\boldsymbol{\sigma}} = \{\boldsymbol{w} \in A^* : \boldsymbol{\sigma} \in \Sigma_{\boldsymbol{w}}\}$. Obviously, the pruned tree is a subset $W^{\boldsymbol{\sigma}} \subseteq W$ and is still a tree since for every sequence $\boldsymbol{w} \in W^{\boldsymbol{\sigma}}$ all subsequences of the first $k \in \{0, \dots, |\boldsymbol{w}|\}$ elements are also contained $\boldsymbol{w}^{:k} \in W^{\boldsymbol{\sigma}}$. After the first action a has been executed, the system states becomes $a(\boldsymbol{\sigma})$ and the sequence tree becomes $a(W) = \{\boldsymbol{w} : a\boldsymbol{w} \in W\}$.

Example 2.3. This example extends the minimal manufacturing scenario from Example 2.2. Assume an initial system state $\boldsymbol{\sigma}$ where no component contains any products. The four defined actions with their preconditions and effects yield a sequence tree $W^{\boldsymbol{\sigma}}$ of feasible sequences. All feasible sequences up to five actions are shown in Figure 2.3.

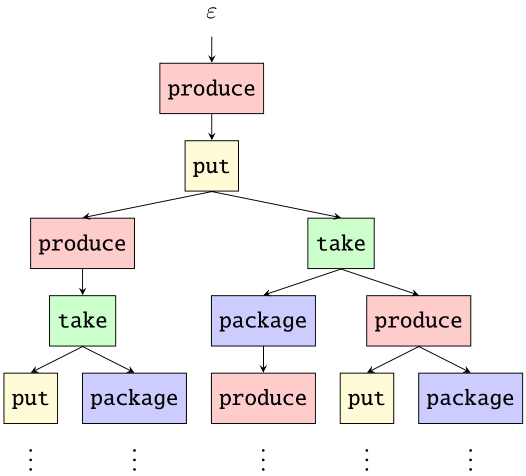


Figure 2.3: Sequence tree for Example 2.2.

Note that the order of the actions in the sequence do not require that the start times of the actions have the same ordering. Consider the action sequence $($`produce` `put` `produce` `take`$)$. The last two actions `produce` and `take` can start at the same time. It is even possible that an action occurs later in a sequence but starts before a preceding action in terms of absolute time.

k	**Machine Tool**	**Box**	**Robot**
0	$(s = 0, t = 0)$	$(s = 0, t = 0)$	$(s = 0, t = 0)$
	`produce`		
1	$(s = 1, t = 5)$	$(s = 0, t = 0)$	$(s = 0, t = 0)$
	`put`	`put`	
2	$(s = 0, t = 10)$	$(s = 1, t = 10)$	$(s = 0, t = 0)$
	`produce`		
3	$(s = 1, t = 15)$	$(s = 1, t = 10)$	$(s = 0, t = 0)$
		`take`	`take`
4	$(s = 1, t = 15)$	$(s = 0, t = 15)$	$(s = 1, t = 15)$

Table 2.1: Evolution of the system state for the sequence `produce`, `put`, `produce`, `take`. The component state s is simplified and describes only the number of products currently at the component. All actions are assumed to have a duration of five time units.

Action sequences can be the result of a planning procedure that are executed only once for the current situation. But sequences can also be used as reusable macro-actions. Many manufacturing systems are organized in lines, within which work-pieces undergo a fixed sequence of actions. In Example 2.3, the sequence (`produce` `put` `take` `package`) could be such a macro action. System parts with little flexibility can thus be modeled with comparatively few macro-actions. These can be integrated seamlessly with more fine-grained actions where more behavioral flexibility is required.

2.2 Parameterized Actions

The action definition from Section 2.1 has been accompanied by examples from discrete manufacturing. In the process industry (e.g. chemicals, pharmaceuticals, food and beverages), many decisions have to be made on a continuous domain. In principle, the set of actions A could represent continuous decisions with an uncountably infinite set of actions. But then we could no longer maintain explicit representations of every action in computer-based simulations. Instead, we allow actions to be parameterized.

Definition 2.7. *The preconditions, effects and durations of a parameterized action a depend on the choice of action parameter $\theta \in \Theta_a$.*

$$a^\theta = (C_a, \bar{\Sigma}_a^\theta, \boldsymbol{e}_a^\theta, d_a^\theta) \tag{2.4}$$

There are no particular restrictions on possible parameter spaces Θ_a. Of course, parameters can also be vectorial or sets, even though we use scalar notation for parameters in general. Actions a that do not define parameters have $\Theta_a = \varnothing$. In that case, the parameter can be omitted in the notation. The following list gives examples for parameters on different scales of measure and sizes of the parameter space.

Nominal and Ordinal Parameters In the simplest case, parameterized actions simply group actions that, in some sense, belong together. For example the action `paint` with categorical parameters $\Theta_{\texttt{paint}} = \{\texttt{blue}, \texttt{green}\}$. There can also be a natural order among the parameters on an ordinal scale, such as $\{\texttt{cold}, \texttt{warm}, \texttt{hot}\}$.

Discrete Parameters Discrete parameters (on an interval or ratio scale) can indicate for example the number of repetitions of an action. When 5,000 parts of a certain product are needed, this can then be achieved by an action sequence with appropriate parameters in much less than 5,000 sequence entries.

Continuous Parameters Process control (in the sense of control theory [Lib11]) usually makes decisions about continuous control values at every considered point in time. An example for this is setting a continuous voltage for an electric motor.

Vectorial Parameters An action can take a vector or some other structured mathematical object for its parameter. We continue to write action parameters as a scalar θ, even though it may be vectorial.

Example 2.4. Take the example of a storage tank in a process control setting. The tank can be filled with liquid and drained afterward. The fill level is controlled by a pump and a valve at the bottom of the tank.

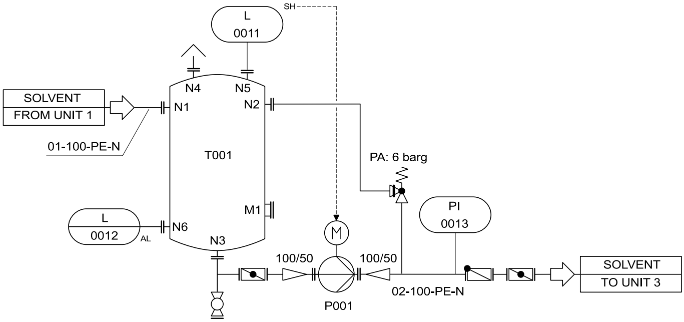


Figure 2.4: Piping and instrumentation diagram (P&ID) of a storage tank. (Reproduced with permission from `https://commons.wikimedia.org/wiki/File:Pump_with_tank_pid_en.svg`.)

Consider the action `drain` acting on the tank. Let $v \in [0, 1]$ be the control value for the valve and τ the duration of the action with $\theta_{\texttt{drain}} = (v, \tau)$. The valve is closed for $v = 0$ and fully open for $v = 1$. In addition, the liquid flow depends on the pressure at the valve and hence the fill level described by the state of the tank $s_{\mathtt{tank}} \in \mathbb{R}_{+}$. The flow of the (incompressible) fluid out of the tank can then be described by a differential equation $\dot{s}_{\mathtt{tank}} = g(s_{\mathtt{tank}}, v)$ according to Bernoulli's principle [Ber38] and $s_{\mathtt{tank}}(t_0)$ is known from the initial state $\boldsymbol{\sigma}_{\mathtt{drain}}$. The effect of the action drain on the tank is

$$e^{(v,\tau)}_{\mathtt{drain}}(\boldsymbol{\sigma}_{\mathtt{drain}})_{\mathtt{tank}} = s_{\mathtt{tank}}(t_0) + \int_{t=t_0}^{t_0+\tau} g(s_{\mathtt{tank}}(t), v)dt\,.$$

With the introduction of parametric actions, problems from control theory can be represented in the model. In Model Predictive Control (MPC) [ML99], a dynamical system is approximated by discretizing the time domain. The problem of optimal control is then posed as an explicit optimization problem that selects control values for every time period. By taking the (continuous) control decisions as action parameters, optimal control problems can be represented with a single action that encompasses all system components. Control with mixed discrete-continuous control values can be represented by either a more complex parameter-space Θ or by representing discrete choices with different actions. But only with several actions can aspects of concurrency be represented, where the system components are synchronized via joint participation in an action and evolve their state independently otherwise.

So far, the model presented in this chapter unifies the treatment of discrete events, concurrency and continuous system dynamics.

2.3 Uncertainty and Observations

Until now, it was implicitly assumed that actions are deterministic. But virtually no production system actually is. There is always an interaction with a stochastic environment, from logistics influenced by weather conditions to the human operator returning late from a break. In addition, even fully automated manufacturing processes are inherently stochastic. This is illustrated by the fact that very few production processes achieve zero-defects production.[2] It

[2] The most common reasons for delays and unforeseen events are, according to [VHL03], machine failure, urgent job arrival, job cancellation, due date change, shortage or delay in the arrival of

is desirable to represent this uncertainty also in the model used for control or planning. With an explicit representation of the uncertainty in the evolution of the system state, plans of higher quality can be achieved by optimizing for the expected reward. In general, uncertainty in the model is represented by belief distributions over the system state and actions with probabilistic outcomes.

This section uses the notation of [Hem66]: Underlined symbols denote random variables whose realizations follow some probability distribution. The same symbol may occur underlined and non-underlined. Here, this can be the distinction between a random variable and a sampled realization of the random variable. Another case is the distinction between an action used as an identifier and the same action used to sample stochastic state transitions.

The system state following an initial state $\boldsymbol{\sigma}$ and an action with uncertainty a is in effect the realization of a random variable $\boldsymbol{\sigma}' \sim \underline{\boldsymbol{\sigma}}' = \underline{a}(\boldsymbol{\sigma})$. The probability (density) of the possible outcomes $\boldsymbol{\sigma}' \in \Sigma$ is $\mathbb{P}(\underline{\boldsymbol{\sigma}}' = \boldsymbol{\sigma}')$. We allow the application of uncertain states with a belief distribution to actions The resulting state has a belief distribution $\mathbb{P}(\underline{a}(\underline{\boldsymbol{\sigma}}) = \boldsymbol{\sigma}') = \int_\Sigma \left[\mathbb{P}(\underline{a}(\boldsymbol{\nu}) = \boldsymbol{\sigma}')\, \mathbb{P}(\underline{\boldsymbol{\sigma}} = \boldsymbol{\nu})\right] d\boldsymbol{\nu}$. We do not consider the case where a is not applicable to the realization of the initial state $\underline{\boldsymbol{\sigma}}$.

By Equation 2.3, non-participating components $c \notin C_a$ are not affected by the action execution. Actions with uncertainty are not exempt from this rule. All state transitions where a non-participating component changes its timed-indexed state via execution of an action a must have zero probability.

$$\forall \boldsymbol{\sigma} \in \Sigma_a,\ \forall \boldsymbol{\sigma}' \in \Sigma,\ \forall c \notin C_a : \sigma_c \neq \sigma'_c \Rightarrow \mathbb{P}(\underline{a}(\boldsymbol{\sigma}) = \boldsymbol{\sigma}') = 0 \qquad (2.5)$$

If the state following the execution of an action with uncertainty is not immediately and fully knowable, we have to indirectly infer probabilities over the resulting state via incomplete or noisy observations. This is called the *partially observable* setting [KLC98]. With partial observability, the components $c \in C$ each generate observations $o_c \in O_c$. An action results in observations from the participating components $O_a = (\times_{c \in C} O_c)$. With a slight abuse of notation, the next state and the observations are both drawn by sampling the action

material, change in job priority, rework or quality problems, over- or underestimation of process time, and operator absenteeism.

$(\boldsymbol{\sigma}', o_a) \sim \underline{a}(\boldsymbol{\sigma})$. The distribution of observations conditionally depends on the action and the state transition $\mathbb{P}(o_a \mid \boldsymbol{\sigma}, a, \boldsymbol{\sigma}')$.

Obviously, for every action a, the observations o_a are conditionally independent of the state of components that do not participate in a. Otherwise, if an observation was conditionally dependent on the state of a non-participating component, then the observation could depend on the outcome of an action that occurs earlier in the action sequence but actually has a later starting time in simulation. This cannot be possible. Therefore, given the state of the participating components $\boldsymbol{\sigma}_a$, the resulting state of the participating components and the observations are conditionally independent of the non-participating components.

$$(\boldsymbol{\sigma}'_a, o_a \mid \boldsymbol{\sigma}_a) \perp \boldsymbol{\sigma}_{C \setminus C_a} \tag{2.6}$$

Now we extend the treatment of uncertainty to action sequences. We write $\Theta = \cup_{a \in A} \Theta_a$ for the set of possible action parameters and $O = \cup_{a \in A} O_a$ for the set of possible observations. The action-subscript for parameters and observations is dropped when the relation is clear from context.

Definition 2.8. *A history $\boldsymbol{h}$ is a sequence of episodes h^k. Each episode consists of the selected action and action parameters, as well as the generated observation.*

$$\boldsymbol{h} = \underbrace{a^1\theta^1o^1}_{h^1}\underbrace{a^2\theta^2o^2}_{h^2}\dots\underbrace{a^{|\boldsymbol{h}|}\theta^{|\boldsymbol{h}|}o^{|\boldsymbol{h}|}}_{h^{|\boldsymbol{h}|}}$$

The set of possible histories $H \subset (A \times \Theta \times O)^*$ implies a tree-graph. The histories in H have finite length. Either because the scenario is *done* when a specific state is reached or by a cutoff at a maximum history depth. A history $\boldsymbol{h}$ can be appended with the next action a, parameters θ and observation $\boldsymbol{o}$ to form $\boldsymbol{h}' = \boldsymbol{h}a\theta\boldsymbol{o}$. In the case of a partially observable system, the current system state can only be inferred with uncertainty. Every history yields a probability distribution for the belief over the final system state that is conditioned on the observations. Recall that $\tilde{a}$ denotes the already parameterized actions.

$$\boldsymbol{\sigma}' \sim \underline{\boldsymbol{h}}(\boldsymbol{\sigma}),\ \underline{\boldsymbol{h}}(\boldsymbol{\sigma}) = \mathbb{P}\left(\left({a^1}^{\theta^1} \cdots {a^{|\boldsymbol{h}|}}^{\theta^{|\boldsymbol{h}|}}\right)(\boldsymbol{\sigma}) \,\middle|\, o^1, \dots, o^{|\boldsymbol{h}|}\right) \tag{2.7}$$

The computation of the belief distribution for the resulting state can be performed with iterative Bayes updates [Jay03] for the intermediary system state between actions. The state belief distribution following from an uncertain initial state is $\underline{\boldsymbol{h}}(\underline{\boldsymbol{\sigma}})$. In general, this computation cannot be performed with the available computational resources. The planning algorithms from the later chapters therefore rely on samples from forward simulations of the system only.

2.4 Reward and Policies

So far, only the dynamics of actions and action sequences were discussed. Now we begin to express preferences between action sequences.

Definition 2.9. *Executing action a with parameters θ inducing a transition of the system state from $\boldsymbol{\sigma}$ to $\boldsymbol{\sigma}'$ generates a reward $r(\boldsymbol{\sigma}, a, \theta, \boldsymbol{\sigma}')$. The reward function is*

$$\mathfrak{r} : \Sigma \times A \times \Theta \times \Sigma \to \mathbb{R}\,. \tag{2.8}$$

With the reward function, each state defines a planning problem with the goal to maximize the future (expected) reward. This is known as the decision-theoretic planning problem [De 70; BDH99]. The following hierarchy of planning problems is distinguished in the literature [LaV06]:

1. Deterministic sequential decision making in deterministic systems is simply known as the planning problem. It results in a fixed sequence of actions.

2. Sequential decision making under uncertainty with full observability is known as the Markov Decision Problem (MDP, [Put94]).

3. The partially observable stochastic case is known as Partially-Observable Markov Decision Problem (POMDP) in the literature [SS73; KLC98]. The decision maker does not have access to a full description of the system state. He can only indirectly infer a belief distribution over the system state based on incomplete or noisy observations.

The application of decision-theoretic planning for the (feedback) control of dynamical systems is discussed in [DW91; Ber+95; BG01]. See [LP12] for an application of MDP solvers to model manufacturing scenarios under uncertainty.

Algorithms for solving decision-theoretic planning problems belong to two distinct groups, online planning algorithms and policy constructing algorithms.

Online planning algorithms are executed for the selection of the next action [Ros+08; SV10] at runtime. Policy constructing algorithms are executed ahead of time. They compute a fixed policy function that takes the current system state (or observed history) to the next action [KHL08].

Definition 2.10. *A policy is a mechanism to select the next action during runtime. In fully observable settings, policies are represented as functions $\pi : \Sigma \to A$ that map from the current system state to the next action. The policy function becomes $\pi : \Sigma \to A \times \Theta$ in settings with parameterized actions.*

In POMDP, policies $\pi : H \to A$ or $\pi : H \to A \times \Theta$ map from the observed history to the next action. Internally, the POMDP policy may decide the next action and parameters based on a belief distribution over the current system state conditioned on the observed history $\mathbb{P}(\boldsymbol{\sigma} \in \Sigma \mid \boldsymbol{h} \in H)$.

For simplicity of the exposition, consider fully observable settings without parameterized actions. For a fixed policy π and a discount rate $\gamma \in [0, 1)$, the value of an initial state $\boldsymbol{\sigma}^0$ is the expected discounted reward.[3]

$$\mathfrak{v}^{\pi}(\boldsymbol{\sigma}^0) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k \mathfrak{r}(\boldsymbol{\sigma}^k, a^k, \boldsymbol{\sigma}^{k+1}) \,\middle|\, a^k = \pi(\boldsymbol{\sigma}^k),\ \boldsymbol{\sigma}^{k+1} \sim a^k(\boldsymbol{\sigma}^k)\right] \tag{2.9}$$

As an alternative to the discount factor, we can limit the number of considered periods. The optimal value of a state $\mathfrak{v}$ is the expected reward resulting from optimal action selection in every step. Based on Bellman's principle of optimality [Bel57], the V-value can be written as a recursive formula.

$$\mathfrak{v}(\boldsymbol{\sigma}) = \max_{a \in A} \mathop{\mathbb{E}}_{\boldsymbol{\sigma}' \sim a(\boldsymbol{\sigma})} \left[\mathfrak{r}(\boldsymbol{\sigma}, a, \boldsymbol{\sigma}') + \gamma \mathfrak{v}(\boldsymbol{\sigma}')\right] \tag{2.10}$$

The corresponding Q-value is the expected reward for selecting action a in state $\boldsymbol{\sigma}$ and optimal decision making thereafter.

$$\mathfrak{q}(\boldsymbol{\sigma}, a) = \mathop{\mathbb{E}}_{\boldsymbol{\sigma}' \sim a(\boldsymbol{\sigma})} \left[\mathfrak{r}(\boldsymbol{\sigma}, a, \boldsymbol{\sigma}') + \gamma \mathfrak{v}(\boldsymbol{\sigma}')\right] \tag{2.11}$$

[3] The V-value and Q-value are longstanding terms in the literature. As both are scalar values we denote them as $\mathfrak{v}$ and $\mathfrak{q}$ in the mathematical notation.

Selecting a value according to $\max_{a \in A} \mathfrak{q}(\boldsymbol{\sigma}, a)$ is the optimal policy. But computing the optimal policy is generally computationally intractable. The remainder of this thesis is concerned with ways of rendering optimization and planning in the framework introduced in this chapter computationally feasible. This is achieved a) with tailored planning algorithms, b) so-called rollout policies that exploit known structure in the planning problem and c) the decomposition of the planning problem into a coupled set of smaller problems that are jointly optimized by cooperating agents.

3 Simulation-Based Planning for Concurrent Production Systems

> *Programming, or program planning, may be defined as the construction of a schedule of actions by means of which an economy, organisation, or other complex of activities may move from one defined state to another, or from a defined state toward some specifically defined objective.*
>
> *Marshal K. Wood and George B. Dantzig [WD51]*

The model from Chapter 2 is generic and can be used to represent many types of systems. Very little constraints are imposed on the system dynamics that can be represented. This chapter develops an algorithm for sequential decision making that does not make additional limiting assumptions. But this richness with respect to possible system dynamics is a drawback when it comes to planning. Most planning algorithms impose a much more limited model structure they exploit to reduce the computational effort. The core postulate of this chapter is the following:

> *The same algorithm can be used for planning and runtime control on all levels of the control hierarchy – ranging from continuous dynamics of a physical system to global supply-chain operations – and for both continuous and discrete production.*

This chapter develops a planning algorithm for the full model from Chapter 2 without additional assumptions. In the first two sections, two techniques are used to reduce the number of action sequences that are visited for planning with action sequences in deterministic scenarios. In Section 3.1 the search tree is explicitly pruned by removing equivalent action sequences (this will have a precise definition). Section 3.2 further speeds up planning by implicitly pruning less promising parts of the search tree via Monte-Carlo Tree Search (MCTS). Section 3.3 extends planning to the full model with parametric actions and uncertainty with partial observability. In order to scale to larger scenarios, Section 3.4 develops a custom rollout policy that uses a relaxation of the planning problem to a Mixed-Integer Linear Program (MILP).

3.1 Tree Search with Backtracking

The model from Chapter 2 can represent concurrency, i.e. parallelism and the synchronization of system components. A consequence of the model is that many action sequences are equivalent with respect to their overall preconditions and effects.

Example 3.1. Take the scenario from Example 2.2 with the modification that the robotic manipulator initially contains a product. The actions `produce` and `package` do not share a participating component as $C_{\texttt{produce}} = \{\texttt{mt}\}$ and $C_{\texttt{package}} = \{\texttt{manip}\}$. It is easy to see that the sequence $(\texttt{produce}\ \texttt{package})$ is equivalent to the sequence $(\texttt{package}\ \texttt{produce})$ in terms of their preconditions and effects. In that sense, the two actions are independent from each other.

This notion of action sequence equivalence is made rigorous based on established results from Trace Theory (TT) [CF69; Maz77]. The theoretical prerequisites are summarized in the following. Full proofs can be found in [DM97a]. Some elementary definitions from Group Theory [DF04] are assumed as known.

3.1.1 Background: Trace Theory

Previous nomenclature from Chapter 2 applies to TT by taking A as a set of letters of action-identifiers that are concatenated to words corresponding to action sequences. Let A be a finite alphabet of letters and A^* the set of all finite-length words over A. The composition of letters forms a free monoid with generating set A and the empty word, denoted ε, as the unit element. Some letters commute and are said to be mutually independent. This is captured in the set of independence relations $Z \subseteq A \times A$. Independence relations are symmetric $(a, b) \in Z \Rightarrow (b, a) \in Z$ and irreflexive $\forall a \in A, (a, a) \notin Z$. We also write $a \perp_Z b$ to denote independence between a and b. From Z follows an equivalence relation between words $\simeq_Z$: two words $\boldsymbol{w}$ and $\boldsymbol{v}$ from A^* are equivalent according to $\simeq_Z$ if $\boldsymbol{v}$ is a permutation of $\boldsymbol{w}$ that can be reached by successive reordering of adjacent elements that commute according to Z. In algebraic terms, the trace monoid $\mathbb{M}(A, Z)$ is the quotient of A^* by the congruence $\simeq_Z$. Its elements, called traces, are pairwise disjoint subsets of A^*. Each trace contains the words that are mutually equivalent according to $\simeq_Z$. Let $[\boldsymbol{w}]_Z$ denote the trace generated from the word $\boldsymbol{w}$. We can now rephrase equivalency as $\boldsymbol{w}$ and $\boldsymbol{v}$ generating the same trace $\boldsymbol{w} \simeq_Z \boldsymbol{v} \Leftrightarrow [\boldsymbol{w}]_Z = [\boldsymbol{v}]_Z$. When it is clear from context which independence relations apply, we simply write $[\boldsymbol{w}]$ for the trace and $\boldsymbol{w} \simeq \boldsymbol{v}$ to denote equivalence.

The independence relations define a partial order between some of the letters in a word. This partial order is a "must appear before" binary relation $\prec$.

$$w^k \prec w^l \Leftrightarrow k < l \wedge \neg(w^k \perp w^l) \tag{3.1}$$

The transformation between equivalent words $\boldsymbol{w} \simeq \boldsymbol{v}$ is described by a permutation of the element indices τ. The partial order of the word elements is invariant to this permutation and therefore

$$\forall k, l \in \{1, \ldots, |\boldsymbol{w}|\},\ w^k \prec w^l \Rightarrow v^{\tau(k)} \prec v^{\tau(l)}\,. \tag{3.2}$$

The partial order yields the same *dependence graph* and *Hasse diagram* for every word from the same trace.[1] The dependence graph $\mathcal{G} = (V, E)$ is a directed acyclic graph with nodes $V = \{w^1, \ldots, w^{|\boldsymbol{w}|}\}$ and edges $E = \{(w, v) \in V^2 : w \prec v\}$. The Hasse diagram $\mathcal{G}' = (V, E')$ is the dependence graph with redundancies removed $E' = \{(w, v) \in E : \nexists u,\ w \prec u \prec v\}$. See Figure 3.1 for an example.

[1] Graphs being equal refers to the existence of an isomorphism between the graphs that takes node labels into account.

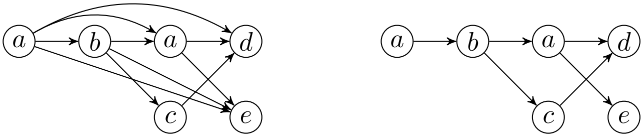


Figure 3.1: The Dependence Graph (left) and Hasse Diagram (right) for the trace $[abcaed]$ according to the trace monoid $\mathbb{M}(A, Z)$ over the alphabet $A = \{a, b, c, d, e\}$ and independencies $Z = \{(a, c), (c, a), (b, d), (d, b), (d, e), (e, d)\}$.

Besides their dependence graph and Hasse diagram, traces are uniquely represented by a word in a normal form. That is, exactly one word from every trace is in the normal form. There exist several normal forms with this property. We consider the Lexicographical Normal Form (LNF): Assume a total ordering of the letters A and a resulting lexicographical ordering over words. Longer words have a larger order than smaller words regardless of their elements. For two words $\boldsymbol{w}, \boldsymbol{v}$ of the same length, $\boldsymbol{w} < \boldsymbol{v}$ if $w^k < v^k$ for the smallest index k where $w^k \neq v^k$. A word is in LNF if it has the smallest lexicographical order among all words of the trace.

3.1.2 Tree-Search with Trace-Based Pruning

With the prerequisites in place, we now take a look back at Chapter 2. Remember that action sequences are operators on the system state with defined preconditions (the operator domain) and effects.

Definition 3.1. *Two action sequences $\boldsymbol{w}$ and $\boldsymbol{v}$ are equivalent $\boldsymbol{w} \simeq \boldsymbol{v}$ if they have the same domain of initial states $\Sigma_{\boldsymbol{w}} = \Sigma_{\boldsymbol{v}}$ and yield identical results $\forall \boldsymbol{\sigma} \in \Sigma_{\boldsymbol{w}},\ \boldsymbol{w}(\boldsymbol{\sigma}) = \boldsymbol{v}(\boldsymbol{\sigma})$.*

The independence of actions is defined based on the possibility to commute them if they occur in adjacent positions in any action sequence.

Definition 3.2. *Two actions a and b are independent $a \perp b$ if commuting them yields an equivalent sequence $\forall \boldsymbol{w}, \boldsymbol{v} \in A^*,\ \boldsymbol{wabv} \simeq \boldsymbol{wbav} \Leftrightarrow a \perp b$.*

It is generally intractable to show the equivalence of two action sequences by enumerating the set of valid initial system states and the effect of the action sequence on those states. For some action pairs however, independence can be shown without evaluating the preconditions and effects:

Proposition 3.3. *Any two actions a, b that do not share a participating component are independent.*

$$C_a \cap C_b = \varnothing \Rightarrow a \perp b \tag{3.3}$$

Proof. For all action sequences $\boldsymbol{w}, \boldsymbol{v}$ there must be $\boldsymbol{w}ab\boldsymbol{v}$ equivalent to $\boldsymbol{w}ba\boldsymbol{v}$. This follows directly from the operational semantics of actions defined in Equation 2.3. □

Example 3.2. Figure 3.2 shows which sequences from Example 2.3 are equivalent and can be pruned. Only the sequences up to a length of five are shown. The initial lexicographic ordering of the actions for the LNF is `produce` $<$ `put` $<$ `take` $<$ `package`.

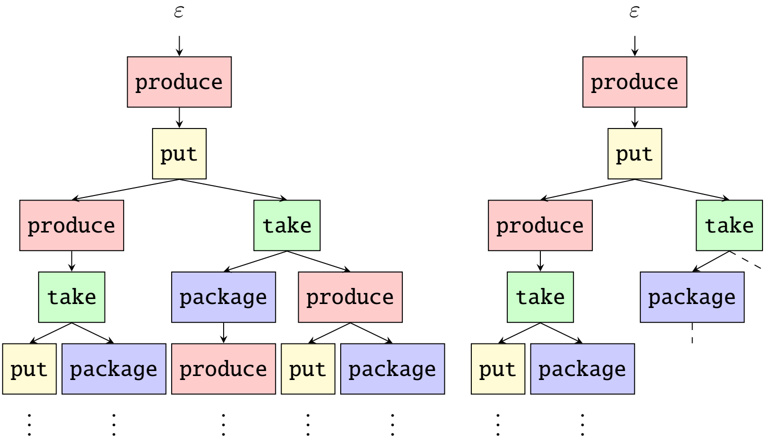


Figure 3.2: Sequence tree from Example 2.3 before (left) and after the pruning of equivalent sequences (right).

Some sequences may additionally be equivalent due to some lucky alignment of the action effects. Such additional equivalencies are not considered in this text. Now that the independence of any two actions is easily computed, we use this information to speed up the search for the best action sequence. Only one

Algorithm 1 Is the action sequence $\boldsymbol{w}a$ in LNF given that $\boldsymbol{w}$ is in LNF?

```
procedure TestLNF($\boldsymbol{w}, a$)
    for $k = |\boldsymbol{w}|, \ldots, 1$ do
        if $C_{w^k} \cap C_a \neq \varnothing$ then
            return true
        else if $w^k > a$ then
            return false
    return true
```

sequence from each trace is considered. The sequences from the same trace are equivalent also with respect to the reward that they generate. To consider only one sequence per trace, we evaluate only the action sequences in LNF.

Proposition 3.4. *If a word $\boldsymbol{w}$ is not in LNF, no word $\boldsymbol{wu}$ starting with the prefix $\boldsymbol{w}$ is in LNF.*

Proof. If $\boldsymbol{w}$ is not in LNF, then a word $\boldsymbol{v} \simeq \boldsymbol{w}$ must exist so that $\boldsymbol{v} < \boldsymbol{w}$. For all $\boldsymbol{u} \in A^*$ there is $\boldsymbol{vu} \simeq \boldsymbol{wu}$ and $\boldsymbol{vu} < \boldsymbol{wu}$. □

From Proposition 3.4, we know that a sequence $\boldsymbol{w}a$ can only be in LNF if $\boldsymbol{w}$ is in LNF. During a depth-first traversal of the search tree, entire subtrees can thus be pruned away. If the action sequence $\boldsymbol{w}$ leading to the current position in the search tree is known to be in LNF, the candidate sequence $\boldsymbol{w}a$ can be tested for LNF very fast. The test is performed by Algorithm 1. It has a worst-case runtime that is linear in the sequence length. In practice the algorithm performs much faster as a breaking condition is usually found within the first few elements of the sequence.

Proposition 3.5. *If the action sequence $\boldsymbol{w}$ is in LNF, then for all $a \in A$ Algorithm 1 returns true if and only if the sequence $\boldsymbol{w}a$ is also in LNF.*

Proof. For a sequence $\boldsymbol{v}$, denote with $Z(\boldsymbol{v}, k) = \{l : l < k \wedge \forall m \in \{l, \ldots, k-1\}, v^k \perp v^m\}$ the indices of the contiguous elements before v^k that all commute with v^k. In the following we show that an action-sequence $\boldsymbol{v}$ is in Lexicographical Normal Form (LNF) if and only if

$$\forall k \in \{1, \ldots, |\boldsymbol{v}|\},\ \forall l \in Z(\boldsymbol{v}, k),\ v^l < v^k\,. \tag{3.4}$$

Algorithm 2 For an LNF sequence $\boldsymbol{w}$, find the LNF equivalent to $\boldsymbol{w}a$.

```
1: procedure AppendLNF($\boldsymbol{w}, a$)
2:     for $k = |\boldsymbol{w}|, \dots, 1$ do
3:         if TestLNF($\boldsymbol{w}^{:k}, a$) then
4:             return $\boldsymbol{w}^{:k} a \boldsymbol{w}^{k+1:}$
5:     return $a\boldsymbol{w}$
```

```
1: procedure LNF($\boldsymbol{w}$)
2:     $\boldsymbol{v} \leftarrow \varepsilon$
3:     for $k = 1, \dots, |\boldsymbol{w}|$ do
4:         $\boldsymbol{v} \leftarrow$ AppendLNF($\boldsymbol{v}, w^k$)
5:     return $\boldsymbol{v}$
```

First, we show that (3.4) must be satisfied for every sequence $\boldsymbol{v}$ in LNF. Assume $\boldsymbol{v}$ is in LNF and does not hold condition (3.4). Then there exists a combination of indices $l < k$ where $v^l > v^k$ and $\forall m \in \{l, k-1\},\ v^m \simeq v^k$. The decomposition $\boldsymbol{v} = \boldsymbol{u} v^l \boldsymbol{q} v^k \boldsymbol{z}$ can be rearranged as $\boldsymbol{y} = \boldsymbol{u} v^k v^l \boldsymbol{q}\boldsymbol{z}$ with possibly empty subsequence $\boldsymbol{u}, \boldsymbol{q}$ and $\boldsymbol{z}$. This contradicts $\boldsymbol{v}$ being in LNF since $\boldsymbol{v} \simeq \boldsymbol{y}$ and $\boldsymbol{y} < \boldsymbol{v}$.

Second, we show that condition (3.4) is sufficient for $\boldsymbol{v}$ to be in LNF. Assume $\boldsymbol{v}$ satisfies (3.4) and is not in LNF. Then there must exist an equivalent sequence $\boldsymbol{y} \simeq \boldsymbol{v}$ in LNF. Equivalent sequences are permutations and $|\boldsymbol{y}| = |\boldsymbol{v}|$. So for $\boldsymbol{y}$ to have a smaller order, there must be an index l where $\boldsymbol{y}$ and $\boldsymbol{v}$ first differ $\boldsymbol{y}^{:l-1} = \boldsymbol{v}^{:l-1}$ and $v^l < y^l$. Due to $\boldsymbol{y} \simeq \boldsymbol{v}$, action v^l must appear in $\boldsymbol{v}$ at an index greater than l and $\boldsymbol{v}$ can be decomposed as $\boldsymbol{v} = \boldsymbol{u} v^l \boldsymbol{q} y^l \boldsymbol{z}$, with possibly empty subsequences $\boldsymbol{u}, \boldsymbol{q}$ and $\boldsymbol{z}$, where y^l commutes with v^l and all elements of $\boldsymbol{q}$. This contradicts (3.4).

Since Equation 3.4 only refers to the preceding elements in the action sequence and $\boldsymbol{w}$ is known to be in LNF, it suffices to test (3.4) for the new element. □

Proposition 3.6. *For any action a and LNF action sequence $\boldsymbol{w}$, Algorithm 2 returns an action sequence that is in LNF and equivalent to $\boldsymbol{w}a$.*

Proof. When the condition in line 3 of Algorithm 2 is not true, then a commutes with w^k and $a < w^k$. Therefore, once the condition in line 3 is true, $\boldsymbol{w}^{:k}a$ is in LNF and the LNF-order of the sequence cannot be improved by moving an element from $\boldsymbol{w}^{k+1:}$ before a. Since $\boldsymbol{w}$ is in LNF, no permutation of the sub-sequence $\boldsymbol{w}^{k+1:}$ has reduced LNF-order. Therefore $\boldsymbol{w}^{:k} a \boldsymbol{w}^{k+1:}$ is in LNF. □

A trivial approach to search for good sequences is to cast the search space as a tree structure and to enumerate the solutions at the leaf nodes of the tree.

Algorithm 3 Depth-First Search

```
$v^{\max} \leftarrow -\infty,\ \boldsymbol{w}^{\max} \leftarrow \epsilon$
procedure DFS($\boldsymbol{\omega}, v, \boldsymbol{w}$)
    if $|\boldsymbol{w}| = k^{\max}$ then
        if $v > v^{\max}$ then
            $v^{\max} \leftarrow v$
            $\boldsymbol{w}^{\max} \leftarrow \boldsymbol{w}$
        return
    for $a \in A : \boldsymbol{\omega} \in \Sigma_a$ do
        $\boldsymbol{\omega}' \leftarrow a(\boldsymbol{\omega})$
        $v' \leftarrow v + r(\boldsymbol{\omega}, a, \boldsymbol{\omega}')$
        DFS($\boldsymbol{\omega}', v', \boldsymbol{w}a$)
    end for
DFS($\boldsymbol{\sigma}, 0, \epsilon$)
return $\boldsymbol{w}^{\max}$
```

Algorithm 4 Pruned Depth-First Search

```
$v^{\max} \leftarrow -\infty,\ \boldsymbol{w}^{\max} \leftarrow \epsilon$
procedure DFS$^{\text{LNF}}$($\boldsymbol{\omega}, v, \boldsymbol{w}$)
    if $|\boldsymbol{w}| = k^{\max}$ then
        if $v > v^{\max}$ then
            $v^{\max} \leftarrow v$
            $\boldsymbol{w}^{\max} \leftarrow \boldsymbol{w}$
        return
    for $a \in A : \boldsymbol{\omega} \in \Sigma_a \wedge$ TestLNF($\boldsymbol{w}, a$) do
        $\boldsymbol{\omega}' \leftarrow a(\boldsymbol{\omega})$
        $v' \leftarrow v + r(\boldsymbol{\omega}, a, \boldsymbol{\omega}')$
        DFS$^{\text{LNF}}$($\boldsymbol{\omega}', v', \boldsymbol{w}a$)
    end for
DFS$^{\text{LNF}}$($\boldsymbol{\sigma}, 0, \epsilon$)
return $\boldsymbol{w}^{\max}$
```

Algorithm 5 Branch & Bound

```
$v^{\max} \leftarrow -\infty,\ \boldsymbol{w}^{\max} \leftarrow \epsilon$
procedure B&B($\boldsymbol{\omega}, v, \boldsymbol{w}$)
    if $|\boldsymbol{w}| = k^{\max}$ then
        if $v > v^{\max}$ then
            $v^{\max} \leftarrow v$
            $\boldsymbol{w}^{\max} \leftarrow \boldsymbol{w}$
        return
    for $a \in A : \boldsymbol{\omega} \in \Sigma_a$ do
        $\boldsymbol{\omega}' \leftarrow a(\boldsymbol{\omega})$
        $v' \leftarrow v + r(\boldsymbol{\omega}, a, \boldsymbol{\omega}')$
        if $v' + \beta(\boldsymbol{\omega}', \boldsymbol{w}a) > v^{\max}$ then
            B&B($\boldsymbol{\omega}', v', \boldsymbol{w}a$)
    end for
B&B($\boldsymbol{\sigma}, 0, \epsilon$)
return $\boldsymbol{w}^{\max}$
```

Algorithm 6 Pruned Branch & Bound

```
$v^{\max} \leftarrow -\infty,\ \boldsymbol{w}^{\max} \leftarrow \epsilon$
procedure B&B$^{\text{LNF}}$($\boldsymbol{\omega}, v, \boldsymbol{w}$)
    if $|\boldsymbol{w}| = k^{\max}$ then
        if $v > v^{\max}$ then
            $v^{\max} \leftarrow v$
            $\boldsymbol{w}^{\max} \leftarrow \boldsymbol{w}$
        return
    for $a \in A : \boldsymbol{\omega} \in \Sigma_a \wedge$ TestLNF($\boldsymbol{w}, a$) do
        $\boldsymbol{\omega}' \leftarrow a(\boldsymbol{\omega})$
        $v' \leftarrow v + r(\boldsymbol{\omega}, a, \boldsymbol{\omega}')$
        if $v' + \beta(\boldsymbol{\omega}', \boldsymbol{w}a) > v^{\max}$ then
            B&B$^{\text{LNF}}$($\boldsymbol{\omega}', v', \boldsymbol{w}a$)
    end for
B&B$^{\text{LNF}}$($\boldsymbol{\sigma}, 0, \epsilon$)
return $\boldsymbol{w}^{\max}$
```

Every node in the tree represents a (partial) action sequence. Depth-first search completely enumerates the solutions at the leaf nodes without having to hold the entire tree in memory. Algorithm 3, called DFS for depth-first search, walks over the tree in recursive fashion. The algorithm backtracks to a prior partial action sequence once all solutions starting at the current position in the tree have been enumerated. The tree of depth $k^{\max}$ contains up to $\sum_{k=0}^{k^{\max}} |A|^k$ nodes and $|A|^{k^{\max}}$ leaf nodes. Enumerating all possible combinations is intractable for all but the most trivial scenarios. Usually the branching factor of the tree is reduced because not all actions from A are applicable in the intermediary system states. But the problem of combinatorial explosion of the search space remains. Proposition 3.4 can be used to test at every node whether the entire subtree starting at the node is not in LNF and can be skipped (pruned). Since every trace contains exactly one sequence in LNF, it is ensured that traversing the tree without the pruned branches still visits every trace once. Algorithm 4, called DFS$^{\text{LNF}}$, extends DFS with the pruning of sequences that are not in LNF. Note that only line 10 of Algorithm 4 has changed compared to DFS.

Another technique to speed up tree search is Branch & Bound [Lit+63]. Branch & Bound is a well-known technique for combinatorial optimization in the presence of so-called admissible heuristics [Pea84]. A heuristic is called admissible if it overestimates the performance that can still result from a partial solution. Here, we express the admissible heuristics as a function $\beta(\boldsymbol{\sigma}, \boldsymbol{w})$ with the current system state $\boldsymbol{\sigma}$ and the current partial action sequence $\boldsymbol{w}$ as arguments. If the admissible heuristics show that the best performance starting from the current partial solution is worse than the best solution that was already encountered, then the entire subtree behind the current partial solution can be pruned. Branch & Bound is implemented in Algorithm 5. In Algorithm 6, Branch & Bound is combined with trace-based pruning. Again, only line 10 is changed to prune out sequences that are not in LNF. The commonality between DFS and Branch & Bound is the use of backtracking to return to a previous partial solution either when all nodes in a subtree have been visited or when the solutions that remain in the subtree are known to be suboptimal.

3.1.3 Evaluation

The effect of the pruning techniques for tree search methods is evaluated based on the Jobshop Scheduling Problem (JSP) [Pin08]. Scheduling is one of the most important planning problems on the shopfloor. It is also computationally challenging. The jobs $j \in J$ have to be processed on the machines M. Every job consists of operations $o \in O_j = \{1, 2, \dots, |M|\}$. The operations need to be processed in-order for every job and are each assigned to a specific machine $m_{j,o} \in M$. The operation duration is $d_{j,o}$. The goal is to find a schedule that assigns operations to machines in order to minimize the finishing time of the last job. See Table 3.1c for a benchmark JSP from the literature. There are $(|J|!)^{|M|}$ possible schedules for a JSP with $|J|$ jobs and $|M|$ machines where every jobs needs to visit every machine once [JM98]. So a JSP with $|J| = 20$ and $|M| = 10$ has 7.2651×10^{183} possible solutions. Compare this to the mass of the observable universe currently estimated to the equivalent of 10^{80} hydrogen atoms. Even though the number of possible solutions is huge, current solution techniques can compute near-optimal solutions for JSP with hundreds of machines and jobs. Many benchmark scheduling problems have even been solved with certified optimaltiy. However, the worst-case analysis of the JSP shows that it is NP-hard for instances where $|J| \geq 3$ and $|M| \geq 3$ [GJS76]. Therefore, unless P $=$ NP, all algorithms for solving the JSP are either heuristic or require a long running time for some (synthetic) JSP problems.

We cast the JSP in the model from Chapter 2 in terms of actions with preconditions and effects. Every machine and every job in a JSP is represented as a component in the system model $C = C_{\texttt{machines}} \cup C_{\texttt{jobs}}$. The operations for each job are represented as actions where both one machine and one job participate. The time-indexed state of the machine components is trivial. It only consists of the time when the machine is available next. The state of the job components is the number of operations that were already performed for the job. The precondition of every action is that the associated operation is next in line for the job. The action effect simply increases the number of finished operations for the job by one. The duration of each action (operation in the JSP) is deterministic. The cost generated by an action is the increase in the maximum timestamp of the components. So the cost of the entire sequence is

Job	Operations (m, p)
1	(0, 5), (1, 10), (2, 5)
2	(1, 10), (0, 5), (2, 5)
3	(2, 5), (1, 10), (0, 5)

(a) Minimal 3×3 JSP.

Job	Operations (m, p)
1	(0, 5), (1, 10), (2, 5), (3, 10)
2	(1, 10), (3, 5), (2, 5), (0, 5)
3	(1, 5), (0, 10), (2, 5), (3, 5)
4	(3, 5), (2, 10), (0, 5), (1, 10)

(b) Minimal 4×4 JSP.

Job	Operations (m, p)
1	(2,1), (0,3), (1,6), (3,7), (5,3), (4,6)
2	(1,8), (2,5), (4,10), (5,10), (0,10), (3,4)
3	(2,5), (3,4), (5,8), (0,9), (1,1), (4,7)
4	(1,5), (0,5), (2,5), (3,3), (4,8), (5,9)
5	(2,9), (1,3), (4,5), (5,4), (0,3), (3,1)
6	(1,3), (3,3), (5,9), (0,1), (4,4), (2,1)

(c) The 6×6 benchmark JSP `ft06` from [FT63].

Table 3.1: Example JSP problems.

the makespan of the solution.

$$\mathfrak{r}(\boldsymbol{\sigma}, a, \boldsymbol{\sigma}') = \big(\max_{c \in C} t_c\big) - \big(\max_{c \in C} t'_c\big) \tag{3.5}$$

Here, t_c refers to the simulation time of component c in the system state $\boldsymbol{\sigma}$ and t'_c refers to the simulation time of the component in the system state $\boldsymbol{\sigma}'$.

Given a partial schedule as an action sequence $\boldsymbol{w}$, the following trivial heuristic computes an upper bound for the reward that can be generated following the execution of $\boldsymbol{w}$. For every component (representing a job or a machine), the remaining operations for the component are summed up and added to its current time.

$$\beta^{JSP}(\boldsymbol{\sigma}, \boldsymbol{w}) = -\max_{c \in C} \left(t_c + \sum_{\substack{a \in A : a \notin \boldsymbol{w}, \\ c \in C_a}} d_a \right) \tag{3.6}$$

The `ft06` benchmark problem from Table 3.1c illustrates the importance of optimisation. An unoptimized solution can require more than 150 seconds to complete all jobs. The optimal solution requires just 55s. Finding the optimal solution for larger problems is not possible without the help of computers. In addition to the `ft06` benchmark, two additional minimal problems are considered. See Table 3.1 for their exact definition. The minimal examples are too small to be of any practical value. But they give an indication of how fast the problem complexity increases. A sequence of 9 unique elements has $9! = 362,880$ permutations. The minimal 3×3 JSP depicted in Table 3.1 also defines 9 actions. But the full scenario tree for the JSP has only 1,680 leaf nodes corresponding to valid schedules. By additionally pruning branches that are not in the LNF only 63 leaf nodes remain. Using the makespan lower bound from Equation 3.6, nodes can be pruned where the lower bound is equal or worse to the best solution encountered so far. With the Branch & Bound pruning, only 3 leaf nodes are actually visited. However, more nodes were visited overall compared to trace-based pruning for full depth-first search. This indicates that mostly branches close to the leaves were pruned with Branch & Bound only. By combining trace pruning with Branch & Bound, only 65 nodes are visited overall. This is a decrease in the search running time by a factor of more than 50 compared to Branch & Bound search without pruning.

On the 4×4 JSP depicted in Table 3.1 an even smaller fraction of the scenario-tree is visited by combining trace-based and makespan-based pruning. However, the number of visited nodes still grows fast with the size of the JSP. On the `ft06` 6×6 JSP, 682,508 nodes are expanded with both bound and trace pruning enabled. Our implementation is capable of visiting over 1,000,000 nodes per second during Branch & Bound search (on a Lenovo T480 laptop computer). However, with only either trace-based pruning or Branch & Bound enabled, we could not solve `ft06` in over a day of computation. A 10×10 JSP could not be solved within several days of compute time even with both trace-based and bound-based are activated. In summary, trace-based pruning

	DFS	B&B	DFS$^{\text{LNF}}$	B&B$^{\text{LNF}}$
Visited Nodes	3,568	405	348	65
Visited Leaf Nodes	1,680	3	63	3
Trace-Pruned Branches	0	0	191	41
Bound-Pruned Branches	0	376	0	35

(a) Visited nodes for the minimal 3×3 JSP

	DFS	B&B	DFS$^{\text{LNF}}$	B&B$^{\text{LNF}}$
Visited Nodes	128,385,941	115,042	105,666	826
Visited Leaf Nodes	63,063,000	9	11,143	9
Trace-Pruned Branches	0	0	93,074	1,131
Bound-Pruned Branches	0	206,025	0	643

(b) Visited nodes for the minimal 4×4 JSP

	DFS	B&B	DFS$^{\text{LNF}}$	B&B$^{\text{LNF}}$
Visited Nodes	?	?	?	682,508
Visited Leaf Nodes	?	?	?	43
Trace-Pruned Branches	0	0	?	2,051,681
Bound-Pruned Branches	0	?	0	750,063

(c) Visited nodes for the `ft06` 6×6 JSP

Table 3.2: Benchmarking of pruning techniques for DFS and Branch & Bound. Questionmarks indicate that an optimization did not terminate after two weeks of computation. So the numbers are outstanding.

reduces the number of visited action sequences to a small fraction. For the 4×4 JSP, the number of action sequences (visited leaf nodes) was reduced by a factor of more than 5,600. The improvements are comparatively increasing with the length of the action sequences.

We have shown that LNF pruning leads to a dramatic reduction of the search space. The reduction is already on many orders of magnitude for the comparably small benchmark problems that were considered. However, even with trace-based pruning, naive tree search does not scale up to scenarios of relevant size. In practice, genetic algorithms are often used to solve combinatorial problems. Modern heuristic solvers find good solutions for JSP with thousands of jobs [Dim15]. But these solvers exploit the specific structure of the JSP. Handling of complex action preconditions is near-impossible for genetic algorithms. The genetic crossing and mutation of two partial solutions will almost always lead to an infeasible action sequence where the preconditions of an action are not met. Repairing an infeasible plan is usually quite difficult. For example if a single product is missing to finish an order of many hundred products.

3.2 Planning for Discrete Action Sequences

The algorithms from Section 3.1 use backtracking to return to previous partial solutions. This requires either that the system state is fully known so that it can be stored in a computer (to return to previous states) or that action sequences can be deterministically repeated. For experiments carried out in the physical world, neither of these assumptions is true. In this section, we further speed up planning by “implicitly pruning” less promising branches with Monte-Carlo Tree Search (MCTS). Many recent breakthroughs in Artificial Intelligence were made possible by MCTS. This includes the AlphaGo system [Sil+17] which is able to play the game of Go with superhuman performance. In short, MCTS enables planning in scenarios with many combinatorial variations where backtracking tree search is not able to cover any significant portion of the search space. In addition, MCTS uses only forward-simulation without backtracking. This reduces the coupling between the planning algorithm and the simulator to the point where real-world simulations could be used to generate samples for the planning algorithm.

3.2.1 Background: Monte-Carlo Tree Search

In MCTS [Bro+12; Mun+14], a scenario tree is explored by iterative playouts. A playout is essentially one run of the scenario with sequential decision making in a series of steps. Every playout starts at the root of the sequence tree and evolves by "forward simulation". In contrast to Branch & Bound, there is no backtracking to previous states within a playout. Historically, MCTS evolved from research on the multi-armed bandit problem: The multi-armed bandit problem is an idealized version of a slot machine. The following short exposition follows [Mun+14] and adapts the notation to the conventions of this text. Consider a multi-armed bandit in a casino where they player can choose a different arm during $k^{\max}$ rounds (this will be extended to sequential decision making later on). The reward of the different arms $a \in A$ is random according to the distribution $\mathbb{P}_a$ with a support on $[0, 1]$. But the player is initially unaware of the reward distribution for every arm. In each round k, the player chooses a bandit $a^k \in A$ and collects a reward $r^k \sim \mathbb{P}_{a^k}$. The rewards generated at the other arm are not observed. The goal is to find a strategy that maximizes the expected sum of payouts during the k rounds. This leads to the so-called *Exploration/Exploitation Tradeoff*: The player could select a bandit with high expected reward. But this might get him stuck at a bandit with suboptimal expected reward. So the player wants to explore other options without loosing too much in the process. The expected reward of the different arms $a \in A$ is $\mu_a = \mathbb{E}[\mathbb{P}_a]$. The expected reward of the best arm (there can be several best arms) is $\mu^* = \max_{a \in A} \mu_a$. Possible strategies for repeated play are analyzed with respect to their *expected regret*. The cumulative regret after k rounds compares the actual reward r with the expected reward of choosing the arm with highest expected reward every time.

$$b_k = k\mu^* - \sum_{l=1}^{k} r^l$$

The expected cumulative regret is therefore

$$\mathbb{E}[b_k] = k\mu^* - \sum_{l=1}^{k} \mathbb{E}[r^l] = \sum_{a \in A} \mathbb{E}[n_a(k)](\mu^* - \mu_a)\,,$$

where $n_a(k)$ denotes the number of draws from a after the first k rounds.

An important tool for bounding the expected cumulative regret is the Chernoff-Hoeffding inequality. Let $(\underline{y}_i)_{i=1,\dots,n}$ i.i.d. samples of a probability distribution with support $[0,1]$ and mean μ. The empirical mean estimator $\underline{\hat{\mu}} = \frac{1}{n}\sum_{i=1}^{n}\underline{y}_i$ is a function of the $\underline{y}_i$ and therefore a random variable as well. The Chernoff-Hoeffding inequality gives probability bounds for the distance between the true and the estimated mean.

$$\mathbb{P}\left(\underline{\hat{\mu}} - \mu \geq \epsilon\right) \leq e^{-2n\epsilon^2} \quad \text{and} \quad \mathbb{P}\left(\underline{\hat{\mu}} - \mu \leq -\epsilon\right) \leq e^{-2n\epsilon^2}$$

The bound is independent of the underlying distribution of the y_i. So it can be applied even when the distribution of the $\underline{y}_i$ is not known!

There exists a variety of approaches for selecting the next draw in the multi-armed bandit setting. Auer et al. [ACF02] proposed the so-called Upper Confidence Bound (UCB) algorithm. Let $\hat{\mu}_{a,k}$ denote the mean return of arm a sampled during the first k rounds. UCB always selects the next arm as follows:

$$a^k = \arg\max_{b\in A} \hat{\mu}_{b,k-1} + \sqrt{\frac{3\log k}{2n_b(k-1)}}$$

Selecting the best arm according to the UCB gives an upper bound for the expected cumulative regret by application of the Chernoff-Hoeffding inequality.

$$\mathbb{E}[b^k] \leq 6 \sum_{\substack{a\in A,\\ \mu^*>\mu_a}} \frac{\log k}{\mu^* - \mu_a} + |A|(\frac{\pi^2}{3} + 1)\,,$$

The expected cumulative regret grows at most logarithmically in k.

The algorithm of Auer et al. [ACF02] selects actions from a flat list of options. Monte-Carlo Tree Search (MCTS) is a family of algorithms that uses the same principle of iterative exploration for planning in sequential decision settings. A popular variation of MCTS, called Upper Confidence on Trees (UCT) [KS06], uses the UCB decision rule for every action choice during a playout. See Figure 3.3 for an overview. Algorithm 7 shows UCT (but with some modifications compared to [KS06] that will be explained in the following). MCTS can be seen as “simulation-based” planning, as no backtracking is used.

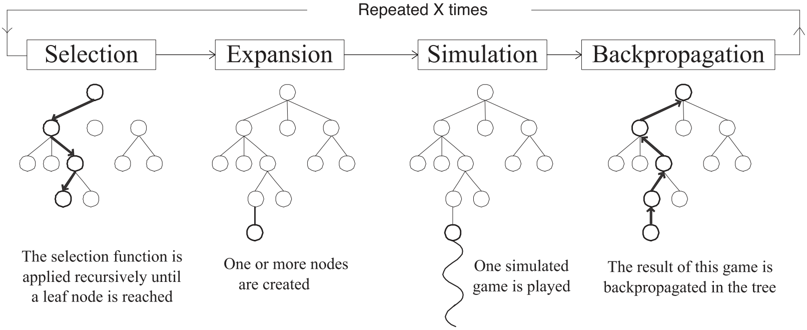

Repeated X times
Selection
Expansion
Simulation
Backpropagation
The selection function is applied recursively until a leaf node is reached
One or more nodes are created
One simulated game is played
The result of this game is backpropagated in the tree

Figure 3.3: Outline of a Monte-Carlo Tree Search. Reproduced from [Cha+08].

Instead, the results from the last simulation are incorporated into statistics about the expected reward for the possible next actions at the current position in the scenario tree. The updated empirical reward statistics is used for decision-making in the following playouts.

The assumption that the reward in every step is in the range $[0, 1]$ is not required for UCT to converge to optimal decisions in the limit. On the downside, there is currently no good characterization of the convergence speed achieved by UCT that doesn't make strong assumptions on the underlying scenario. It is only known that for a large enough number of plays, every branch of the scenario tree is visited. Since only a small subset of the tree is explored in practice before the algorithm is terminated, MCTS struggles with scenarios where the reward is "unevenly distributed". That is, if a big reward can be found after a long sequence of actions where small changes to the action sequence lead to much worse results, it is then not very likely that the reward is encountered at all and MCTS will not assign a correct value estimation for the choices. A workaround for this is Reward Shaping [NHR99] where the reward is distributed such that it may be encountered early on in the action sequences.

UCT may require tuning of the parameter α which regulates the importance of the upper confidence bias. Setting α corresponds to making a choice for the Exploration/Exploitation tradeoff between the two extremes "always explore" and "always exploit". Note that UCB was developed for action selection in stochastic scenarios with unknown reward distributions for the different actions.

Algorithm 7 The Upper Confidence on Trees (UCT) algorithm [KS06].

```
procedure UCT($\boldsymbol{\sigma}^0$)
    $q[\cdot] \leftarrow 0$
    $n[\cdot] \leftarrow 0$
    while enough time do
        PLAY($\boldsymbol{\sigma}^0, \epsilon$)
    return $\arg\max_{a \in A} q[a]$
```

```
procedure ROLLOUT($\boldsymbol{\sigma}$)
    if DONE($\boldsymbol{\sigma}$) then
        return 0
    $a \leftarrow \pi_A(\boldsymbol{\sigma})$
    $(\boldsymbol{\sigma}', r) \leftarrow a(\boldsymbol{\sigma})$
    return $r +$ ROLLOUT($\boldsymbol{\sigma}'$)
```

```
procedure PLAY($\boldsymbol{\sigma}, \boldsymbol{w}$)
    if DONE($\boldsymbol{\sigma}$) then
        return 0
    $B \leftarrow \{b \in A : n[\boldsymbol{w}b] = 0\}$
    if $B \neq \varnothing$ then
        $a \leftarrow \pi_B(\boldsymbol{\sigma})$
        $(\boldsymbol{\sigma}', r) \leftarrow a(\boldsymbol{\sigma})$
        $r \leftarrow r +$ ROLLOUT($\boldsymbol{\sigma}'$)
    else
        $a \leftarrow \arg\max_{b \in A, \boldsymbol{\sigma} \in \Sigma_b} \left[q[\boldsymbol{w}b] + \alpha\sqrt{\frac{\log n[\boldsymbol{w}]+1}{n[\boldsymbol{w}b]}}\right]$
        $(\boldsymbol{\sigma}', r) \leftarrow a(\boldsymbol{\sigma})$
        $r \leftarrow r +$ PLAY($\boldsymbol{\sigma}', \boldsymbol{w}a$)
    $n[\boldsymbol{w}a] \leftarrow n[\boldsymbol{w}a] + 1$
    $q[\boldsymbol{w}a] \leftarrow q[\boldsymbol{w}a] + \frac{r - q[\boldsymbol{w}a]}{n[\boldsymbol{w}a]}$
    return $v$
```

Here we apply the same principle to tree-search for deterministic scenarios. There is an informal argument for this. Many rollout policies are stochastic, similar to the uniform sampling policy described above. So the rollout takes samples from the reward distribution that is implied by applying the current rollout policy to the subsequent steps. When nodes are visited several times, the underlying distribution for the policy changes. So past experience does not necessarily match the samples taken later on. Nevertheless, the UCB-based selection rule shows good performance in practice.

3.2.2 Monte-Carlo Tree Search for Discrete Action Sequences

We introduce a modification of UCT called UCT$^{\text{LNF}}$. See Algorithm 8 for details. UCT$^{\text{LNF}}$ is inspired from UCT but deviates in a few key aspects. First, the original UCT algorithm gradually builds up a tree structure where the nodes correspond to partial action sequences. But only one new node is added with every play. Once the edge of the current tree is reached, the remaining steps are “rolled out” and only the sum of rewards for the rollout is considered. UCT$^{\text{LNF}}$ does not use rollouts and adds nodes for all selected actions in the sequence.

Algorithm 8 UCT for Deterministic Actions with Trace-Based Pruning

```
procedure $\mathrm{UCT}^{\mathrm{LNF}}(\boldsymbol{\sigma}^0)$
    $n[\cdot] \leftarrow 0,\ q[\cdot] \leftarrow 0$
    while enough time do
        $Y \leftarrow \textsc{Play}^{\mathrm{LNF}}(\boldsymbol{\sigma}^0)$
        $\textsc{Update}^{\mathrm{LNF}}(Y)$
    return $\arg\max_{a \in A} q[a]$
```

```
procedure $\textsc{Update}^{\mathrm{LNF}}(Y)$
    $z[\cdot] \leftarrow 0$
    for $(\boldsymbol{w}, \boldsymbol{r}) \in Y$ do
        for $k = |\boldsymbol{w}| \dots, 1$ do
            $\boldsymbol{g} \leftarrow \boldsymbol{w}^{:k}$
            if $z[\boldsymbol{g}] > 0$ then
                break
            $n[\boldsymbol{g}] \leftarrow n[\boldsymbol{g}] + 1,\ z[\boldsymbol{g}] \leftarrow 1$
            $q[\boldsymbol{g}] \leftarrow r^k + \max_{a \in A,\ n[\boldsymbol{g}a] > 0} q[\mathrm{LNF}(\boldsymbol{g}a)]$
```

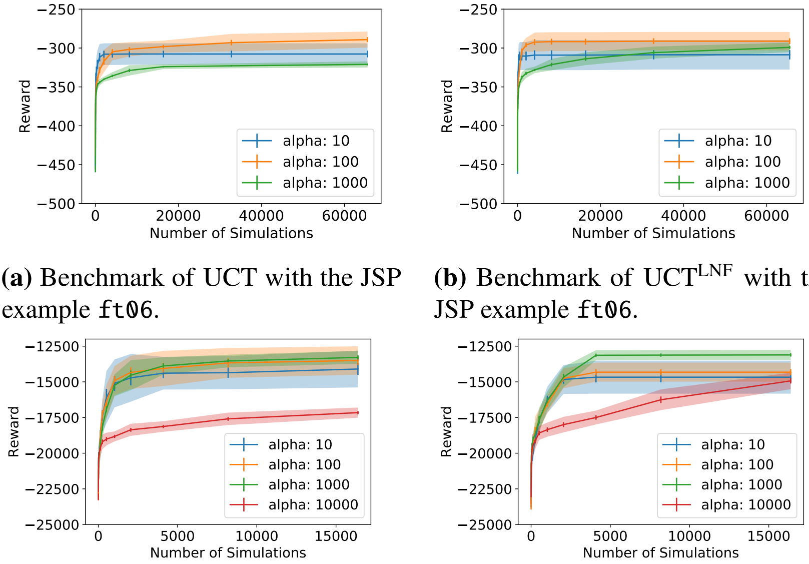
```
procedure $\textsc{Play}^{\mathrm{LNF}}(\boldsymbol{\sigma})$
    $\boldsymbol{w} \leftarrow \varepsilon,\ \boldsymbol{r} \leftarrow \varepsilon,\ Y \leftarrow \varnothing$
    while $\neg\textsc{done}(\boldsymbol{\sigma})$ do
        $B \leftarrow \{b \in A : n[\boldsymbol{w}b] = 0\}$
        if $B \neq \varnothing$ then
            $a \leftarrow \pi_B(\boldsymbol{\sigma})$
        else
            $a \leftarrow \arg\max_{b \in A,\ \boldsymbol{\sigma} \in \Sigma_b} \left[ q[\mathrm{LNF}(\boldsymbol{w}b)] + \alpha \sqrt{\frac{\log n[\boldsymbol{w}] + 1}{n[\mathrm{LNF}(\boldsymbol{w}b)]}} \right]$
        $(\boldsymbol{\sigma}, v) \leftarrow a(\boldsymbol{\sigma})$
        $Y \leftarrow Y \cup \{(\boldsymbol{w}a, \boldsymbol{r}v),\ \mathrm{LNF}(\boldsymbol{w}a, \boldsymbol{r}v)\}$
        $(\boldsymbol{w}, \boldsymbol{r}) \leftarrow \mathrm{LNF}(\boldsymbol{w}a, \boldsymbol{r}v)$
    return $Y$
```

Second, the original UCT stores a statistics about the average reward that was achieved after selecting a node (action). Instead, we perform a maximization in every step of the Update procedure. Therefore, the value estimation of every node is the maximum reward that can be achieved by following the best known sequence of actions after selecting the node (action). Third, we do not only update a single sequence for every play. Instead, when we arrive at a sequence that is not in LNF, then we permute the sequence to the equivalent LNF sequence. But we store the original sequence and the LNF sequence and run the Update procedure on both of them. All three modifications to the original UCT algorithm are interrelated. This is explained next.

The easiest way to restrict the search to sequences in LNF only is to simply remove actions leading to non-LNF sequences at the current node in the search tree. But when choosing one action at a time, this often leads to dead-ends where no action can be chosen without breaking the LNF constraint. Since we do not want to backtrack to previous system states during a playout, virtually no playout would complete. Therefore we use the LNF algorithm to repair the action sequence as we go along. However, simply permuting the sequence to LNF after every action selection is not compatible with the UCT approach either: Soon, some nodes are always selected because they have a very low visit count. But as the sequence is permuted to LNF, this nodes continue to not have their visit count n increased. Suppose an example with three actions $a < b < c$ that all commute and where every action can be chosen only once in every sequence. The sequence starting with the action c would always become the LNF sequences ac or bc after the second action choice. So the node for the first action c is nevery updated. But the UCT algorithm will always choose c first as long as the counter $n[c]$ is not increased. This is solved by updating all sequences that were encountered in the current playout, regardless of whether they are in LNF or not. The counter z for the node updates within the current playout ensures that no node is updated twice for one playout. Note that we use the algorithm LNF in a slightly different fashion. Giving the action sequence $\boldsymbol{w}$ and the reward history $\boldsymbol{r}$ as input, both are permuted to yield an LNF action sequence where the index of actions still matches the index of the corresponding reward.

3.2.3 Evaluation

We evaluate the algorithms UCT and UCT$^{\mathrm{LNF}}$ on the benchmark Jobshop Scheduling Problems (JSP): The `ft06` benchmark with 6 jobs on 6 machines from [FT63] and the `abz5` benchmark with 10 jobs on 10 machines from [ABZ88]. However, we change the classical JSP in one important aspect: Instead of optimizing the makespan, the time when the last job finishes, we target the sum of the finishing times (the tardiness if the job is immediately due) for all jobs. The reason for the change is to ensure that the immediate reward of actions depends on a small number of participating components C_a only. When optimizing the makespan, it is of course possible to return the negative increase of the maximum simulation time for all components. But then the actions would have all components C as participants. We prefer to have only a small subset of the components participating in each action, so that the trace-based equivalence



(a) Benchmark of UCT with the JSP example `ft06`.

(b) Benchmark of UCT$^{\text{LNF}}$ with the JSP example `ft06`.

(c) Benchmark of UCT with the JSP example `abz5`.

(d) Benchmark of UCT$^{\text{LNF}}$ with the JSP example `abz5`.

Figure 3.4: JSP benchmarks for Monte-Carlo Tree Search. Every benchmark was run 10 times. The lines give the average empirical reward. The standard error is indicated by tick marks and the variance is shown in a lighter shade.

of action sequences as defined in Section 3.1.2 can be used to prune the sequence tree. The rollout policy π is set to select randomly among the possible choices with a uniform distribution.

As can be seen in Figure 3.4, MCTS makes big improvements to the best-known solution early on. Depending on the exploration parameter α, the algorithm then "converges" towards a reward value that is no longer improved upon even with very long running times. This is not a true convergence however, since we know that all branches of the search tree are explored eventually with the UCB rule. The more complex `abz5` example shows an interesting phenomenon where higher α lead to better results. But it takes longer until "convergence" is reached. This insight makes sense with regards to the UCB action selection. Once all actions at a given node have been explored several times, the UCB rule will predominantly choose actions with a known high reward. So the observed convergence is explained by a shift from exploration to exploitation in the action selections. Pruning sequences that are not in LNF speeds up the convergence to some final best reward. This was expected as the pruning effectively reduces the size of the search tree.

3.3 Planning with Uncertainty and Continuous Action Parameters

As defined in Section 2.2, parameterized actions $\boldsymbol{a}$ take parameters from the set Θ_a. But the algorithms from the previous sections 3.1 and 3.2 can only handle discrete choices of deterministic actions. If the set of parameters is finite, we can simply extend the tree-search techniques to search over the joint space of actions and their parameters. This is however not possible if the action parameters are uncountable. For example if a parameter is defined on a continuous domain. Furthermore, while the UCB rule for bandit problems is defined for decision making in stochastic unknown environments, UCT assumes deterministic actions. This section deals with the extension of the simulation-based planning approach to actions a that can be both stochastic and have parameters on a continuous parameter space.

3.3.1 Background: MCTS under Uncertainty

For MDP, where the outcome depends not only on decision-making, but also on the response from the stochastic system model, it has been known for some time that a randomly sampled subset of the scenario tree that covers only a vanishing fraction of the full scenario is enough to compute near-optimal actions from any state [KMN02]. If ϵ is the admissible error for the estimation of the V-value of the current state (see Section 2.4), then the number of sample playouts grows exponentially in ϵ. It does however not depend on the size and complexity of the state representation. With the advent of MCTS, sampling based methods have also been used for sequential decision-making under uncertainty [KS06].

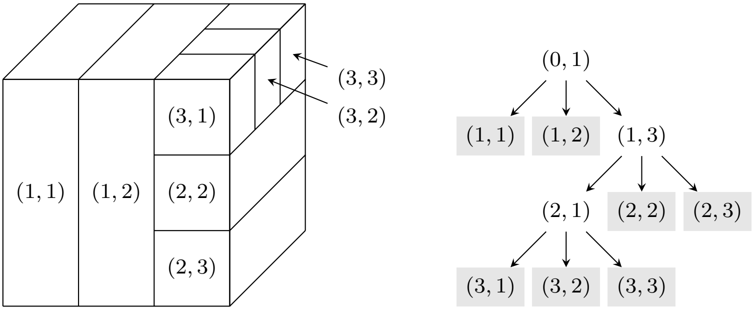
The POMCP algorithm (Partially Observable Monte-Carlo Planning) extends MCTS to scenarios with partial observability (POMDP) [SV10]. The authors [SV10] write with regards to the performance of their invention POMCP:

Algorithm 9 Partially Observable Monte-Carlo Planning

```
procedure POMCP($\underline{\boldsymbol{\sigma}}^0$)
    $q[\,\cdot\,] \leftarrow 0$
    $n[\,\cdot\,] \leftarrow 0$
    while enough time do
        $\boldsymbol{\sigma} \sim \underline{\boldsymbol{\sigma}}^0$
        $\textsc{Play}(\boldsymbol{\sigma}, \varepsilon)$
    return $\arg\max_{a \in A} q[a]$

procedure ROLLOUT($\boldsymbol{\sigma}, \boldsymbol{h}$)
    if $\textsc{done}(\boldsymbol{\sigma})$ then
        return $0$
    $a \leftarrow \pi_A(\boldsymbol{h})$
    $(\boldsymbol{\sigma}', o) \sim \underline{a}(\boldsymbol{\sigma})$
    $v \leftarrow r(\boldsymbol{\sigma}, a, \boldsymbol{\sigma}')$
    return $v +$
        $\textsc{Rollout}(\boldsymbol{\sigma}', \boldsymbol{h}ao)$

procedure PLAY($\boldsymbol{\sigma}, \boldsymbol{h}$)
    if $\textsc{done}(\boldsymbol{\sigma})$ then
        return $0$
    $B \leftarrow \{b \in A : n[\boldsymbol{h}b] = 0\}$
    if $B \neq \varnothing$ then
        $a \leftarrow \pi_B(\boldsymbol{h})$
        $(\boldsymbol{\sigma}', o, r) \sim \underline{a}(\boldsymbol{\sigma})$
        $r \leftarrow r + \textsc{Rollout}(\boldsymbol{\sigma}', \boldsymbol{h}ao)$
    else
        $a \leftarrow \arg\max_{b \in A} \Big[ q[\boldsymbol{h}b] +$
            $\alpha \sqrt{\frac{\log n[\boldsymbol{h}]+1}{n[\boldsymbol{h}b]}} \Big]$
        $(\boldsymbol{\sigma}', o, r) \sim \underline{a}(\boldsymbol{\sigma})$
        $r \leftarrow r + \textsc{Play}(\boldsymbol{\sigma}', \boldsymbol{h}ao)$
    $n[\boldsymbol{h}] \leftarrow n[\boldsymbol{h}] + 1$
    $n[\boldsymbol{h}a] \leftarrow n[\boldsymbol{h}a] + 1$
    $q[\boldsymbol{h}a] \leftarrow q[\boldsymbol{h}a] + \frac{r - q[\boldsymbol{h}a]}{n[\boldsymbol{h}a]}$
    return $r$
```

> [On a benchmark problem], POMCP achieved the same performance with 4 seconds of online computation to the state-of-the-art solver SARSOP with 1000 seconds of offline computation.

While the current system state $\boldsymbol{\sigma}$ is known to the simulator used to sample state transitions, the decision-making only relies on the observations resulting from the actions and the reward statistics that were collected prior. See Algorithm 9 for details. The search tree now not only consists of nodes representing actions, but is a bipartite graph of actions and the resulting observations. The rollout policy π takes the full history of actions and observations as input. A trivial rollout policy is to sample uniformly from the previously unexplored actions. This policy is a popular choice as it does not depend on prior knowledge and assumptions about the scenario and the state representation.



(a) The unit cube split into cells.

(b) Tree hierarchy of cells.

Figure 3.5: Example for Optimistic Optimization on the unit cube. The unit cube is split into cells based on a hierarchical partitioning of the domain. Every cell has an index $(\hbar, i)$ that also represents a node in the partitioning tree: $\hbar$ is the height of the tree at that node (the number of splits) and i is the index within the set of cells of height $\hbar$. The initial node $(0, 1)$ represents the entire domain, here depicted as the unit cube. Leaf nodes are marked gray in the partitioning tree.

3.3.2 Background: Optimistic Optimization

Optimization of functions on a continuous domain has a long history. Very efficient solvers exist today for the optimization of convex functions where gradient information is available [BV04]. Gradient-free global optimization of non-convex functions remains challenging. Classical solution techniques are the Dividing Rectangles (DIRECT) algorithm [JPS93] as well as heuristic genetic algorithms [SP97]. While the latter provides no lower bounds for performance, the convergence speed of DIRECT depends on an upper bound of the function's gradient known as the Lipschitz bound. In recent years, the ideas for optimization in discrete bandit settings and MCTS have been translated to optimization of continuous functions. The resulting techniques are known as Optimistic Optimization (OO). See [Mun+14] for a comprehensive account.

Suppose we want to maximize some function $f : X \to \mathbb{R}$ over the domain X.

$$\boldsymbol{x}^* = \arg\max_{\boldsymbol{x} \in X} f(\boldsymbol{x}) \tag{3.7}$$

If we can make assumptions of convexity, then a range of established methods and commercial tools can be used to solve the optimization problem [BV04]. OO does not require an assumption of convexity. The basic idea is to iteratively dissect X into disjoint cells of decreasing size that cover the entire function domain (Figure 3.5a). The hierarchy of cells forms a tree structure (Figure 3.5b). An "optimistic" upper bound is computed for every cell. This upper bound guides the continuing selection and splitting of the cells. The breakthrough of recent work is to not require a Lipschitz bound for f to compute the upper bound. Instead, the function is assumed to be smooth around the optimizer with respect to a semi-metric.[2] This is a much weaker assumption.

Algorithm 10 Deterministic Optimistic Optimization (DOO) for the optimization of an unknown function $f : X \to \mathbb{R}$. This formulation is for the special case where the search domain is the n-dimensional unit cube $X = [0,1]^n$ and cells are split into three children.

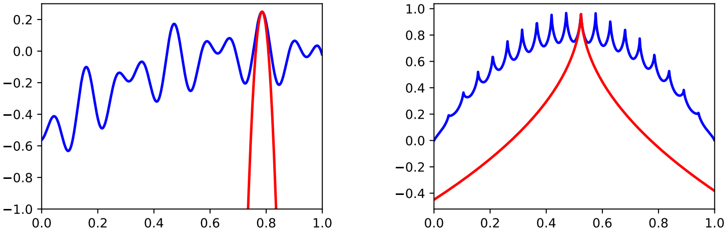
```
procedure DOO($f$)
    $L \leftarrow \{(0,1)\}$                                        ▷ Set of leaf nodes
    $\boldsymbol{x}[0,1] \leftarrow \mathbf{1}\frac{1}{2}$
    while enough time do
        $(\hbar, i) \leftarrow \arg\max_{(\bar{d},j)\in L} [f(\boldsymbol{x}[\bar{d},j]) + \delta(\bar{d})]$
        SPLIT($\hbar, i$)
    return $\arg\max_{(\hbar,i)\in L} f(\boldsymbol{x}[\hbar,i])$

procedure SPLIT($\hbar, i$)
    $v \leftarrow \max\{g : \boldsymbol{x}[\hbar+1,j] \neq \varnothing\}$     ▷ Highest node index at depth $\hbar+1$
    $d \leftarrow \mathrm{mod}(\hbar,n)+1$                   ▷ The dimension to split at depth $\hbar$
    $\boldsymbol{o} \leftarrow \boldsymbol{\nu}_d\left(\frac{1}{3}\right)^{\lfloor \hbar/n \rfloor+1}$          ▷ New distance along the split dimension
    $L \leftarrow L \setminus \{(\hbar,i)\}$                    ▷ Remove the cell from the leaves
    for $j \in \{1,\dots,3\}$ do                           ▷ Add new leaf nodes
        $L \leftarrow L \cup \{(\hbar+1, v+j)\}$
        $\boldsymbol{x}[\hbar+1, v+j] \leftarrow \boldsymbol{x}[\hbar,i] + (j-2)\boldsymbol{o}$
```

Deterministic Optimistic Optimization The first considered OO algorithm is Deterministic Optimistic Optimization (DOO). Algorithm 10 shows a simplified

[2] A semi-metric has $\ell(\boldsymbol{x},\boldsymbol{y}) = \ell(\boldsymbol{y},\boldsymbol{x})$ and $\ell(\boldsymbol{x},\boldsymbol{y}) = 0 \Rightarrow \boldsymbol{x} = \boldsymbol{y}$. Different from a regular norm, the triangle inequality (a consequence of the Cauchy-Schwarz inequality) is not required to hold.

version of DOO. It assumes the domain of f is the n-dimensional unit cube and cells are split into three children. In every iteration, the cell with the highest upper bound is selected and split into 3 children. The cells are denoted as $(\hbar, i)$ where $\hbar$ is the depth of the tree and i is an index for the cells at the same depth.[3] The set L contains the leaf nodes of the current search tree (cf. Figure 3.5b). The midpoint of the visited cells $(\hbar, i)$ is stored as $\boldsymbol{x}[\hbar, i]$. The upper bound of each cell is computed from an evaluation at the midpoint and the bias $\delta(\hbar)$ that depends on the depth-position of the cell. The choice of δ depends on the target function f and semi-metric ℓ. More detail on that can be found in the next paragraph. Along which dimension to split is determined from the tree-depth $\hbar$ at which the cell is situated.

Example 3.3. Figure 3.6 shows the graph of two example functions we seek to maximize on the domain $X = [0, 1]$. Notably, the Garland function is not differentiable at some points on the domain and has no Lipschitz constant. The function is also not differentiable at the optimizer $\pi/6$. But there exists a semi-metric for which the Garland function is locally smooth around the optimizer.



(a) Sine and quadratic: $f_1(x) = 0.25\sin(50x) \cdot \sin(10x) - (x - 0.75)^2$ with a scaled Euclidean metric fitted to the optimizer.

(b) Garland function: $f_2(x) = 4x(1-x)(\frac{3}{4} + \frac{1}{4}(1 - \sqrt{|\sin(60x)|}))$ and the semi-metric $\ell(x, y) = \beta \|x - y\|^{1/2}$ fitted to the optimizer.

Figure 3.6: Example functions that are locally smooth around the optimized for a semi-metric ℓ.

[3] Symbols with a crossing bar as in $\hbar$ are used to denote height-indices in a tree.

To show convergence of DOO, the following assumptions are made for f and its domain [Mun+14].

1. There exists a semi-metric $\ell : X \times X \to \mathbb{R}_+$ for which f is locally smooth around the optimizer: Denote the maximum value of f on its domain with $f^* = f(\boldsymbol{x}^*)$. The function is locally smooth around the optimizer if $f(\boldsymbol{x}^*) - f(\boldsymbol{x}) \leq \ell(\boldsymbol{x}, \boldsymbol{x}^*)$ for all $x \in X$.

2. The domain of each cell is $X_{\hbar,i} \subseteq X$. The midpoint of the cell is $\boldsymbol{x}_{\hbar,i}$. The cell diameter $\delta(\hbar)$ decreases with increasing depth $\hbar$ and $\sup_{\boldsymbol{x} \in X_{\hbar,i}} \ell(\boldsymbol{x}, \boldsymbol{x}_{\hbar,i}) \leq \delta(\hbar)$. So the value of the midpoint and the cell diameter give an upper-bound for the best solution the cell can contain overall.

3. The cells are well-shaped in the sense that there exists a $\mu > 0$ such that for any depth $\hbar \geq 0$ all cells $(\hbar, i)$ of that depth fully contain an ℓ-ball with radius $\mu\delta(\hbar)$ centered in $x_{\hbar,i}$. So all cells have a positive volume.

For every region $(\hbar, i)$ containing the optimizer $\boldsymbol{x}^* \in X_{\hbar,i}$, we have $f(\boldsymbol{x}_{\hbar,i}) + \delta(\hbar) \geq f(\boldsymbol{x}_{\hbar,i}) + \ell(\boldsymbol{x}_{\hbar,i}, \boldsymbol{x}^*) \geq f^*$. Since the leaf nodes always cover the entire function domain, cells with $\boldsymbol{x}_{\hbar,i} + \delta(\hbar) < f^*$ are never expanded as they are dominated by the leaf node containing the optimizer. So the cells potentially expanded at depth $\hbar$ are $I_\hbar = \{(\hbar, i) : f(\boldsymbol{x}_{\hbar,i}) + \delta(\hbar) \geq f^*\}$.

(Stochastic) Simultaneous Optimistic Optimization DOO requires no global Lipschitz bound of the target function. But it requires knowledge of a semi-metric ℓ that is smooth around the optimizer. The ℓ is not known in many cases. The Simultaneous Optimistic Optimization (SOO) algorithm [Mun11] adopts ideas from the DIRECT algorithm [JPS93] to achieve nearly the same convergence results as DOO even without knowledge of ℓ. SOO still assumes the existence of a such a semi-metric ℓ. But it suffices to show the existence of any such semi-metric for the convergence analysis without actually using it in the algorithm. In many cases, the function f itself can be used to construct a suitable semi-metric! Take any norm $\|\cdot\|$ for the function domain X. The semi-metric $\ell(\boldsymbol{x}, \boldsymbol{y})$ for points $\boldsymbol{x}$, $\boldsymbol{y}$ on X is constructed as follows. With $\eta = \|\boldsymbol{x} - \boldsymbol{y}\|$ the distance on the domain norm, the distance on ℓ is the difference between the optimizer and the worst point in the η-ball around the optimizer:

$$\tilde{\ell}(\eta) = \sup_{\|\boldsymbol{x}-\boldsymbol{x}^*\|\leq\eta} (f^* - f(\boldsymbol{x})), \quad \ell(\boldsymbol{x},\boldsymbol{y}) = \tilde{\ell}(\|\boldsymbol{x}-\boldsymbol{y}\|) \tag{3.8}$$

We forego full convergence proofs at this point and refer to the original paper [Mun11]. Stochastic SOO (StoSOO) [VCM13] extends SOO to the optimization of stochastic functions. Cells are split only after κ evaluations. The mean of the sampled values at the cell midpoint is used for the evaluation. The authors of [VCM13] provide a convergence bound for the expected regret on the order of $O(\log^2(\iota)/\sqrt{\iota})$ where ι is the number of samples taken from the stochastic function $\underline{f}$. The tuning parameter η controls how much emphasis the algorithm is putting on exploration, i.e. the tradeoff between exploration and exploitation.

Algorithm 11 shows a simplified version of StoSoo for the n-dimensional unit cube where cells are split into three child cells after κ evaluations. We will now explain the major changes compared to DOO. First, the visited nodes are split into leaf nodes L and internal nodes N. The internal nodes have been sampled κ times and are no longer evaluated. Second, since we do know the semi-metric ℓ (and hence the cell diameter δ), an upper confidence bias is added to the cell evaluation for the selection. Third, the algorithm iterates over the depth-level $\hbar$ of the cells. One cell is selected at every level (if there is an improvement compared to the previous levels) and the cell is either sampled again or split.

3.3.3 Planning for Parameterized Action Sequences

To integrate StoSOO with MCTS, we introduce so-called hybrid trees. Hybrid trees contain nodes for actions and parameters. Hybrid trees are bipartite as a parameter selection must follow an action selection and vice versa. Partially-observable hybrid trees (POHT) use three types of nodes: actions, parameters and the resulting observations. See Figure 3.7 for an example. Note that hybrid trees not only grow at the leaf nodes. The paramter-nodes represent a cell in the parameter space Θ_a of the associated action a. As the cells of a continuous domain can be partitioned indefinitely often, a hybrid tree can grow new branches at the parameter-nodes during planning.

For planning in POHT, we want to combine OO with MCTS. The upper confidence bound is used to select discrete actions and OO is used to select and iteratively refine the choice of action parameters. The StoSOO algorithm samples from the stochastic target function $\underline{f}$ several times within the SplitStoSOO procedure. This prevents its unmodified use for simulation-based planning – without backtracking – in sequential decision-making settings. Instead, we want every call to the OO subproblem to return exactly one parameter combination θ to continue the playout with the following steps in the scenario. The accumulated reward is then used to update the statistics for the involved branches of the scenario tree.

Algorithm 11 Stochastic Simultaneous Optimistic Optimization (StoSOO) on the $[0,1]^n$ cube. Cells are split after having been sampled κ times.

```
procedure StoSOO($\underline{f}, \kappa, \iota, \eta$)
    $L \leftarrow \{(0,1)\}$, $\boldsymbol{x}[0,1] \leftarrow \mathbf{1}\frac{1}{2}$
    $q[0,1] \sim \underline{f}(\boldsymbol{x}[0,1])$, $n[0,1] \leftarrow 1$
    while less than $\iota$ samples taken do
        $q^{\max} \leftarrow -\infty$
        for $\hbar = 0, \dots, \min(\operatorname{depth}(L), \hbar^{\max})$ do
            $i \leftarrow \arg\max_{j:(\hbar,j) \in L} q[\hbar, j] + \sqrt{\frac{\log(\iota^2/\eta)}{2n[\hbar,j]}}$
            $r \leftarrow q[\hbar, i] + \sqrt{\frac{\log(\iota^2/\eta)}{2n[\hbar,i]}}$
            if $r \geq q^{\max}$ then
                $q^{\max} \leftarrow r$
                if $n[\hbar, i] < \kappa$ then
                    SampleStoSOO($\hbar, i$)
                else
                    SplitStoSOO($\hbar, i$)
    return $\arg\max_{(\hbar,i), n[\hbar,i] > 0} q[\hbar, i]$
```

```
procedure SampleStoSOO($\hbar, i$)
    $y \sim \underline{f}(\boldsymbol{x}[\hbar, i])$
    $n[\hbar, i] \leftarrow n[\hbar, i] + 1$
    $q[\hbar, i] \leftarrow q[\hbar, i] + \frac{y - q[\hbar,i]}{n[\hbar,i]}$
```

```
procedure SplitStoSOO($\hbar, i$)
    $v \leftarrow \max\{l : \boldsymbol{x}[\hbar+1, l] \neq \varnothing\}$
    $d \leftarrow \operatorname{mod}(\hbar, n) + 1$
    $\boldsymbol{o} \leftarrow \boldsymbol{\nu}_d \left(\frac{1}{3}\right)^{\lfloor \hbar/n \rfloor + 1}$
    for $j \in \{1, \dots, 3\}$ do
        $L \leftarrow L \cup \{(\hbar+1, v+j)\}$
        $\boldsymbol{x}[\hbar+1, v+j] \leftarrow \boldsymbol{x}[\hbar, i] + (j-2)\boldsymbol{o}$
        SampleStoSOO($\hbar+1, v+j$)
    $L \leftarrow L \setminus \{(\hbar, i)\}$
```

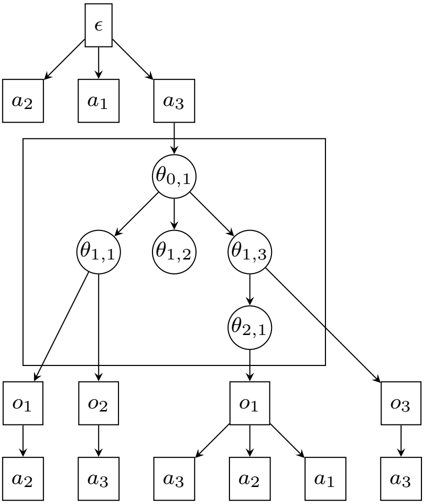


Figure 3.7: Partially Observable Hybrid Tree. The rectangle encompassing several circular nodes denotes a subtree for the optimisation of an action parameter with optimistic optimization. The tree is not fully explored for better visual representation.

Previous authors have used OO for sequential decision-making with a continuous action-space [MWL11; Bus+13; BPM18]. The TrailBlazer algorithm of [GVM16] combines discrete action selection with continuous search. But it uses backtracking to return to a previous position in the scenario tree. In this thesis, we want to avoid storing the system state for backtracking search. So the algorithms can just as well be performed with playouts in physical experiments only.

We now develop the novel Partially Observable Hybrid Tree Planning (POHTP) algorithm that combines MCTS – and in particular the approach for partially-observable planning from to the POMCP algorithm – with Optimistic Optimization for parameter selection on continuous domains. See Algorithm 12 for the full details. In every step for sequential decision making, the algorithm is presented with the choice of several discrete actions that are each parameterized from a continuous domain. If there is only one action, POHTP reduces to StoSOO as a special case. On the other hand, if the actions have no parameters, POHTP reduces to a variation of POMCP. The difference to the original POMCP is the update mechanism that maximizes over possible choices to compute the V-value estimate (instead of taking the empirical reward from the previous plays), full playouts and a switch between exploitation and exploration within each playout that is explained in the next paragraph.

In contrast to POMCP, no rollouts are used that aggregate the reward beyond the previously constructed tree. Instead, the full history of every playout is recorded and used to update the value estimates for the nodes in a second UPDATE procedure. Furthermore, the upper confidence bound is not used for decision-making at every step. Instead, the algorithm initially takes optimal decisions (for the current value estimates) and switches to an explorative regime at the depth d of the decision tree. In the StoSOO algorithm, the dimension along which to split the current cell is determined by the depth in the search tree. The important ingredient of StoSOO to achieve fast convergence is to select cells from a specific depth in each iteration. Similar to StoSOO, POHTP for every iteration selects a depth at which the "exploration" (splitting in StoSOO) begins. The depth d for this switch of the action-selection regime is iterated together with number of performed playouts. Every action contributes one level to the depth of the decision tree. The parameters of the action a contribute according to the depth in the embedded tree for the parameter selection $L[\boldsymbol{h}a]$ for the action a after an observed history $\boldsymbol{h}$. The depth of the parameter-selection tree $\mathrm{depth}(L[\boldsymbol{h}a])$ is the number of times the smallest cell represented in the tree has been split.

The POHTP procedure initializes the algorithm and then iterates over a series of playouts. Importantly, the exploration depth for decision-making d is cut off at the maximum decision-making depth at $\log_2$ of the number of playout iterations.

The PLAY procedure simulates steps until the current scenario is "done". If a node in the search has not been encountered before, the policy π is used to select action and action-parameters. Otherwise, the action is selected via a UCB evaluation and the parameter is selected via OO.

The UPDATE procedure stores the empirical mean reward that is directly generated by an action-parameter combination as $e[\boldsymbol{h}a\theta]$. The Q-value associated with the action-parameter combination additionaly takes the expected following reward into account. This is again the empirical mean over the observations

Algorithm 12 Partially Observable Hybrid Tree Planning (POHTP)

```
procedure POHTP($\boldsymbol{\sigma}^0$)
    $n[\cdot] \leftarrow 0,\ q[\cdot] \leftarrow 0,\ e[\cdot] \leftarrow 0,\ L[\cdot] \leftarrow \{(0,1)\},\ \boldsymbol{x}[\cdot\,;0,1] \leftarrow \mathbf{1}\frac{1}{2},\ đ \leftarrow 1$
    while enough time do
        $(\boldsymbol{h}, \boldsymbol{r}) \leftarrow$ PLAY($\boldsymbol{\sigma}^0$)
        UPDATE($\boldsymbol{h}, \boldsymbol{r}$)
        $đ \leftarrow đ + 1$
        if $đ > \log_2(n[\epsilon])$ then
            $đ \leftarrow 1$
    return $\arg\max_{(a,\theta):n[a\theta]>0} q[a\theta]$
```

```
procedure PARAM($\boldsymbol{h}a, đ$)
    if $đ > \text{depth}(L[\boldsymbol{h}a])$ then
        return $\arg\max_{\theta:\exists(ŧ,i)\in L[\boldsymbol{h}a],\ \boldsymbol{x}[\boldsymbol{h}a;ŧ,i]=\theta} q[\boldsymbol{h}a\theta]$
    else if $đ < 1$ then
        $G \leftarrow L[\boldsymbol{h}a]$
    else
        $G \leftarrow L[\boldsymbol{h}a; d]$
    $(\hbar, i) \leftarrow \arg\max_{(ŧ,j)\in G,\ \boldsymbol{x}[\boldsymbol{h}a;ŧ,j]=\theta} \left[q[\boldsymbol{h}a\theta] + \alpha\sqrt{\frac{\log n[\boldsymbol{h}a]+1}{n[\boldsymbol{h}a\theta]}}\right]$
    $\theta \leftarrow \boldsymbol{x}[\boldsymbol{h}a; \hbar, i]$
    if $n[\boldsymbol{h}a\theta] < \kappa \wedge \hbar = \hbar^{\max}$ then
        return $\theta$
    $u[\boldsymbol{h}a; \hbar, i] \leftarrow u[\boldsymbol{h}a; \hbar, i] + 1$
    $\mu \leftarrow u[\boldsymbol{h}a; \hbar, i]$
    $n \leftarrow \dim(\Theta_a)$
    $\delta \leftarrow \text{mod}(\hbar, n) + 1$
    $\xi \leftarrow |L[\boldsymbol{h}a; \hbar + 1]| + 1$
    $\boldsymbol{x}[\boldsymbol{h}a; \hbar + 1, \xi] \leftarrow \theta + (\mu - 2)\boldsymbol{\nu}_\delta\left(\frac{1}{3}\right)^{\lfloor \hbar/n \rfloor + 1}$
    $L[\boldsymbol{h}a] \leftarrow L[\boldsymbol{h}a] \cup \{(\hbar + 1, \xi)\}$
    if $\mu = 3$ then
        $L[\boldsymbol{h}a] \leftarrow L[\boldsymbol{h}a] \setminus \{(\hbar, i)\}$
    return $\boldsymbol{x}[\boldsymbol{h}a; \hbar + 1, \xi]$
```

```
procedure UPDATE($\boldsymbol{h}$)
    for $k = |\boldsymbol{h}|, \ldots, 1$ do
        $\boldsymbol{g} = \boldsymbol{h}^{:k-1}$
        $(a, \theta, \boldsymbol{o}, r) \leftarrow \boldsymbol{h}^k$
        $n[\boldsymbol{g}a\theta] \leftarrow n[\boldsymbol{g}a\theta] + 1$
        $e[\boldsymbol{g}a\theta] \leftarrow \frac{r - e[\boldsymbol{g}a\theta]}{n[\boldsymbol{g}a\theta]}$
        $q[\boldsymbol{g}a\theta] \leftarrow e[\boldsymbol{g}a\theta] + \sum_{\boldsymbol{o}\in O} \frac{q[\boldsymbol{g}a\theta\boldsymbol{o}]n[\boldsymbol{g}a\theta\boldsymbol{o}]}{n[\boldsymbol{g}a\theta]}$
        $n[\boldsymbol{g}a] \leftarrow n[\boldsymbol{g}a] + 1$
        $q[\boldsymbol{g}a] \leftarrow \max_{\phi\in\Theta_a,\ n[\boldsymbol{g}a\phi]>0} q[\boldsymbol{g}a\phi]$
        $n[\boldsymbol{g}] \leftarrow n[\boldsymbol{g}a] + 1$
        $q[\boldsymbol{g}] \leftarrow \max_{b\in A,\ n[\boldsymbol{g}b]>0} q[\boldsymbol{g}b]$
```

```
procedure PLAY($\boldsymbol{\sigma}, đ$)
    $\boldsymbol{h} \leftarrow \varepsilon$
    while $\neg$DONE($\boldsymbol{\sigma}$) do
        $B \leftarrow \{b \in A : n[\boldsymbol{h}b] = 0\}$
        if $đ > 1 \wedge n[\boldsymbol{h}] > 0$ then
            $a \leftarrow \arg\max_{b\in A, n[\boldsymbol{h}b]>0} q[\boldsymbol{h}b]$
            $đ \leftarrow đ - 1$
            $\theta \leftarrow$ PARAM($\boldsymbol{h}a, đ$)
        else if $B \neq \varnothing$ then
            $(a, \theta) \leftarrow \pi_B(\boldsymbol{\sigma})$
            $đ \leftarrow đ - 1$
        else
            $a \leftarrow \arg\max_{b\in A} \left[q[\boldsymbol{h}b] + \alpha\sqrt{\frac{\log n[\boldsymbol{h}]+1}{n[\boldsymbol{h}b]}}\right]$
            $đ \leftarrow đ - 1$
            $\theta \leftarrow$ PARAM($\boldsymbol{h}a, đ$)
        $đ \leftarrow đ - \text{depth}(L[\boldsymbol{h}a])$
        $(\boldsymbol{\sigma}', \boldsymbol{o}, r) \sim \underline{a}^\theta(\boldsymbol{\sigma})$
        $\boldsymbol{h} \leftarrow \boldsymbol{h}a\theta\boldsymbol{o}r,\ \boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma}'$
    return $\boldsymbol{h}$
```

following the action-parameter combinations. The Q-value for the action and the observation maximize over the respective choices.

So the periods in the observed history are $h^k = (a^k \theta^k o^k)$. Updating the parameterization-nodes is very similar to the updating of action-nodes in POMCP. The considered statistic simply keeps count of how often the node was visited and the empirical reward generated in the ensuing subtree. Since action parameters are now selected as well, the rollout policy $\pi(s, A)$ for choosing an action also returns a matching parameterization. Choosing a good rollout policy is crucial since the optimization of parameterized actions leads to a large number of branches and the search encounters a previously unknown part of the scenario tree in most iterations.

The PARAM procedure returns exactly one parameter combination for the current selected action a. Again, if d is higher than the depth of $L[\boldsymbol{h}a]$, then the best leaf node is returned. Other PARAM is in the explorative regime. If d is smaller than one, then the upper confidence bound is used to select the best parameter among all parameters in the tree. If instead d points to a depth-level inside $L[\boldsymbol{h}a]$, then a node at this depth is selected. Note that a level can become empty when all nodes in the level have been split. If $L[\boldsymbol{h}a]$ contains the indices of the leaf cells, then $L[\boldsymbol{h}a; d]$ denotes the first level containing leaf cells above or at depth d.

$$L[\boldsymbol{h}a; d] = \{(c, i) \in L[\boldsymbol{h}a] : c \geq d, \nexists e \in \{d, \ldots, c-1\}, \exists (e, j) \in L[\boldsymbol{h}a]\}$$

If the thus selected parameter has been sampled less than κ times, it will be returned. Otherwise the selected cell in the parameter space is split. And the current cell is removed from the list of leaf cells.

The POHTP algorithm can be combined with the pruning of equivalent sequences according to Section 3.1.2. Now the equivalence applies not only to action sequences, but to histories where every step consists of an action, an action parameter and the resulting observation. Entire steps can be commuted if the respective actions are independent. Note that no problem-specific structure is exploited in the POHTP algorithm developed in this chapter.

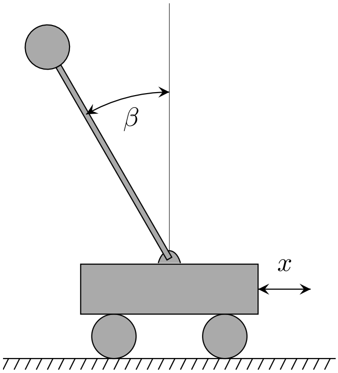


Figure 3.8: The inverted pendulum problem. The goal is to perform a swing-up of the pendulum to an angle $\beta = 0$ and to keep the pendulum in the upright position.

3.3.4 Evaluation

Swingup of an inverted pendulum

Planning for parameterized actions is evaluated for an inverted pendulum displayed in Figure 3.8. The inverted pendulum is one of the canonical problems in the literature on optimal control [Lib11]. A pendulum is attached to a cart that is free to move in the horizontal plane. The goal is to perform a swing-up to bring the pendulum into an upright position – and keep it there – by the precise application of an accelerating force to the cart.

The full problem definition is as follows. Assume a single system component pend for the pendulum and a single action acc with a parameter $\Theta_{\texttt{acc}} = [-2N, 2N]$ for the force applied to accelerate the cart. The cart and the pendulum are approximated by point-masses m_c and m_p of 1kg each. The length of the pendulum l is one meter. The angle of the pendulum β gives the difference from the upright position. The position of the cart x is in meters from the point of origin. The initial state of the pendulum is $s^0_{\texttt{pend}} = (\beta = 0.5,\ \dot{\beta} = 0,\ x = 0,\ \dot{x} = 0)$. The state evolution of the pendulum is described by two coupled differential equations for the cart position x and the pendulum angle β [FYK92]:

$$\begin{aligned} \dot{x} &= \frac{-m_p g \sin(\beta) + m_p l \sin(\beta)\dot{\beta}^2 + u}{m_c + m_p \sin^2(\beta)} \\ \dot{\beta} &= \frac{(m_p + m_c) g \sin(\beta) - m_p l \sin(\beta)\cos(\beta)\dot{\beta}^2 - \cos(\beta) u}{l(m_c + m_p \sin^2(\beta)} \end{aligned} \tag{3.9}$$

The simulation time is discretized into periods of 0.2s. The effect of the action acc is the (numerical) solution to the forward simulation of Equation (3.9) for the control value u given by the action parameter. The reward returned by acc is a cost term associated with the resulting pendulum state s' and the energy expenditure for the control.

$$r_{\mathsf{acc}}(u, s') = 2\|\beta'\| + x + u^2 \tag{3.10}$$

By $\|\cdot\|$ we denote the angular distance from the upright position.

The control (acceleration) applied in each period is the result of POHTP with 512 playouts over a horizon of 15 steps with a 0.2s time discretization. Even though the step length is discretized, the simulation uses the Runge-Kutta method for precise forward-simulation of the underlying differential equation.

After the 512 playouts, the acceleration parameters with the best Q-value is selected and applied. The optimization is then repeated for the resulting system state. This resembles optimal control based on MPC [ML99]. But as we directly optimize on the model from Section 2, we make less limiting assumptions than traditional MPC based on convex optimization.

As can be seen in Figure 3.9, the POHTP algorithm achieves the swing-up and balancing of the inverted pendulum. Note that the angle plateaus at multiples of 2π due to the use of the angular distance in the cost function. Adapting the cost function with a higher penalty for the angle leads to a more speedy swingup. But at a greater cost for the control energy u.

Optimal Order Quantity under Uncertainty

The scenario is concerned with the operations of a pencil factory. A customer gives the order for 50,000 pencils with his company logo on the casing. The customer is willing to pay $2 for each pencil. No payment is made if the order is

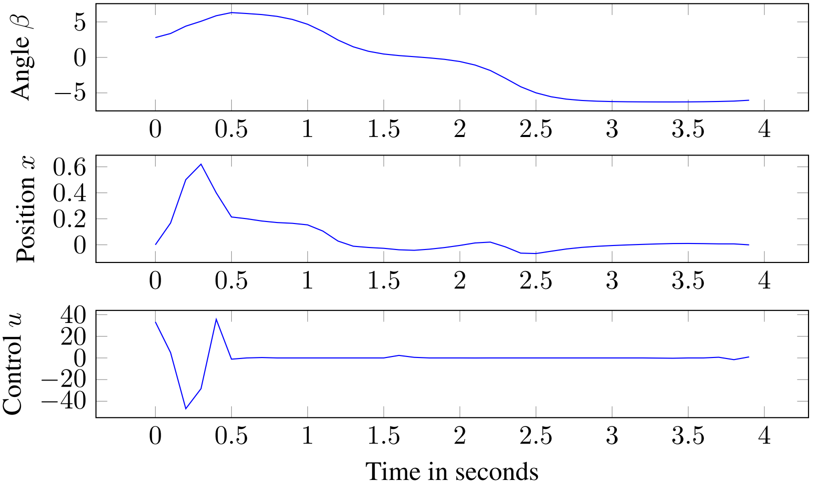


Figure 3.9: Swingup of an inverted pendulum. The initial of the pendulum is at 160 degrees ($\theta = \frac{160}{180}\pi$). The upper time series show the penalization for $|\theta|$. The lower time series show the penalization of $|\operatorname{atan2}(\sin(\theta), \cos(\theta))|$ where "loopovers", e.g. angles at a multiple of 2π are not penalized.

incomplete or arrives too late. The plastic pencil casings with the logo printing are bought from a supplier that demands \$0.5 for each casing. The production costs in our factory are \$1. So, in theory, the pencils are sold to the customer with a margin of \$0.5 each. However, some pencils do not make it through quality control. Every pencil has a 10% chance of being sorted out by an *inline quality control* system that verfies every single product. Due to the time constraints, it is not possible to reorder additional pen casings at the supplier once production has started. The question now is: how many pencil casings should be ordered at the supplier initially in order to maximize the expected earnings?

In order to compare the results of our tree-search planning, we implemented a simple "brute-force" solution technique: Monte-Carlo simulations for all possible solutions in the relevant range. The simulation model is as follows. Assume that n pencil casings have been delivered by the supplier. Start to produce pencils until either 50.000 pencils are done or more than $n - 50.000$ casings were discarded due to quality problems (otherwise, we would continue

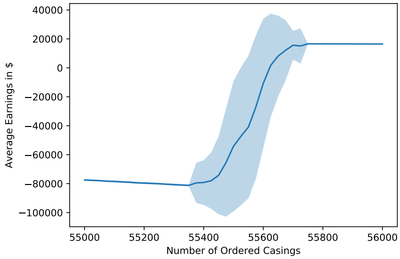


Figure 3.10: Expected reward for different numbers of ordered pencil casings. The average was computed over 100 simulation runs each. The shaded area indicates the empirical standard deviation.

losing money during production without being able to complete the order eventually). Figure 3.10 shows the results of the Monte-Carlo simulation in the range between 55.000 and 56.000 ordered pencil casings. The results show a phase transition between virtually all samples not completing the order to virtuall all samples completing the order. In the transition range, only some orders are completed and this number differs between simulations. Therefore, the standard deviation for the reward is much bigger in the transition range. According to the Monte-Carlo simulations, the optimal order quantity is 55.750, resulting in an expected reward of $1,6575. With less ordered pencil cases, there is a high likelihood of the order not completing. Every additional pencil case incurs higher cost than the additional likelihood of completing the order justifies.

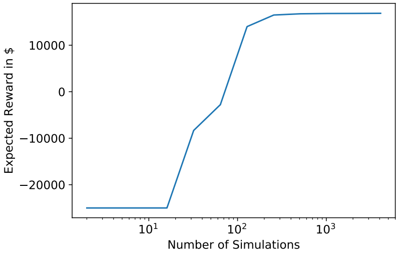


Figure 3.11: Convergence speed of the optimization for the order quantity under uncertainty example.

Figure 3.11 shows the empirical reward of the best paramater after a certain number of plays.

3.4 Planning with Linear Actions

Most manufacturing systems perform repetitive tasks. While lot sizes have generally become smaller, most products are still produced in bulk. When many product instances are considered individually, the action sequences can become very long. This section identifies a large class of actions where reasoning and planning for action repetition is simplified.

Example 3.4. Consider a stamping press that takes in raw material from an aluminum coil. The action `stamp` puts the produced work-pieces of type `p1` into a lattice box adjacent to the press. Every execution of `stamp` increases the number of parts in the lattice box by one and reduces the length of the remaining aluminum coil by 2.5cm. Suppose that 20m of coil are loaded initially. How often can `stamp` be repeated and how many additional parts will be in the lattice box afterward? The answer is of course trivial. But how can this type of reasoning be made accessible to a planning algorithm that operates on the actions from Definition 2.4?

3.4.1 Linear Actions and Action Repetition

Reasoning about the effects of the action `stamp` in Example 3.4 is easy and intuitive. The action has a fixed effect and repeating the action n times multiplies the effect by n. We can compute the maximum number of repetitions of `stamp` that are possible starting from the described initial state. It is implicit to Example 3.4, that if the action `stamp` can be repeated n times, any number of repetitions between zero and n is also possible. We now spell out these implicit assumptions in the form of conditions that so-called *linear actions* have to conform to in addition to Definition 2.4.

Definition 3.7. *An action $\bar{a}$ is linear if the following conditions hold.*

1. *The effect of the action $\boldsymbol{e}_{\bar{a}}$ is the generator of a semimodule [Gol99] $E_{\bar{a}}$ that is closed under composition $(E_{\bar{a}}, \circ)$ and multiplication with non-negative scalars, so that $(\boldsymbol{e}_{\bar{a}} \circ \boldsymbol{e}_{\bar{a}})(\boldsymbol{\sigma}_{\bar{a}}) = (2\boldsymbol{e}_{\bar{a}})(\boldsymbol{\sigma}_{\bar{a}})$ for all feasible $\boldsymbol{\sigma}_{\bar{a}}$.*

2. *Let $\bar{a}^n$ denote the n-fold application of $\bar{a}$. If the action can be repeated n times (that is $\bar{a}^{n-1}(\boldsymbol{\sigma}) \in \Sigma_{\bar{a}}$) then any number of repetitions between zero and n is possible.*

$$\boldsymbol{\sigma} \in \Sigma_{\bar{a}} \wedge \bar{a}^n(\boldsymbol{\sigma}) \in \Sigma_{\bar{a}} \Rightarrow \forall k \in \{0, \ldots, n\},\ \bar{a}^k(\boldsymbol{\sigma}) \in \Sigma_{\bar{a}} \tag{3.11}$$

3. *The action duration is identical for all feasible initial states.*

$$\forall(\boldsymbol{\sigma}, \boldsymbol{\sigma}') \in \Sigma_{\bar{a}} \times \Sigma_{\bar{a}},\ d_{\bar{a}}(\boldsymbol{\sigma}_{\bar{a}}) = d_{\bar{a}}(\boldsymbol{\sigma}'_{\bar{a}}) \tag{3.12}$$

4. *A constant reward $\mathfrak{r}_{\bar{a}}$ is generated for every action repetition.*

Linear actions have advantages over normal actions: First, once the maximum number of repetitions has been established, the preconditions don't need to be verified for every repetition. Second, the effect of repeatedly applying the action can be computed with analytical shortcuts instead of n-fold composition of the effect function.

The notation for repeated application of an action resembles the notation for action parameterization. This is intentional. Repetition of linear actions is a special case in the general framework of parameterized actions. If a parameterized action is also linear, the notation $a^{\theta,n}$ indicates that the same parameter $\theta \in \Theta_{\bar{a}}$ is applied for each of the n repetitions.

The following joke from the mathematical folklore [RD05] sets the frame for discussing the composition of linear actions and the superposition of the effects.

A biologist, a physicist, and a mathematician sit in a street café watching the crowd. Across the street they see a man and a woman entering a building. Ten minutes later they reappear together with a third person.

BIOLOGIST: *They have reproduced.*

PHYSICIST: *The measurement wasn't accurate.*

MATHEMATICIAN: *If exactly one person enters the building now, it will be empty again.*

The mathematician treats the operators “person entering the house” and “person leaving the house” as elements from an algebraic group (entering is the inverse of leaving). The group is indeed closed under composition. But the operator resulting from the composition does not apply to all situations. Obviously, there can be no negative number of people in the house. Translated to our case, components cannot contain a negative number of products. This universal constraint has ramifications on the definition of linear action.

Suppose that for the considered linear action $\bar{a}$, the participating components $C_{\bar{a}}$ can hold products inside the component. So their state is described by a tuple $s = (\xi, \boldsymbol{p})$ for the configuration ξ and the number of contained products for every product type $\boldsymbol{p}$ (cf. Section 2.1). Definition 3.7 implies a linear effect on the contained products in the components. This effect can be expressed by a fixed change vector $\boldsymbol{\delta}_c^{\bar{a}} \in \mathbb{Z}^{|P|}$. So for a state transition $\boldsymbol{\sigma}' = \bar{a}(\boldsymbol{\sigma})$ and component $c \in C_{\bar{a}}$, the state transition is between $s_c = (\xi, \boldsymbol{p})$ and $s_c' = (\xi', \boldsymbol{p}')$ and the product change is $\boldsymbol{p}' = \boldsymbol{p} + \boldsymbol{\delta}_c^{\bar{a}}$.

Since the number of products in the component cannot be negative, there is a universal positivity constraint for all linear actions. The feasible states all conform to the positivity constraint $\Sigma_{\bar{a}} \subseteq \Sigma_{\bar{a}}^{+}$.

$$\Sigma_{\bar{a}}^{+} = \{\boldsymbol{\sigma} \in \Sigma : \forall c \in \bar{C}_{\bar{a}},\ \sigma_c = ((\xi, \boldsymbol{p}), t),\ \boldsymbol{p} + \boldsymbol{\delta}_c^{\bar{a}} \succeq \mathbf{0}\}\,. \tag{3.13}$$

For many linear actions, the condition (3.11) can be shown to hold with a convexity argument. This is illustrated by the following example.

Example 3.5. This example builds on the previous Example 3.4. Suppose $\texttt{stamp}$ is a linear action with the participating components $C_{\texttt{stamp}} = \{\texttt{box}, \texttt{press}\}$. The lattice box has no particular configuration state and $\Xi_{\texttt{box}} = \varnothing$. The configuration of the press $\xi_{\texttt{press}} = (\xi_{\texttt{tool}}, \xi_{\texttt{coil}})$ consists of the press tooling for either product $\texttt{p1}$ or $\texttt{p2}$ and the remaining length of the coil. So the configuration space for the press is $\Xi_{\texttt{press}} = \{\texttt{p1}, \texttt{p2}\} \times \mathbb{R}_{+}$. An additional condition of the press is that at least 50cm of coil need to remain after the action in order to facilitate replenishing. The condition of the lattice box is that the maximum load of 300kg shall not be exceeded.

The effect on the products in the lattice box $\boldsymbol{\delta}_{\mathtt{box}}^{\mathtt{stamp}} = \boldsymbol{\nu}_{\mathtt{p1}}$ contains mostly zeros with a single one-entry at the `p1` position. There is no effect on the lattice box configuration. So the effect on the `box` is $(ne_{\mathtt{stamp}})(\boldsymbol{\sigma}_{\mathtt{stamp}})_{\mathtt{box}} = (\varnothing, \boldsymbol{p}_{\mathtt{box}} + n\boldsymbol{\nu}_{\mathtt{p1}})$. Let the vector $\boldsymbol{\mu}$ describe the weight of every product. Then the initial states that are feasible for the lattice box are

$$\Sigma_{\mathtt{stamp:box}} = \left\{\boldsymbol{\sigma} \in \Sigma : \boldsymbol{\mu}^{\top}(\boldsymbol{p}_{\mathtt{box}} + \boldsymbol{\nu}_{\mathtt{p1}}) \leq 300\right\}.$$

The action has no effect on the products contained in the press and $\boldsymbol{\delta}_{\mathtt{press}}^{\mathtt{stamp}} = \mathbf{0}$. (As before, $\mathbf{0}$ is the null-vector of appropriate size). For simplicity, we refer to the state after executing `stamp` as $\boldsymbol{\sigma}'$ (with analogous notation for its components). The action has no effect on the tooling of the press $\xi'_{\mathtt{tool}} = \xi_{\mathtt{tool}}$ and the remaining coil length is reduced by a fixed amount $\xi'_{\mathtt{coil}} = \xi_{\mathtt{coil}} - 2.5$. So the n-fold repetition has the following effect $(ne_{\mathtt{stamp}})(\boldsymbol{\sigma}_{\mathtt{stamp}})_{\mathtt{press}} = ((\xi_{\mathtt{tool}}, \xi_{\mathtt{coil}} - 2.5n), \mathbf{0})$. At least 50cm of coil need to remain in the press and

$$\Pi_{\mathtt{press}}(F_{\mathtt{stamp}}) = \left\{\boldsymbol{\sigma} \in \Sigma : \xi_{\mathtt{coil}} - 2.5 \geq 50\right\}.$$

The valid initial states for `stamp` must lie in the feasible set for both the press and the lattice box. In addition, no negative number of products must be contained in a component is given by the positivity constraint $\Sigma^{+}_{\mathtt{stamp}}$. In total, the feasible states for beginning the action are

$$\Sigma_{\mathtt{stamp}} = \Sigma_{\mathtt{stamp:box}} \cap \Sigma_{\mathtt{stamp:press}} \cap \Sigma^{+}_{\mathtt{stamp}}.$$

The following proof sketch shows that (3.11) holds for `stamp`. Let $\psi(\boldsymbol{\sigma}) = (\boldsymbol{p}_{\mathtt{box}}, \xi_{\mathtt{coil}})$ a projection of the set of feasible initial states $X = \{\psi(\boldsymbol{\sigma}) : \boldsymbol{\sigma} \in \Sigma_{\mathtt{stamp}}\}$. The three constraints $\Sigma_{\mathtt{stamp:box}}$, $\Sigma_{\mathtt{stamp:press}}$ and

$\Sigma^{+}_{\texttt{stamp}}$ are then expressed as a system of linear inequalities for all $\boldsymbol{x} \in X$.

$$\begin{bmatrix} -\boldsymbol{\mu}^{\top} & 0 \\ \mathbf{0}^{1\times|P|} & 1 \\ \boldsymbol{I}^{|P|} & 0 \end{bmatrix} \boldsymbol{x} \succeq \begin{bmatrix} \boldsymbol{\mu}^{\top}\boldsymbol{\delta}^{\texttt{stamp}}_{\texttt{box}} - 300 \\ 52.5 \\ -\boldsymbol{\delta}^{\texttt{stamp}}_{\texttt{box}} \end{bmatrix} . \tag{3.14}$$

Since X is equivalent to $\{\boldsymbol{x} \in (\mathbb{N}_0^{|P|} \times \mathbb{R}) : \text{condition (3.14) is true}\}$, the space of projected valid initial states is convex. The effect of `stamp` on X is described by a linear operator $f(\boldsymbol{x}) = \boldsymbol{x} + (\boldsymbol{\delta}^{\texttt{stamp}}_{\texttt{box}}, 2.5)$. The operators `stamp` and f are related as $\psi \circ \texttt{stamp} = f \circ \psi$. The n-fold application of f is a linear equation. Due to the described convexity property of X, for all $n \in \mathbb{N}_0$ and initial state representations $\boldsymbol{x} \in X$, there is

$$\left(\boldsymbol{x}+n \begin{bmatrix} \boldsymbol{\delta}^{\texttt{stamp}}_{\texttt{box}} \\ 2.5 \end{bmatrix}\right) \in X \Rightarrow \forall k \in \{0,\dots,n\},\ \left(\boldsymbol{x}+k \begin{bmatrix} \boldsymbol{\delta}^{\texttt{stamp}}_{\texttt{box}} \\ 2.5 \end{bmatrix}\right) \in X\,.$$

Since for every $\boldsymbol{x} \in X$ there exists at least one $\boldsymbol{\sigma} \in \Sigma_{\texttt{stamp}}$ such that $\boldsymbol{x} = \psi(\boldsymbol{\sigma})$, the linear action `stamp` satisfies Equation 3.11.

3.4.2 MILP Relaxation of the Planning Problem

Linear actions were introduced with the promise to simplify reasoning and planning of action sequences with many repetitions. Now we relax the planning problem with linear actions so that it can be solved as a Mixed-Integer Linear Program (MILP) [BW05]. The MILP formulation can be solved with off-the-shelf solvers [Gur16]. In constrast to MCTS, the planning complexity for the relaxed planning problem is mostly independent of the number of repetitions for each action. Used as part of a rollout policy, the MILP relaxation allows the scaling to scenarios with hundreds of individual products that are considered at once. On the downside, it imposes limits on the model dynamics that can be represented.

Assumption 3.8. *In the remainder of this chapter, the following two assumptions are made.*

1. *All considered actions are linear.*
2. *The constraints for the feasible initial states Σ_a, only refer to the number of contained products and not the component configuration.*
3. *All actions can be executed "in parallel" even if they have the same components participating and their effect superimposes for the final system state.*

In the Example 3.5, suppose a second action `take` that takes out one finished piece from the lattice box. In order to take out 500 pieces, we need to run `stamp` 500 times as well. Now that we assume actions can run "in parallel" on the same components, how can the preconditions for the feasible initial states be represented? So far we have worked with feasible initial states Σ_a. For linear actions, this can be transformed to the set of feasible post-states $\Gamma_a \subseteq \Sigma$.

$$\Gamma_a = \{\boldsymbol{\sigma} \in \Sigma : \exists \boldsymbol{\omega} \in \Sigma_a,\ \boldsymbol{\sigma} = a(\boldsymbol{\omega})\} \tag{3.15}$$

For linear actions with a fixed effect vectors $\boldsymbol{\delta}_c^a$, the conversion between Σ_a and Γ_a can be achieved by a simple translation of the constraints describing the set Σ_a. Instead of tracking the feasible initial states before the execution of the actions, we only demand that the final system state, when each action has been repeated the desired number of times, is a feasible post-state for all the actions.

Every action and every component are assigned an index from $\{1, \ldots, |A|\}$ and $\{1, \ldots, |C|\}$ respectively. Let $\boldsymbol{x} = (\boldsymbol{p}_c)_{c \in C}$ denote the concatenated column vector for the products initially contained in the different components. So $\boldsymbol{x}$ is a vector with $|C||P|$ elements. In the second statement of Definition 3.7, it is demanded that a feasible n-fold repetition indicates that any number of repetitions between zero and n must be feasible. Since the effect on the number of contained products (and only these are considered here) is linear, the constraints for the feasible initial states must be the intersection of a convex set with the set of integers. Otherwise, it would be possible to find a system stat $\boldsymbol{\sigma}$ where the action a in question can be executed n times but not $n - 1$ times. As the constraints encoded in Σ_a (and therefore also Γ_a) are convex in that sense, they can be represented as the intersection of half-spaces via a set of linear inequalities [BV04] defined by a matrix $\boldsymbol{H}^a$ and vector $\boldsymbol{g}^a$, such that $\boldsymbol{H}^a \boldsymbol{x} \succeq \boldsymbol{g}^a$.

In a production scenario, most actions are associated with costs for material, energy, worker's wages, and so on. But some actions have positive reward, such as finishing an order for the customer. After completing the order, we could make more products. But if the customer won't pay for them they only incurr costs. This is represented for the MILP as follows: Denote with the vector $\boldsymbol{n}$ the number of repetitions for each action. The vector $\mathfrak{r}$ contains the costs incurred for every repetition of the actions.

Goals are defined by a number of target repetitions $\boldsymbol{n}^g \in N_0^{|A|}$ for every action. Each repetition of the action a up to n_a^g yields the additional goal reward $\mathfrak{r}_a^g$. The total goal reward for the repetitions $\boldsymbol{n}$ – in addition to the reward generated from each action's fixed reward $\mathfrak{r}_a$ – is

$$\sum_{a \in A} \left[\min\{n_a, n_a^g\}\, \mathfrak{r}_a^g \right] . \tag{3.16}$$

The MILP computes (3.16) by the introduction of a slack variable $\boldsymbol{v}$ that counts the missing repetitions for each action according to the goal definition. The objective function takes the reward for reaching all goals and subtracts the missing repetitions according to the slack variable.

The column vector $\boldsymbol{\delta}_a \in \mathbb{N}^{|C||P|}$ describes the effect of action a on the products contained in all components. These effect vectors are assembled to a matrix $\boldsymbol{\Delta} \in \mathbb{Z}^{|C||P| \times |A|}$ for the effect across all actions.

$$\boldsymbol{\delta}_a = (\boldsymbol{\delta}_c^a)_{c \in C}, \quad \boldsymbol{\Delta} = [\boldsymbol{\delta}_1, \ldots, \boldsymbol{\delta}_{|A|}] \tag{3.17}$$

The post-state after executing all action repetitions $\boldsymbol{n}$ is $\boldsymbol{x}' = \boldsymbol{x} + \Delta \boldsymbol{n}$. Optimizing the repetitions to maximize the reward under the defined constraints then gives the MILP formulation:

$$\bar{V}(\boldsymbol{x}, \boldsymbol{n}^g, \boldsymbol{r}^g) = \max_{\boldsymbol{n} \in \mathbb{N}_0^{|A|}} \left[\boldsymbol{n}^\top \mathfrak{r} - \boldsymbol{v}^\top \boldsymbol{r}^g \right] + {\boldsymbol{n}^g}^\top \boldsymbol{r}^g \tag{3.18}$$

such that

$$\boldsymbol{x} + \boldsymbol{\Delta n} = \boldsymbol{x}' \tag{3.19}$$

$$\boldsymbol{H}^a \boldsymbol{x}' \succeq \boldsymbol{g}^a, \qquad \forall a \in A \tag{3.20}$$

$$\boldsymbol{n} + \boldsymbol{v} \succeq \boldsymbol{n}^g \tag{3.21}$$

$$\boldsymbol{x}' \succeq \boldsymbol{0}, \quad \boldsymbol{n} \succeq \boldsymbol{0}, \quad \boldsymbol{v} \succeq \boldsymbol{0} \tag{3.22}$$

$$\boldsymbol{x}' \in \mathbb{R}_0^{|C||P|}, \boldsymbol{n} \in \mathbb{N}_0^{|A|}, \quad \boldsymbol{v} \in \mathbb{R}_0^{|A|} \tag{3.23}$$

Function $\bar{V}$ approximates the value of the system state $\boldsymbol{x}$ (with only the contained products) for optimal decision making in the MILP relaxation. The goal is to maximize the reward, including the goal reward. The constant additive term $\boldsymbol{n}^{g\top} \boldsymbol{r}^g$ can be removed for the actual optimization. But it is required to recover the actual V-value for the relaxed planning problem including the goal reward. Lagrangian relaxation is used to penalize if an action a is repeated less than n_a^g times. The slack vector $\boldsymbol{v}$ gives the number of repetitions lacking for every action. (If the goal is met, the slack variable is zeroed out by the optimizer.) In practice, the dimensionality of $\boldsymbol{x}$ and $\boldsymbol{\Delta}$ can be reduced by considering only the products that are actually referred to by the linear actions. The maximum number of repetitions for each action is $n^{\max}$. This maximum number of repetitions is only introduced to model binary values: The vector $\boldsymbol{m}$ contains binary values for the fixed reward incurred if an action is repeated at least once.

The constraints for the optimization are as follows. The post-state $\boldsymbol{x}'$ after all repetitions have been executed is computed in (3.19). The post-conditions for all actions must hold simultaneous for $\boldsymbol{x}'$ according to (3.20). In (3.21), the slack value $\boldsymbol{v}$ is set to the number of missing repetitions according to the goal definition. The constraints in (3.22) ensure that the number of products in the final state, the number of repetitions and the slack repetitions are all non-negative. In (3.23), the repetitions $\boldsymbol{n}$ are required to be integers. The number of remaining products $\boldsymbol{x}'$ and the slack $\boldsymbol{v}$ are real values. But they will only take on integer values since the repetitions are a natural number and the system dynamics in $\boldsymbol{\Delta}$ leads to integer changes.

For every action selection in the rollout policy, the relaxed planning problem is solved. Actions that are slated for zero repetitions by the relaxed solution and actions that are not immediately executable due to their preconditions are ignored. A sensitivity analysis is performed for the remaining actions: For every action a modified version of the original MILP is solved where the number of repetitions for that action is fixed to be one less than in the original solution. The difference of the new solution in the objective function is a grade for "importance" of that action. The rollout policy then returns the action with the highest importance and the number of repetitions chosen for the action via the MILP relaxation.

Algorithm 13 Rollout policy for linear actions. Takes as input the current state and returns a linear action for the next step and its repetitions.

```
**procedure** $\pi_{\text{linear}}(\boldsymbol{\sigma}, \boldsymbol{h}, \boldsymbol{n}^g, \boldsymbol{r}^g)$
    $\boldsymbol{x} \leftarrow (\boldsymbol{p}_c)_{c \in C}$    ▷ State vector of contained products in all components
    $\boldsymbol{n}^* \leftarrow \bar{V}(\boldsymbol{x}, \boldsymbol{n}^g, \boldsymbol{r}^g)$    ▷ Optimal repetitions in the MILP relaxation
    $B \leftarrow \{b \in A : \boldsymbol{\sigma} \in \Sigma_b \wedge n_b^* > 0 \wedge \textsc{TestLNF}(\boldsymbol{h}, b)\}$
    $b \sim \mathcal{U}(B)$    ▷ Uniform sampling among the eligible actions
    $n' \leftarrow \max\{m \in \{1, \ldots, n_b^*\} : b^{m-1}(\boldsymbol{\sigma}) \in \Sigma_b\}$
    **return** $(b, n')$
```

The above description of the setting can be translated to a MILP. Since all actions are linear, no additional relaxations are required besides the assumption that actions can execute "in parallel". By using the solution to the linear program to guide the rollout, the global optimum is found already in the first rollout.

3.4.3 Evaluation

Consider a simplified supply chain for the production of mobile phones. See Figure 3.12 for an overview. The OEM (Original Equipment Manufacturer) owns the phone brand as well as production sites for soldering, assembly and packaging. Parts are bought from suppliers. The final phone is assembled from a case, a battery, a screen and a PCB (printed circuit board) with a chipset soldered on. If the production capacity of the OEM is insufficient, assembled phones can be bought from an external contract manufacturer. The cost for soldering, assembly and packaging are \$10 each. Transportation costs are not assumed for the example. The following prices are demanded by the suppliers.

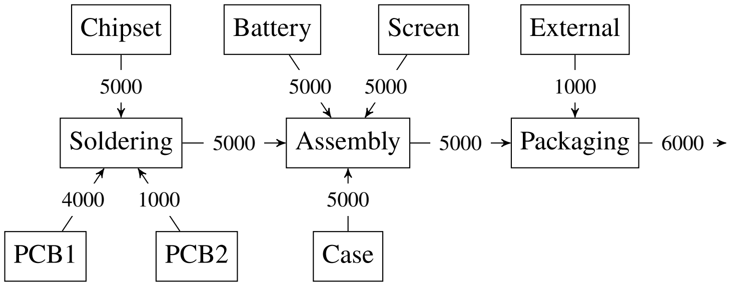


Figure 3.12: Supply chain example. The arrows denote the possible number of transported products between production sites and suppliers.

- PCB1: $5
- PCB2: $2
- Chipset: $20
- Battery: $30
- Screen: $30
- Case: $10
- Assembled Phone: $150

The supplier PCB1 has limited stock and can deliver at most 1,000 PCB. The chipset supplier has limited stock of 5,000 remaining chipsets. As a consequence, the first 1,000 phones cost $122 to make (bill of material and production costs). PCB for additional phones have to be bought from the alternative supplier PCB2 at a higher price. The phones then cost $125 to make. For more than 5,000 phones, the required chipset is no longer available. But assembled phones can still be bought from the contract manufacturer. This comes at the increased cost of $160 for each phone: $150 for the phone and $10 for branding and packaging. So buying from the contract manufacturer is more expensive than producing the phones in the OEM's production facilities. It might however be required to buy assembled phones in order to complete a large order.

Consider now a scenario where a customer orders 6,000 phones for the price of $175 each. What are the maximum earnings (revenue minus cost) the OEM can achieve in each scenario? For this supply-chain example, the optimization problem was solved exactly by the MILP. Hence the decisions by the rollout policy immediately led to the globally optimal action and parameter sequence. (This is not the case for all planning problems. For example when only a subset of the actions is linear.) The MILP was solved with the commercial solver Gurobi [Gur16]. For details, refer to the literature for optimization of convex functions and optimization over integers [BV04; BW05]. The phones are sold for \$1,050,000 and were produced at a cost of \$762,000. This leaves a profit margin of \$288,000.

4 Distributed Planning for Self-Organizing Production Systems

> *Outside the firm, price movements direct production, which is co-ordinated through a series of exchange transactions on the market. Within a firm, these market transactions are eliminated and in place of the complicated market structure with exchange transactions is substituted the entrepreneur-coordinator, who directs production. It is clear that these are alternative methods of coordinating production. Yet, having regard to the fact that, if production is regulated by price movements, production could be carried on without any organization at all might we ask, why is there any organization?*
>
> *Ronald H. Coase [Coa37]*

The coordination of industrial production is historically performed either by a central planner or market-mechanisms for coordination. The former is fraught with the problem of keeping the model for planning up-to-date and the complexity of planning itself. The latter has the problem of suboptimal solutions arising from market-based coordination. The core idea of markets is to have selfish participants maximize their personal gain. Under some technical conditions, markets are "efficient" for the incorporation of information into prices and the allocation of goods according to a preference function of the buyers [MF70]. From Game Theory, we know the existence of suboptimal equilibria in situations with competing agents where no participant has an incentive to change his strategy even though an equilbrium with higher overall welfare exists [Nas51]. In this thesis we instead assume cooperating agents that aim to jointly maximize the overall reward.

This chapter extends the model from Chapter 2 to include multiple agents that coordinate their actions in a distributed fashion. Afterwards the POHTP algorithm from Chapters 3 is adapted for the distributed setting. The result is a distributed planning algorithm where agents exchange messages for coordination via "utility propagation". The postulate for this chapter is the following:

> *Independent agents can jointly perform planning in a production scenario where every agent only has a simulation model of the system part in his visible scope.*

4.1 Background: The Generalized Distributive Law

Judea Pearl introduced Belief Propagation (BP) as a way to efficiently compute inference tasks on (conditional) probability distributions [Pea88]. The algorithms that perform BP have become known as "message passing" algorithms since they are based on the exchange of messages representing conditional distributions between nodes in a graph [KF09]. In the years following the publication of [Pea88] the similarities between BP and other preexisting techniques in different scientific communities have been discovered. The common core of these techniques has been developed into the Generalized Distributive Law (GDL) family of algorithms [AM00; KFL01]. The GDL comprises as special cases the Baum-Welch algorithm for state estimation in Hidden Markov Models [Wel03], the Max-Plus algorithm for finding the maximum a-posteri event in a probability distribution, Turbo codes for error correction on noisy communication channels, and many more. We now summarize message passing for the distributed optimization of a function. Full proofs are omitted here. They can be found together with more pointers to the literature in [AM00; KF09].

The function $g : X \to \mathbb{R}$ is defined for the domain X. For simplicity, let X contain only a finite number of members. The domain decomposes into variables v such that $X = \times_{v \in V} X_v$. We write $\boldsymbol{x} \in X$ for the vector with

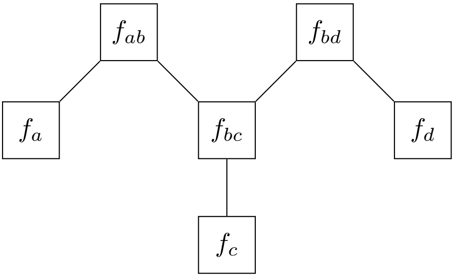


Figure 4.1: Factor graph for a problem decomposition. Factors that share are variable are neighbors and connected with an edge. The subscript indicates the variables in the domain of the factor.

entries $x_v \in X_v$. The function g decomposes into a sum of factors $f \in F$.

$$g(\boldsymbol{x}) = \sum_{f \in F} f(\boldsymbol{x}_f)$$

Every factor depends on a subset of the variables $V_f \subseteq V$ and $X_f = \times_{v \in V_f} X_v$. In the context of a vector $\boldsymbol{x}$, the projection of $\boldsymbol{x}$ on the variables in the domain of f is written as $\boldsymbol{x}_f$. The goal is to find the maximizer $\boldsymbol{x}^* = \arg\max_{\boldsymbol{x} \in X} g(\boldsymbol{x})$. Usually, the optimization for each factor $\arg\max_{\boldsymbol{x}_f \in X_f} f(\boldsymbol{x}_f)$ is much easier as it only has to consider a fraction of the full domain X.

Now we constrain the domain for the optimization by fixing some of the variables. Assume that the variables in the scope of the factor f have been fixed to some $\boldsymbol{y}_f \in X_f$. The optimization problem with this additional constraints is said to be conditioned on $\boldsymbol{y}_f$:

$$\arg\max_{\boldsymbol{x} \in X \mid \boldsymbol{y}_f} g(\boldsymbol{x}) \tag{4.1}$$

Since the domain X_f is finite, we can write a table with the results of Equation 4.1 for each $\boldsymbol{y}_f \in X_f$. This changes the perspective of the optimization. We can ask which $\boldsymbol{y}_f \in X_f$ is best, knowing what the optimal "reaction" will be. Tabular representations of this kind (for finite domains) are the messages that are exchanged in the message passing algorithms.

If the factors form a tree-graph, then the computational effort for solving the optimization problem can be reduced drastically. Figure 4.1 represents the factors of an example problems as nodes. The factor name indicates the variables in the factor domain. So the factor f_{bd} has the domain $X_{f_{bd}} = X_b \times X_d$. This tree structure is a so-called junction tree with respect to the variables of each factor [Cow+99]. In a junction tree, nodes can be connected (are neighbors) if they share at least one variable. They don't have to be connected if they share a variable. But if they do share a variable, then all nodes on the paths between these two nodes must also refer to that variable. An example in Figure 4.1 are the nodes f_{ab} and f_{bd}. They share the variable b. So all nodes on a path between f_{ab} and f_{bd} must have b in their domain for the graph to be a junction tree.

On a tree-graph, every edge separates the tree into two otherwise unconnected halves. Take the edge (f_{ab}, f_{bc}) in Figure 4.1. Cutting at the edge splits the set of factor functions into disjoint sets $F = F_{ab} \cup F_{bc}$. This results in two smaller optimization problems $g_{ab}(\boldsymbol{x}) = \sum_{f \in F_{ab}}(\boldsymbol{x}_f)$ and $g_{bc}(\boldsymbol{x}) = \sum_{f \in F_{bc}}(\boldsymbol{x}_f)$. Note that, given a fixed assignment to x_b, the two subproblems are "conditionally independent" from one another. That is, for a fixed x_b, the overall optimization problem can be solved by optimizing each subproblem individually and merging the partial solutions.

By convention, we denote the nodes representing factor functions as i and $j \in N(i)$. The set $N(i)$ contains the neighbors with a direct edge to i. The subtree behind the edge (i, j) on the side of i contains the factor functions $F_{i \to j} \subseteq F$. The factors i and j share the variables $V_{ij} = V_i \cap V_j$ with the domain $\boldsymbol{x}_{ij} \in X_{ij}$. The message sent from i to j then is (a tabular representation of) a function $\mathfrak{m}_{i \to j} : X_{ij} \to \mathbb{R}$. It contains the value of the best-possible solution for the subproblem with the factors $F_{i \to j}$ conditioned on the shared domain.

$$\mathfrak{m}_{i \to j}(\boldsymbol{y}_{ij}) = \underset{\boldsymbol{x} \in X \mid \boldsymbol{y}_{ij}}{\arg\max} \sum_{f \in F_{i \to j}} f(\boldsymbol{x}_f) \tag{4.2}$$

In the junction tree, the messages $\mathfrak{m}_{i \to j}$ can be computed in such a way that the computation at every node i only considers the domain X_i.The result of Equation 4.2 can then be computed by only considering the local factor and the received messages:

$$\mathfrak{m}_{i\to j}(\boldsymbol{y}_{ij}) = \operatorname*{arg\,max}_{\boldsymbol{x}_i \in X_i \,|\, \boldsymbol{y}_{ij}} \left[f_i(\boldsymbol{x}_i) + \sum_{l \in N(i)\setminus\{j\}} \mathfrak{m}_{l\to i}(\boldsymbol{x}_{il}) \right] \tag{4.3}$$

In a so-called forward-backward pass, the messages are first sent out by the edge-nodes with only one neighbor. Here, computing the message with Equation 4.2 is straightforward, as it only requires access to the factor function of the node itself. Other nodes compute and send out a message to their neighbor j as soon as a message has arrived from all other neighbors (not considering the receiving neighbor j). In a tree-graph, the exchanged messages converge after a forward-backward pass when a message has been sent over every edge in both directions. Every node i then chooses the solution

$$\operatorname*{arg\,max}_{\boldsymbol{x}_i \in X_i \,|\, \boldsymbol{y}_{ij}} \left[f_i(\boldsymbol{x}_i) + \sum_{l \in N(i)} \mathfrak{m}_{l\to i}(\boldsymbol{x}_{il}) \right] \tag{4.4}$$

from his domain X_i. If the solution of Equation 4.4 is unique at every node, then the nodes agree with respect to the assignment of shared variables and the joint assignment of values to $\boldsymbol{x}$ is optimal. Additional communication is required to break ties. In practice, small random disturbances added to the values of the exchanged messages prevent ties effectively. Message passing generally works on loopy graphs as well. The convergence is then not guaranteed. Still, the results are often surprisingly good. If convergence in a loopy graph is achieved, the solution quality can be characterized according to the so-called Bethe free energy [YFW01].

The algorithm just presented is known under the name “Max-Plus” or “Max-Sum” according to the operations for joining partial solutions and marginalization. The general approach also works in any algebraic semiring where the operators $\max$ and $+$ are replaced with their respective counterparts. The underlying principle of the message-passing algorithm is traced back to the distributive property of the two operators of the semiring, hence the name Generalized Distributive Law. In this text, we are only considering the GDL for finite domains. See the publication [WJ+08] for the application of the GDL to inference on continuous probability distribution from the exponential family.

4.2 A Model of Distributed Concurrent Production Systems

The established models for multi-agent coordination decompose the planning problem in such a way that the sets of actions available to each agent are disjoint. Compare for example with the popular MA-STRIPS [BD08] and DEC-POMDP [BZI00] models. We take a different route. The actions available to every agent and the components that are visible in their scope overlap. This overlap is the common language that is required for coordination. Informally spoken, the overlap acts as the “hinge” between the per-agent models.

Definition 4.1. *A distributed planning problem is represented by a tuple*

$$\left(C, A, \boldsymbol{\sigma}^0, I, \{C_i, A_i, \mathfrak{r}_i\}_{i \in I}\right) .$$

The definitions for the set of components C, the actions A and the initial system state $\boldsymbol{\sigma}^0$ are identical to the central planning problem from Chapter 3. The additional agents $i \in I$ each have a limited scope with regards to the part of the overall system that is visible to them. This is reflected in the visible components $C_i \subseteq C$, and the actions $A_i \subseteq A$ with parameters from Θ_i. The joint state of the components in the scope of i is $\boldsymbol{\sigma}_i$ with the state-space $\Sigma_i = \Sigma_{C_i}$. Each agent has a private reward function $\mathfrak{r}_i : \Sigma_i \times A_i \times \Theta_i \times \Sigma_i \to \mathbb{R}$. The components and actions in the scope of two agents i and j can overlap. The shared components and actions are $C_{ij} = C_i \cap C_j$ and $A_{ij} = A_i \cap A_j$. Two agents are considered neighbors if they share a component in their scope. The set $J(i)$ contains the neighbors of the agent i. The neighbor relation is of course symmetric $j \in N(i) \Leftrightarrow i \in N(j)$.

$$J(i) = \{j \in (I \setminus \{i\}) : C_i \cap C_j \neq \varnothing\}$$

Example 4.1. Consider again the manufacturing scenario from Example 2.2. Now, two agents jointly control the system. One agent is responsible for production and the other for packaging. Each agent has only a subset of the system components in his local scope. The lattice box in the middle is in the scope of both agents. See Figure 4.2 for details.

The agents see all actions where a component in their scope participates. Therefore, $C_{\texttt{prod}} = \{\texttt{produce}, \texttt{put}, \texttt{take}\}$ and $C_{\texttt{pack}} = \{\texttt{put}, \texttt{take}, \texttt{package}\}$. But since the agent prod has no visibility for the packaging robot, he can only have a partial view on the action take. The same is true for the pack agent and the action put.

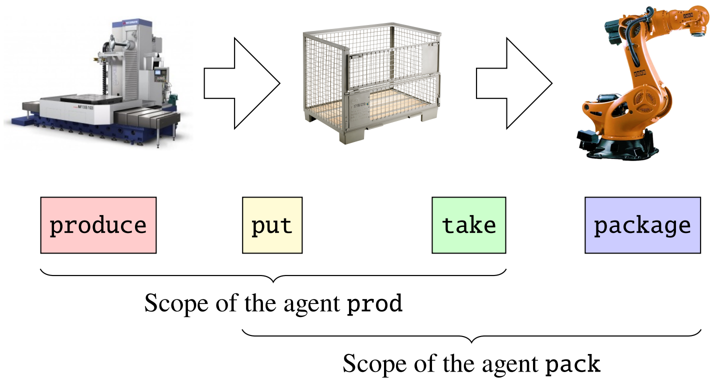


Figure 4.2: Minimal production scenario from Example 2.2 with two agents and their respective scope.

Every agent i is equipped with a simulation model of the components and actions in his scope. An agent may have actions a in his scope where some of the components participating in the action are outside of the scope of i so $C_a \nsubseteq C_i$. It would be preferable that all actions entirely fit into the scope of every participating agent. But then we could not correctly model interactions across the boundaries of an agent scope. Take the situation of Example 4.1. Products are moved from the machine tool to the packaging robot. For this, the products leave the scope of the agent prod and enter the scope of the agent pack. The component of the lattice box is shared by both agents. More formally, $\texttt{box} \in (C_{\texttt{prod}} \cap C_{\texttt{pack}})$. The agent prod must be able to predict – in his private model of the system dynamics – when the lattice box will be free again. But he does not see the packaging robot who takes out products. If the action representations for the individual agents were to include all participating components and the agents know all actions that act on components in their scope, then the agents would have to keep all components C in their scope.

To overcome this, each agent has an internal representation of the actions that only describes the preconditions and effect on components that are in the agents scope. In Example 4.1, the agent `prod` has a partial representation of the action `take` to work with action sequences where multiple products are sequentially moved to `pack`. From the perspective of the agent `prod`, the products simply disappear when they are actually moved to the component `package`. The remainder of this section spells out the assumptions that are required for the agents' individual partial system models to be mutually compatible. This is required for the distributed planning methods introduced in the later sections of this chapter.

Definition 4.2. *Based on Definition 2.4, the projection of an action a to the scope of an agent i is*

$$a_i = (C_{a,i}, \bar{\Sigma}_{a,i}, \boldsymbol{e}_{a,i}, d_{a,i})$$

- *with participating components $C_{a,i} = C_a \cap C_i$,*
- *feasible initial states $\bar{\Sigma}_{a,i} = \Pi_{C_{a,i}}(\bar{\Sigma}_a)$, and*
- *effects and durations $\boldsymbol{e}_{a,i} : \bar{\Sigma}_{a,i} \to S_{C_{a,i}}$ and $d_{a,i} : \bar{\Sigma}_{a,i} \to \mathbb{R}_+$.*

Similar to the global action definition, per-agent actions a_i are operators on the domain $\Sigma_{a,i} = \{\boldsymbol{\sigma}_i \in \Sigma_i : \Pi_{C_{a,i}}(\boldsymbol{\sigma}_i) \in \bar{\Sigma}_{a,i}\}$. The full operator signature is $a_i : \Sigma_{a,i} \to \Sigma_i$.

Assumption 4.3. *If a component in the scope of an agent i participates in an action a, then the projected action a_i is in the scope of that agent.*

$$\forall i \in I,\, \forall a \in A,\, C_{a,i} \neq \varnothing \Rightarrow a_i \in A_i$$

The participating agents of the action a are those with at least one participating component in their scope $I_a = \{i \in I : C_{a,i} \neq \varnothing\}$. From the definition of $\bar{\Sigma}_{a,i}$, the projected action imposes less constraints on the feasible initial states. The inverse projection of the feasible set is $\Pi^{-1}_{C_{a,i}}(\bar{\Sigma}_{a,i}) = \{\boldsymbol{\sigma} \in \Sigma : \Pi_{C_{a,i}}(\boldsymbol{\sigma}) \in \bar{\Sigma}_{a,i}\}$. Since less constraints are imposed on feasible initial states $\Sigma_a \subseteq \Pi^{-1}_{C_{a,i}}(\bar{\Sigma}_{a,i})$. So there may be global states $\boldsymbol{\sigma}$ where agent i beliefs $\boldsymbol{\sigma}_i$ to be feasible for his projected action a_i so that $\boldsymbol{\sigma}_i \in \Sigma_{a,i}$ but which are not feasible for the original action $\boldsymbol{\sigma} \notin \Sigma_a$. In order to prevent the agents from jointly selecting an action that is infeasible for the current global system state $\boldsymbol{\sigma}$ (and possible damaging equipment or endangering human operators) we assume that the action definitions and the decomposition into agents does not lead to infeasible action selections if the agents jointly agree on the feasibility.

Assumption 4.4. *If the participating agents $i \in I_a$ agree that action a is feasible based on their individual projected action a_i with preconditions $\Sigma_{a,i}$, then the action is also globally feasible.*

$$\Big(\bigcap_{i \in I_a} \Pi^{-1}_{C_{a,i}}(\bar{\Sigma}_{a,i}) \Big) \subseteq \Sigma_a$$

Assumption 4.4 implies restrictions for the possible preconditions with respect to timing and synchronization between agents. The feasibility of an action can not depend on the simulation time of a participating components outside the scope of a participating agent. Otherwise, it would be possible to construct situations where all agents jointly, but incorrectly, belief an action to be feasible. Suppose a situation where molten iron ore is transferred from a component exclusively in the scope of agent i to a component exclusively in the scope of agent j. The global action for the transfer rightly imposes constraints on the simulation time to ensure that the component with the molten ore does not idle for too long. But this timing constraint cannot be represented in the projected action for either agent, violating Assumption 4.4. If timing conditions are critical, then all participating need to be in the scope of the agents.

In addition to assumptions for the feasible initial states, we limit the effect so that the post-state of the components $C_{a,i}$ is correctly predicted by a_i. That means for deterministic scenarios that the effect on the components $C_{a,i}$ follows from the initial state of the $C_{a,i}$. In stochastic scenarios, the distribution for the post-states of the components $C_{a,i}$ is conditionally independent from components outside of $C_{a,i}$.

Assumption 4.5. *If an action a has a participating agent i, then the effect on the components $C_{a,i}$ is determined by the initial state of the components $C_{a,i}$ only. If the action is deterministic, then*

$$\forall \boldsymbol{\sigma}, \boldsymbol{\omega} \in \Sigma_a,\ \boldsymbol{\sigma}_{a,i} = \boldsymbol{\omega}_{a,i} \Rightarrow \Pi_{C_{a,i}}(\boldsymbol{e}_{a,i}(\boldsymbol{\sigma})) = \Pi_{C_{a,i}}(\boldsymbol{e}_{a,i}(\boldsymbol{\omega}))\,.$$

If the action is stochastic, so that the post-state and observations are sampled as $(\boldsymbol{\sigma}', \boldsymbol{o}_a) \sim \underline{a}(\boldsymbol{\sigma})$ with $\boldsymbol{s}'$ the untimed state of the components from $\boldsymbol{\sigma}'$, then

$$(\boldsymbol{s}'_{a,i}, \boldsymbol{o}_{a,i} \,|\, \boldsymbol{\sigma}_{a,i}) \ \perp\ \boldsymbol{\sigma}_{C \setminus C_i}\,.$$

Note that Assumption 4.5 restricts the effect on the resulting state $\boldsymbol{s}_{a,i}$, but not on the resulting simulation time of the components. This allows the time synchronization of components across the scope of a single agent.

Last, we require that the reward generated by the actions does not depend on the simulation time of the components. This will become important later on, when the agents predict their reward based on an internal simulation model that is restricted to their scope.

Assumption 4.6. *For any two system states $\boldsymbol{\sigma}$ and $\boldsymbol{\sigma}'$ where the untimed component states are identical $\boldsymbol{s} = \boldsymbol{s}'$, the reward from any feasible action a is identical*

$$\mathfrak{r}(\boldsymbol{\sigma}, a, \theta, a(\boldsymbol{\sigma})) = \mathfrak{r}(\boldsymbol{\sigma}', a, \theta, a(\boldsymbol{\sigma}'))\,. \tag{4.5}$$

If an action a is completely outside the scope of agent i, then the projection is the identity action ε – also used to denote an empty action sequence. The identity action can be simply omitted in an action sequence.

$$C_a \cap C_i = \varnothing \Rightarrow a_i = \varepsilon$$

Action sequences $\boldsymbol{w}$ are projected to the scope of agent i as

$$\Pi_i(\boldsymbol{w}) = \boldsymbol{w}_i = (w_i^k)_{k \in \{1,\ldots,|\boldsymbol{w}|\}}\,.$$

Since actions project to the identity ε for an agent i that is not participating in it, the sequence $\boldsymbol{w}_i$ may contain less elements than $\boldsymbol{w}$. We continue to use the same index notation k for both global and per-agent action sequences and make the number of sequence members explicit only when this is needed.

As described in Section 2.1, the set $W = A^*$ contains all action sequences of finite length generated from a set of base actions A. It implies a tree-graph where every edge denotes an action that is appended to the previous sequence. The sequence trees $W^{\boldsymbol{\sigma}}$ with a defined initial state $\boldsymbol{\sigma}$ contains only feasible sequences starting from the initial state. The sequence tree $W_i = A_i^*$ considered by the agent i contains all sequences formed from the actions in i's scope. The sequence tree $W_i^{\boldsymbol{\sigma}}$ contains the sequences from W_i that agent i beliefs to be feasible starting from the initial state $\boldsymbol{\sigma}_i$. The inverse projection of the per-agent sequence tree $\Pi_i^{-1}(W_i^{\boldsymbol{\sigma}})$ contains all global sequences that are compatible with (project to) a sequence from $W_i^{\boldsymbol{\sigma}}$ and that are also feasible for some compatible initial state $\boldsymbol{\omega}$ with $\boldsymbol{\omega}_i = \boldsymbol{\sigma}_i$.

$$\Pi_i^{-1}(W_i^{\boldsymbol{\sigma}}) = \{\boldsymbol{w} \in W : \exists \boldsymbol{\omega} \in \Sigma,\ \boldsymbol{\omega}_i = \boldsymbol{\sigma}_i,\ \boldsymbol{w} \in W^{\boldsymbol{\omega}},\ \boldsymbol{w}_i \in W_i^{\boldsymbol{\sigma}}\} \quad (4.6)$$

Since the considered system dynamics takes concurrency into account, the index of an action in the action sequence could no longer coincide with the order in which the actions are executed according to the simulation time. The state of component c after executing the first k actions is $w^{:k}(\boldsymbol{\sigma})_c = \sigma_c^k = (s_c^k, t_c^k)$.

Proposition 4.7. *From the Assumptions 4.3, 4.4 and 4.5 follows that for any global state $\boldsymbol{\sigma} \in \Sigma$ and agent $i \in I$*

$$\Big(\bigcap_{j \in I} \Pi_j^{-1}(W_j^{\boldsymbol{\sigma}})\Big) = W^{\boldsymbol{\sigma}} \subseteq \Pi_i^{-1}(W_i^{\boldsymbol{\sigma}}) .$$

Proof. Consider the subset relation $W^{\boldsymbol{\sigma}} \subseteq \Pi_i^{-1}(W_i^{\boldsymbol{\sigma}})$. Assume there exists a sequence $\boldsymbol{v} \in W^{\boldsymbol{\sigma}}$. But an agent i beliefs that the projected sequence is not feasible for his scope so that $\boldsymbol{v} \notin \Pi_i^{-1}(W_i^{\boldsymbol{\sigma}})$. Let k the index in $\boldsymbol{v}$ where $\boldsymbol{v}^{:k} \in \Pi_i^{-1}(W_i^{\boldsymbol{\sigma}})$. Such a k must exist since the empty sequence that is always feasible. From Equation 4.6 the agent i regards the shortened sequence as feasible $\Pi_i(\boldsymbol{w}^{:k}) \in W_i^{\boldsymbol{\sigma}}$. A consequence of Assumption 4.3 and Assumption 4.5 is that the agent i correctly predicts the (untimed) system state of the components in his scope based on the projected sequence $\boldsymbol{w}_i^{:k}$.

$$\forall \boldsymbol{\sigma} \in \Sigma_{\boldsymbol{w}^{:k}},\ \boldsymbol{\sigma}' = \boldsymbol{w}^{:k}(\boldsymbol{\sigma}),\ \boldsymbol{\omega}_i = \boldsymbol{w}_i^{:k}(\boldsymbol{\sigma}_i),\ \boldsymbol{s}_i^{\boldsymbol{\sigma}'_i} = \boldsymbol{s}_i^{\boldsymbol{\omega}_i}$$

Here, $\boldsymbol{s}_i^{\boldsymbol{\sigma}'}$ and $\boldsymbol{s}_i^{\boldsymbol{\omega}_i}$ denote the untimed state of the components in i's scope in the respective timed state vectors $\boldsymbol{\sigma}'$ and $\boldsymbol{\omega}_i$. Since $\boldsymbol{w} \in W^{\boldsymbol{\sigma}}$ we know that $\boldsymbol{\sigma}' \in \Sigma_{w^{k+1}}$. From the preconditions of projected action from Definition 4.2 it must be that $\boldsymbol{\omega}_i \in \Sigma_{w^{k+1},i}$. This contradicts the initial assumption.

We first show that $\left(\bigcap_{j\in I} \Pi_j^{-1}(W_j^{\boldsymbol{\sigma}})\right) \subseteq W^{\boldsymbol{\sigma}}$. Assume an action sequence $\boldsymbol{u}$ where $\boldsymbol{u} \notin W^{\boldsymbol{\sigma}}$ and $\boldsymbol{u} \in \left(\bigcap_{j\in I} \Pi_j^{-1}(W_j^{\boldsymbol{\sigma}}\right)$. There exists an index l such that the subsequence $\boldsymbol{u}^{:l}$ is contained in $W^{\boldsymbol{\sigma}}$ but $\boldsymbol{u}^{:l+1}$ is not. Let $\boldsymbol{\sigma}' = \boldsymbol{u}^{:k}(\boldsymbol{\sigma})$. The agents agree that their projection of u^{k+1} is feasible $\forall i \in I,\ \boldsymbol{\sigma}_i \in \Sigma_{u^{k+1},i}$. (The identity action ϵ is always feasible.) But $\boldsymbol{\sigma}' \notin \Sigma_{u^{k+1}}$. This contradicts Assumption 4.4. The equality relation $\left(\bigcap_{j\in I} \Pi_j^{-1}(W_j^{\boldsymbol{\sigma}})\right) = W^{\boldsymbol{\sigma}}$ is then a direct consequence of the previously established fact that for all agents $j \in I$ the set $\Pi_j^{-1}(W_j^{\boldsymbol{\sigma}})$ is a superset of W^{σ}

□

Proposition 4.7 summarizes the first important result for distributed planning from this section. All feasible global sequences are projected to a feasible sequence from the standpoint of the individual agents. On the other hand, an individual agent could assume a sequence $\boldsymbol{w}_i$ to be valid that has no feasible global counterpart. If the agents however jointly agree on a sequence by each considering the projection to their scope, then the sequence is globally feasible.

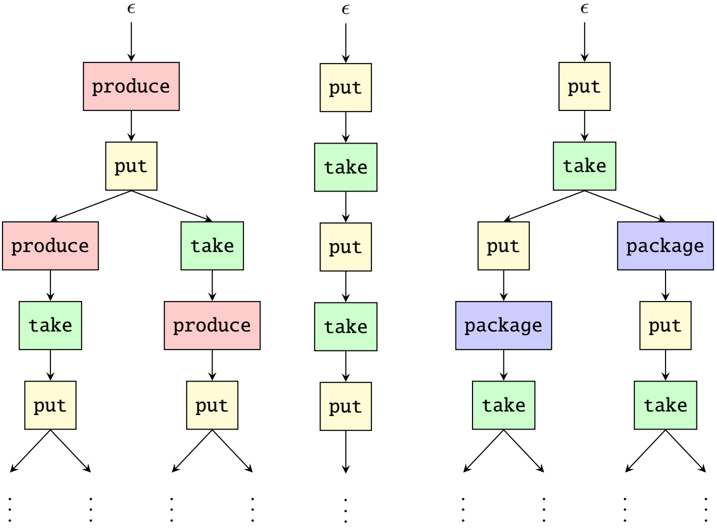
4.3 Distributed Planning for Deterministic Action Sequences

The sequence tree shared between two agents i and $j \in N(i)$ is written as W_{ij}. It contains all sequences of the joint actions $W_{ij} = (A_i \cap A_j)^*$. Note that the sequences in W_{ij} may be unfeasible. They are partial sequences and each agent has to "fill the holes" for the components in his scope. Based on a (partial) sequence $\boldsymbol{w}_{ij} \in W_{ij}$, the sequence tree of the individual agent i can be conditioned to contain only sequences that are in accordance with the shared sequence $\boldsymbol{w}_{ij}$.

Definition 4.8. *For an action sequence $\boldsymbol{v}_{ij} \in W_{ij}$ shared by the agents i and $j \in N(i)$, the conditional tree $W_i^{\boldsymbol{\sigma}}|\boldsymbol{v}_{ij}$ contains only those sequences for agent i whose projection to W_{ij} is compatible with $\boldsymbol{v}_{ij}$ in the following sense.*

$$W_i^{\boldsymbol{\sigma}}|\boldsymbol{v}_{ij} = \{\boldsymbol{w}_i \in W_i^{\boldsymbol{\sigma}} : \boldsymbol{w}_{ij} = \boldsymbol{v}_{ij}\}$$

Example 4.2. Consider the two agents from Example 4.1 and an initial state $\boldsymbol{\sigma}$ where no component contains products.



(a) $W^{\boldsymbol{\sigma}}_{\texttt{prod}}|\boldsymbol{v}_{\texttt{prod,pack}}$ **(b)** $\boldsymbol{v}_{\texttt{prod,pack}}$ **(c)** $W^{\boldsymbol{\sigma}}_{\texttt{pack}}|\boldsymbol{v}_{\texttt{prod,pack}}$

Figure 4.3: Conditioned sequence trees of two agents.

Every agent internally considers the sequence tree $W^{\boldsymbol{\sigma}}_{\texttt{prod}}$ and $W^{\boldsymbol{\sigma}}_{\texttt{pack}}$ respectively. By imposing the shared sequence $\boldsymbol{v}_{\texttt{prod,pack}}$, the agents sequence trees are pruned to the conditional sequence trees shown in Figure 4.3b, Also compare with the sequence tree for the overall scenario from Figure 2.3.

For a given initial state $\boldsymbol{\sigma}$, the global reward generated from a history $\boldsymbol{w}$ is

$$\mathfrak{r}(\boldsymbol{\sigma}, \boldsymbol{w}) = \sum_{k=1}^{|\boldsymbol{w}|} \mathfrak{r}\big(\boldsymbol{w}^{:k-1}(\boldsymbol{\sigma}), w^k, \boldsymbol{w}^{:k}(\boldsymbol{\sigma})\big)\,.$$

The local reward for the agents $i \in I$ (who know the initial state of the components in their scope $\boldsymbol{\sigma}_i$) is

$$\mathfrak{r}_i(\boldsymbol{\sigma}_i, \boldsymbol{w}_i) = \sum_{k=1}^{|\boldsymbol{w}|} \mathfrak{r}_i\big(\boldsymbol{w}_i^{:k-1}(\boldsymbol{\sigma}_i), w_i^k, \boldsymbol{w}_i^{:k}(\boldsymbol{\sigma}_i)\big)\,.$$

Now, we can state the planning problem for action sequences as a factorized optimization problem with factors $\mathfrak{r}_i$ and overlapping factor domains W_i.

Proposition 4.9. *If the reward generated by the actions a is factorized into per-agent reward as $\mathfrak{r}(\boldsymbol{\sigma}, a, \boldsymbol{\sigma}') = \sum_{i\in I} \mathfrak{r}_i(\boldsymbol{\sigma}_i, a_i, \boldsymbol{\sigma}'_i)$, then the reward $\mathfrak{r}(\boldsymbol{w})$ for a global action sequence $\boldsymbol{w}$ factorizes to the sum of the per-agent reward functions $\mathfrak{r}_i : W_i \to \mathbb{R}$.*

$$\mathfrak{r}(\boldsymbol{\sigma}, \boldsymbol{w}) = \sum_{i\in I} \mathfrak{r}_i(\boldsymbol{\sigma}_i, \boldsymbol{w}_i)$$

Proof. Take the following sequence of equations. The gist of the proof lies in the equality between the Equations 4.7 and 4.8.

$$\begin{aligned}
\mathfrak{r}(\boldsymbol{\sigma}, \boldsymbol{w}) &= \sum_{k=1}^{|\boldsymbol{w}|} \mathfrak{r}\big(\boldsymbol{w}^{:k-1}(\boldsymbol{\sigma}), w^k, \boldsymbol{w}^{:k}(\boldsymbol{\sigma})\big) && (4.7)\\
&= \sum_{k=1}^{|\boldsymbol{w}|} \sum_{i\in I} \mathfrak{r}_i\big(\boldsymbol{w}_i^{:k-1}(\boldsymbol{\sigma}_i), w_i^k, \boldsymbol{w}_i^{:k}(\boldsymbol{\sigma}_i)\big) && (4.8)\\
&= \sum_{i\in I} \sum_{k=1}^{|\boldsymbol{w}|} \mathfrak{r}_i\big(\boldsymbol{w}_i^{:k-1}(\boldsymbol{\sigma}_i), w_i^k, \boldsymbol{w}_i^{:k}(\boldsymbol{\sigma}_i)\big) = \sum_{i\in I} \mathfrak{r}_i(\boldsymbol{\sigma}_i, \boldsymbol{w}_i)
\end{aligned}$$

First, Proposition 4.7 guarantees that for any system state $\boldsymbol{\sigma}$ and feasible sequence $\boldsymbol{w}$ the projected sequence $\boldsymbol{w}_i$ is feasible for the projected system state $\boldsymbol{w}_i$. Secondly, from Assumption 4.5 follows that knowledge of $\boldsymbol{w}_i$ suffices to determine the untimed state of the components C_i. Third, the agents reward depends only on the untimed state of the components in their scope according to Assumption 4.6. Therefore the reward of the individual agents is determined by just the components in their scope and the action sequence projected to their scope. □

Definition 4.10. *The V-value of the agent i with agent-state $\boldsymbol{\sigma}_i$ is the sum of rewards generated by the best feasible action sequence from the perspective of the agent.*

$$\mathfrak{v}_i(\boldsymbol{\sigma}_i) = \max_{\boldsymbol{w}_i \in W_i^{\boldsymbol{\sigma}}} \mathfrak{r}_i(\boldsymbol{\sigma}_i, \boldsymbol{w}_i)$$

We are not discounting later reward to compute the V-value. Instead it is implied that the tree W either has a maximum height. Either because the scenario is “done” after a finite number of actions or based on a cutoff height of the sequence tree.

Definition 4.11. *The conditional V-value of an agent i for the action sequence $\boldsymbol{v}_{ij}$ shared with the neighbor $j \in N(i)$ is the best reward the agent can achieve with a sequence that projects to $\boldsymbol{v}_{ij}$.*

$$\mathfrak{v}_i(\boldsymbol{\sigma}_i \mid \boldsymbol{v}_{ij}) = \begin{cases} \max\limits_{\boldsymbol{w}_i \in W_i^{\boldsymbol{\sigma}} | \boldsymbol{v}_{ij}} \mathfrak{r}_i(\boldsymbol{\sigma}_i, \boldsymbol{w}_i), & W_i^{\boldsymbol{\sigma}} | \boldsymbol{v}_{ij} \neq \varnothing \\ -\infty, & \textit{else} \end{cases}$$

The local V-value is defined as $-\infty$ if the shared sequence $\boldsymbol{v}_{ij}$ is infeasible for the initial state of the components in the agent's scope $\boldsymbol{\sigma}_i$. This becomes important later on when $\mathfrak{v}_i$ is used by the agents to signal preferences for shared sequences to their neighbors.

Proposition 4.12. *In a setting with just two agents i and j, the global V-value can be decomposed as follows.*

$$\mathfrak{v}(\boldsymbol{\sigma}) = \max_{\boldsymbol{w} \in W^{\boldsymbol{\sigma}}} \mathfrak{r}(\boldsymbol{\sigma}, \boldsymbol{w}) = \max_{\boldsymbol{w}_{ij} \in W_{ij}} \Big[\mathfrak{v}_i(\boldsymbol{\sigma}_i \mid \boldsymbol{w}_{ij}) + \mathfrak{v}_j(\boldsymbol{\sigma}_j \mid \boldsymbol{w}_{ij}) \Big]$$

Proof. Let $\boldsymbol{u} \in W$ an optimal global action sequence, so the reward generated by $\boldsymbol{u}$ is $\mathfrak{r}(\boldsymbol{\sigma}, \boldsymbol{u}) = \mathfrak{v}(\boldsymbol{\sigma})$. (There may be several optimal action sequences.) This reward decomposes into the private reward of the agents i and j.

$$\mathfrak{r}(\boldsymbol{\sigma}, \boldsymbol{u}) = \mathfrak{r}_i(\boldsymbol{\sigma}_i, \boldsymbol{u}_i) + \mathfrak{r}_j(\boldsymbol{\sigma}_j, \boldsymbol{u}_j)$$

Given the shared sequence $\boldsymbol{u}_{ij}$, agent j can choose any action sequence from $W_j^{\boldsymbol{\sigma}} | \boldsymbol{u}_{ij}$ without impacting the reward for i. We know that $\mathfrak{r}_j(\boldsymbol{\sigma}_j, \boldsymbol{u}_j) = \max_{\boldsymbol{v}_i \in W_j^{\boldsymbol{\sigma}} | \boldsymbol{u}_{ij}} \mathfrak{r}_j(\boldsymbol{\sigma}_j, \boldsymbol{v}_j)$. Otherwise, if the agent j could find a better sequence than $\boldsymbol{u}_i$, $\boldsymbol{u}$ could not be a maximizer for the global sequence. Since $\boldsymbol{u}_j \in W_j^{\boldsymbol{\sigma}} | \boldsymbol{u}_{ij}$ we know that $W_j^{\boldsymbol{\sigma}} | \boldsymbol{u}_{ij}$ is nonempty and therefore with Definition 4.11

$$\mathfrak{r}(\boldsymbol{\sigma}, \boldsymbol{u}) = \mathfrak{r}_i(\boldsymbol{\sigma}_i, \boldsymbol{u}_i) + \mathfrak{v}_j(\boldsymbol{\sigma}_j \mid \boldsymbol{u}_{ij}) .$$

The same line of reasoning can be followed for the agent i so that $\mathfrak{r}(\boldsymbol{\sigma}, \boldsymbol{u}) = \mathfrak{v}_i(\boldsymbol{\sigma}_i, \boldsymbol{u}_{ij}) + \mathfrak{v}_j(\boldsymbol{\sigma}_j \mid \boldsymbol{u}_{ij})$. □

In the case with two agents, given precomputed conditional V-value functions $\mathfrak{v}_i(\boldsymbol{\sigma} \mid \boldsymbol{w}_{ij})$, (e.g. available as a lookup table), the optimization problem to find the optimal reward for a given system state $\boldsymbol{\sigma}$ is simplified from a search over $W^{\boldsymbol{\sigma}}$ to a search over just W_{ij}. Now assume a case with three agents i, j, l. The agents i and j have shared components and the agents j and l have shared components. But i and l do not share any components. Conditioned on the action sequence $\boldsymbol{w}_{jl}$, the agent l is not only independent from the actions of j but also of the actions of i. This mechanism is used for “utility propagation” between agents that form a tree-graph. The difference to the original GDL is that the overall domain W is not a cartesian product of the W_i.

Assumption 4.13. *The agents and their neighbor relations between the agents form a tree-graph. The components in the scope of agent i are in the shared scope with at most one of i’s neighboring agents.*

$$\forall j \in N(i),\ \forall c \in C_{ij} \Rightarrow \forall l \in N(i) \setminus \{j\},\ c \notin C_{il}$$

For a given initial system state $\boldsymbol{\sigma}$, the messages $\mathfrak{m}_{i \to j} : W_{ij} \to \mathbb{R}$ exchanged between neighboring agents are computed as follows:

$$\mathfrak{m}_{i \to j}(\boldsymbol{v}_{ij}) = \max_{\boldsymbol{w}_i \in W_i^{\boldsymbol{\sigma}} | \boldsymbol{v}_{ij}} \Big[\mathfrak{v}_i(\boldsymbol{\sigma}_i, \boldsymbol{w}_i) + \sum_{l \in N(i) \setminus \{l\}} \mathfrak{m}_{l \to i}(\boldsymbol{w}_{il}) \Big] \tag{4.9}$$

The messages $\mathfrak{m}_{i \to j} : W_{ij} \to \mathbb{R}$ describe the best reward that the subgraph of agents "behind" the edge $i \to j$ can achieve for a given shared sequence $\boldsymbol{w}_{ij}$. Computing the messages $\mathfrak{m}$ quickly becomes intractable. The number of possible shared sequences grows exponentially with the length of the shared sequence. Furthermore, the optimization performed for each messages $\mathfrak{m}_{i \to j}$ and shared sequence $\boldsymbol{w}_{ij}$ requires in itself an optimization of over $W_i^{\boldsymbol{\sigma}} | \boldsymbol{v}_{ij}$ that takes the messages received by the other neighbours $N(i) \setminus j$ into account. The conditional tree $W_i^{\boldsymbol{\sigma}} | \boldsymbol{v}_{ij}$ also grows exponentially with the tree depth. (Although pruning non-conforming sequences can drastically reduce the overall size).

The Max-Plus algorithm from the GDL family is used to efficiently solve the maximum a-posteriori (MAP) problem of finding the event with highest probability. It has been used for "utility propagation" for multi-agent decision making by [KV05]. To overcome the combinatorial explosion of the search space, we develop a novel combination of Max-Plus with MCTS. See Algorithm 14 for the full algorithm specification. Similar to standard UCT, the Distributed Upper Confidence on Trees (DUCT) algorithm performs iterative playouts and generates statistics that guide decision-making in later playouts. Every agent $i \in I$ stores the reward he can make after a sequence $\boldsymbol{w}_i$ in a hashmap $v_i[\boldsymbol{w}_i]$. As before, the hashmap returns zero when no entry has been set prior. Every agent is performing independent playouts based on his internal simulator. Actions are selected according to the UCT rule with the addition that conditional reward signaled by they neighbors is considered as well. For the updates, the computed V-value estimate for a sequence $\boldsymbol{w}_i$ considers only the reward the agent receives. But the action a_i^* are selected to maximize the reward for all agents. Afterwards, the messages to the neighboring agents are computed. Here the action a_i^* is implied by the use of the local V-value v_i. Note that the reward signaled by the agent j to i is not mirrored back in the messages from i to j. Here, the agents are jointly exploring the scenario tree. Initially, all messages are set to zero. Therefore, if rewards are generally negative, then the agents will initially overestimate the value of sequences where no empirical reward estimate from the neighbors exist.

Algorithm 14 Distributed Upper Confidence on Trees (DUCT) Algorithm

```
procedure DUCT($\boldsymbol{\sigma}^0$)
    $m_{i\to j}[\,\cdot\,] \leftarrow 0 \quad \forall i \in I, j \in N(i)$
    $n_i[\,\cdot\,] \leftarrow 0,\ v_i[\,\cdot\,] \leftarrow 0 \quad \forall i \in I$
    while enough time do
        for $i \in I$ do
            $(\boldsymbol{w}_i, \boldsymbol{r}_i) \leftarrow \textsc{Play}_i(\boldsymbol{\sigma}_i^0)$
            $\textsc{Update}_i(\boldsymbol{w}_i, \boldsymbol{r}_i)$
    return $\arg\max_{a \in A} \Big[ \sum_{\substack{i \in I:\\ a_i \in A_i}} v_i[a_i] \Big]$

procedure $\textsc{Play}_i(\boldsymbol{\sigma}_i)$
    $\boldsymbol{w}_i \leftarrow \varepsilon,\ \boldsymbol{r}_i \leftarrow \varepsilon$
    while $\neg\textsc{done}(\boldsymbol{\sigma}_i)$ do
        $B \leftarrow \{b_i \in A_i : n[\boldsymbol{w}_i b_i] = 0\}$
        if $B \neq \varnothing$ then
            $a_i \leftarrow \pi_i^B(\boldsymbol{\sigma}_i)$
        else
            $a_i \leftarrow \underset{b_i \in A_i}{\arg\max} \Big[ v_i[\boldsymbol{w}_i b_i] + \alpha \sqrt{\frac{\log n[\boldsymbol{w}_i]+1}{n[\boldsymbol{w}_i b_i]}} +$
                $\sum_{j \in N(i)} m_{j\to i}(\Pi_{ij}(\boldsymbol{w}_i b_i)) \Big]$
        $(\boldsymbol{\sigma}_i, e_i) \leftarrow a_i(\boldsymbol{\sigma}_i)$
        $\boldsymbol{w}_i \leftarrow \boldsymbol{w}_i a_i,\ \boldsymbol{r}_i \leftarrow (r_i^1, r_i^2, \ldots, e_i)$
    return $\boldsymbol{w}_i, \boldsymbol{r}_i$

procedure $\textsc{Update}_i(\boldsymbol{w}_i, \boldsymbol{r}_i)$
    for $k = |\boldsymbol{w}_i|, \ldots, 1$ do
        $\boldsymbol{u}_i \leftarrow \boldsymbol{w}_i^{:k}$
        $n[\boldsymbol{u}_i] \leftarrow n[\boldsymbol{u}_i] + 1$
        $a_i^* \leftarrow \underset{a_i \in A_i : n[\boldsymbol{u}_i a_i] > 0}{\arg\max} \Big[ q[\boldsymbol{u}_i a_i] + \sum_{j \in N(i)} m_{j\to i}[\Pi_{ij}(\boldsymbol{u}_i a_i)] \Big]$
        $v_i[\boldsymbol{u}_i] \leftarrow r_i^k + v_i[\boldsymbol{u}_i a_i^*]$
        for $j \in N(i)$ do
            $u \leftarrow v_i[\boldsymbol{u}_i] + \sum_{l \in N(i) \setminus \{l\}} m_{l\to i}(\Pi_{il}(\boldsymbol{u}_i))$
            if $e > m[\Pi_{ij}(\boldsymbol{u}_i)]$ then
                $m[\Pi_{ij}(\boldsymbol{u}_i)] \leftarrow e$
```

4.4 Distributed Planning under Uncertainty

In stochastic scenarios, following the description from Section 2.3, the current state is not known with absolute certainty. Instead, the system state can only be inferred from indirect observations. Recall that histories $\boldsymbol{h}$ are comprised of episodes $h^k = (a^k \theta^k \boldsymbol{o}^k)$ with an action, action-parameters and observations. The set of all possible global histories of finite length is $H = (A \times \Theta \times O)^*$. The set of histories H implies a tree-graph with edges between observations and actions, actions and parameters, and parameters and observations if they can occur in sequence in a history. From every history, the current system state can be inferred $\mathbb{P}(\boldsymbol{\sigma} \mid \boldsymbol{h})$. For this, an belief distribution for the initial state $\underline{\boldsymbol{\sigma}}^0$ is updated with the received observations. We now make the additional distinction between histories and *complete histories*. Complete histories are the leafs in the tree-graph of H. They denote histories after which the scenario is "done". This can also be enforced by a maximum history depth. The set of complete histories is $\overline{H} \subseteq H$.

The algorithm enhances our prior work in [Pfr16a] in several regards. Most importantly, every agent participating in the decision making has a local simulator to predict the evolution of the system state based on his restricted local knowledge. An important inspiration came from [AO15]. However, they use Variable Elimination [KF09] for selecting joint actions instead of message passing. In addition, they rely on a central simulator for the global system to generate sample plays. Since MCTS is an online algorithm, they would require a simulator for the global system also at runtime for the agent coordination.

As the agents $i \in I$ have a limited scope, they can only observe a portion of the full history. In particular, the projection to the agent-scope $\Pi_i(\boldsymbol{h}) = \boldsymbol{h}_i$ contains only the episodes with actions $a \in A_i$. The observed action parameters are from the full parameter space of the action $\theta \in \Theta_a$. The observations received by the agent i are from components that participate in action a and are in the scope of the agent $\boldsymbol{o}_i \in O_{a,i} = (\times_{c \in C_{a,i}} O_c)$.

Definition 4.14. *A history $\boldsymbol{h} \in H$ projects to the scope of an agent $i \in I$ as*

$$\boldsymbol{h}_i = \underbrace{a_i^1 \theta_i^1 \boldsymbol{o}_i^1}_{h_i^1} \dots \underbrace{a_i^{|\boldsymbol{h}|} \theta_i^{|\boldsymbol{h}|} \boldsymbol{o}_i^{|\boldsymbol{h}|}}_{h_i^{|\boldsymbol{h}|}} \ .$$

The set of histories for agent i is $H_i = (A_i \times \Theta_i \times O_i)^*$ where $\Theta_i = (\cup_{a \in A_i} \Theta_a)$ and $O_i = (\cup_{a \in A_i} O_{a,i})$.

Similar to the projection of deterministic action sequences from Section 4.3, the episodes are indexed with k. If an episode of the global sequence refers to an action outside of agent i's scope, then the action projects to the identity operator $a \notin A_i \Rightarrow a_i = \varepsilon$ and the episode does not occur in the agent's local history. The global history thus may contain more episodes than are visible to the individual agent. The difference in the index k will be made explicit only when the meaning is not clear from context.

Agents have a local decision-making policy π_i. Since the components in the agents scope are overlapping, neighboring agents need to coordinate to select the next action in their shared scope. Disagreement would lead to incompatible action selections for the components in the shared scope. This needs to be avoided. So the local decision-making policy of an agent i is no longer a deterministic result from the local history $\boldsymbol{h}_i$ alone. It also depends on the coordination with neighboring agents – and hence on the observations the agents $j \in N(i)$ have made that are not necessarily in the scope of i. This lack of information from the limited viewpoint of agent i is expressed by taking the policy as a random variable as well. The next action and action parameters are sampled as $(a_i, \theta_i) \sim \underline{\pi}_i(\boldsymbol{h}_i)$.

The value for optimal decision making in each node (from the viewpoint of agent i), the V-value and the Q-value, can be computed recursively with Bellman's Equation [Put94]. Note that the decision-making step is split into action-selection and parameter-selection. To simplify the notation, we refer to both the V-value and the Q-value of an agent history as q.

Definition 4.15. *The Q-value of selfish agent i assumes optimal decision making by i. The other agents $I \setminus \{i\}$ coordinate with i by agreeing to i's decisions for the components in the shared scope.*

$$\mathsf{q}_i(\boldsymbol{h}_i) = \begin{cases} 0, & \boldsymbol{h}_i \in \overline{H}_i \\ \max\limits_{a_i \in A_i} \mathsf{q}_i(\boldsymbol{h}_i a_i), & \textit{else} \end{cases} \tag{4.10}$$

$$\mathsf{q}_i(\boldsymbol{h}_i a_i) = \max_{\theta_i \in \Theta_{a,i}} \mathsf{q}_i(\boldsymbol{h}_i a_i \theta_i) \tag{4.11}$$

$$\mathfrak{q}_i(\boldsymbol{h}_i a_i \theta_i) = \mathop{\mathbb{E}}_{\substack{\boldsymbol{\sigma}_i, \boldsymbol{\sigma}'_i \in \Sigma_i, \\ \boldsymbol{o}_i \in O_{a,i}}} \Big[\mathfrak{r}_i(\boldsymbol{\sigma}_i, a_i, \theta_i, \boldsymbol{\sigma}'_i) + \mathfrak{q}_i(\boldsymbol{h}_i a_i \theta_i \boldsymbol{o}_i) \,\Big|\, \boldsymbol{h}_i a_i \theta_i \Big] \tag{4.12}$$

A distinction by cases is made whether the last episode contains only an action, an action with action parameters, or an action with parameters and the resulting observations. When action, parameter and observation are appended to a history, as in $\boldsymbol{h}'_i = \boldsymbol{h}_i a_i \theta_i \boldsymbol{o}_i$, then $\boldsymbol{h}'_i$ matches with Equation 4.10 that is defined for histories with complete episodes. Equations 4.11 and 4.12 take histories with an incomplete last episode as input. The case distinction in Equation 4.10 is required so that the recursive formulation terminates at the leaf nodes of H_i.

A shared history $\boldsymbol{h}_{ij} \in H_{ij} = (A_{ij} \times \Theta_{ij} \times O_{ij})^*$ between an agent i and his neighbor $j \in N(i)$ contains only the episodes with a shared action $a \in A_{ij}$. The agents i and j both observe the complete action parameters for the shared actions. The observations in the shared history contain only the observations from components in the shared scope $O_{ij} = (\cup_{a \in A_{ij}} (\times_{c \in C_a \cap C_{ij}} O_c))$. The V-value and Q-value are now conditioned on a shared history for the future episodes. Observations that are not compatible with the shared history are marginalized out in the probabilistic expectation. Incompatible action and parameter choices are disallowed in the maximization steps.

Definition 4.16. *A conditional Q-value for a selfish agent i assumes optimal decision making under the constraint that future episodes (after the initial history $\boldsymbol{h}_i$) project to the partial history $\boldsymbol{g}_{ij}$ shared with the neighbor $j \in N(i)$.*

$$\mathfrak{q}_i(\boldsymbol{h}_i \,|\, \boldsymbol{g}_{ij}) = \begin{cases} 0, & \boldsymbol{h}_i \in \overline{H}_i \\ \displaystyle\max_{a_i \in (A_i \setminus A_{ij}) \cup \{a^1_{ij}\}} \mathfrak{q}_i(\boldsymbol{h}_i a_i \,|\, \boldsymbol{g}_{ij}), & \textit{else} \end{cases} \tag{4.13}$$

$$\mathfrak{q}_i(\boldsymbol{h}_i a_i \,|\, \boldsymbol{g}_{ij}) = \begin{cases} \mathfrak{q}_i(\boldsymbol{h}_i a_i \theta^1_{ij} \,|\, \boldsymbol{g}_{ij}), & a_i = a^1_{ij} \\ \displaystyle\max_{\theta_i \in \Theta_{a,i}} \mathfrak{q}_i(\boldsymbol{h}_i a_i \theta_i \,|\, \boldsymbol{g}_{ij}), & \textit{else} \end{cases} \tag{4.14}$$

$$\mathfrak{q}_i(\boldsymbol{h}_i a_i \theta_i \,|\, \boldsymbol{g}_{ij}) = \begin{cases} \underset{\substack{\boldsymbol{\sigma}_i,\boldsymbol{\sigma}'_i \in \Sigma_i, \\ \boldsymbol{u}_i \in O_{a,i}, \\ \boldsymbol{u}_{ij} = \boldsymbol{o}^1_{ij}}}{\mathbb{E}} \Big[\mathfrak{r}_i(\boldsymbol{\sigma}_i, a_i, \theta_i, \boldsymbol{\sigma}'_i) + \\ \qquad \mathfrak{q}_i(\boldsymbol{h}_i a_i \theta_i \boldsymbol{u}_i \,|\, \boldsymbol{g}^{2:}_{ij}) \,\Big|\, \boldsymbol{h}_i a_i \theta_i\Big], & a_i = a^1_{ij} \\ \underset{\substack{\boldsymbol{\sigma}_i,\boldsymbol{\sigma}'_i \in \Sigma_i, \\ \boldsymbol{o}_i \in O_{a,i}}}{\mathbb{E}} \Big[\mathfrak{r}_i(\boldsymbol{\sigma}_i, a_i, \theta_i, \boldsymbol{\sigma}'_i) + \\ \qquad \mathfrak{q}_i(\boldsymbol{h}_i a_i \theta_i \boldsymbol{o}_i \,|\, \boldsymbol{g}_{ij}) \,\Big|\, \boldsymbol{h}_i a_i \theta_i\Big], & \textit{else} \end{cases} \tag{4.15}$$

The action, action parameters and observations of the first episode of the partial history $\boldsymbol{g}_{ij}$ are referred to as a^1_{ij}, θ^1_{ij} and $\boldsymbol{o}^1_{ij}$ respectively. As episodes are added to per-agent history $\boldsymbol{h}_i$, the remaining shared history for future episodes is getting shorter. Once no shared history for the conditioning remains with $\boldsymbol{g}_{ij} = \varepsilon$, the Q-value and V-value fall back to the formulations from Equations 4.10 to 4.12. The action choice is constrained to agree with the remaining partial history $\boldsymbol{g}_{ij}$. Actions without participating components from C_{ij} can be freely chosen as they are not constrained by the partial history. The case distinction in Equation 4.14 requires that the matching parameters from the partial history $\boldsymbol{g}_{ij}$ are taken if the action is from the partial history. Finally, Equation 4.15 also makes a case distinction for actions from the constraining partial history $\boldsymbol{g}_{ij}$. Equation 4.15 then recurses by adding the expected reward from future episodes constrained on the remaining partial history. If the current action is from the shared history, then the remaining shared history $\boldsymbol{g}^{:2}_{ij}$ has the current episode removed.

The agents in Definition 4.16 are self-interested. In order to have collaborative agents jointly optimize the global reward (across all agents), each individual agents has to assess the impact of his choices on the expected future reward for himself as well as for the other agents. Analogous to the previous section on distributed planning for deterministic action sequences, the agents are assumed to form a tree-graph with their neighborhood relations. (Cf. the discussion of Assumption 4.13.) To coordinate, the agents exchange messages $\mathfrak{m}_{i \to j} : H_{ij} \to \mathbb{R}$ with their neighbors $j \in N(i)$. The value of the message $\mathfrak{m}_{i \to j}(\boldsymbol{g}_{ij})$ evaluated for a given shared history $\boldsymbol{g}_{ij}$ describes the expected Q-value (the reward for optimal play in the remaining episodes) for the agents behind the edge $i \to j$ conditional to the given shared partial history. See the later Definition 4.18 for the messages. The per-agent history is projected to the shared scope with $\boldsymbol{h}_{ij} = \Pi_{il}(\boldsymbol{h}_i)$. The expected immediate reward in the subtree behind the edge $l \to i$

$$\bar{\mathfrak{r}}_{j\to i}(\boldsymbol{h}_i, a_i\theta_i\boldsymbol{o}_i) = \mathfrak{m}_{j\to i}(\boldsymbol{h}_{ij}) - \mathfrak{m}_{j\to i}(\Pi_{ij}(\boldsymbol{h}_i a_i\theta_i\boldsymbol{o}_i))$$

refers to the reward the agents in the sub-tree expect to make when they “fill the gaps” between the partial histories $\boldsymbol{h}_{ij}$ and $\Pi_{ij}(\boldsymbol{h}_i a_i\theta_i\boldsymbol{o}_i)$.

Definition 4.17. *The global Q-value for an unselfish agent i conditioned on the future partial shared history $\boldsymbol{g}_{ij}$ assumes optimal decision making by the agent i with respect to the expected global reward and a fixed policy followed by the other agents.*

$$\mathfrak{q}_i^*(\boldsymbol{h}_i \,|\, \boldsymbol{g}_{ij}) = \begin{cases} -\infty, & \boldsymbol{h}_i \in \overline{H}_i,\ \boldsymbol{g}_{ij} \neq \varepsilon \\ 0, & \boldsymbol{h}_i \in \overline{H}_i,\ \boldsymbol{g}_{ij} = \varepsilon \\ \underset{a_i \in (A_i \setminus A_{ij}) \cup \{a_{ij}^1\}}{\arg\max} \ \mathfrak{q}_i^*(\boldsymbol{h}_i a_i \,|\, \boldsymbol{g}_{ij}), & \textit{else} \end{cases} \tag{4.16}$$

$$\mathfrak{q}_i^*(\boldsymbol{h}_i a_i \,|\, \boldsymbol{g}_{ij}) = \begin{cases} \mathfrak{q}_i^*(\boldsymbol{h}_i a_i \theta_{ij}^1 \,|\, \boldsymbol{g}_{ij}), & a_i = a_{ij}^1 \\ \max\limits_{\theta_i \in \Theta_{a,i}} \mathfrak{q}_i^*(\boldsymbol{h}_i a_i \theta_i \,|\, \boldsymbol{g}_{ij}), & \textit{else} \end{cases} \tag{4.17}$$

$$\mathfrak{q}_i^*(\boldsymbol{h}_i a_i \theta_i \,|\, \boldsymbol{g}_{ij}) = \begin{cases} \underset{\substack{\boldsymbol{\sigma}_i, \boldsymbol{\sigma}_i' \in \Sigma_i, \\ \boldsymbol{u}_i \in O_{a,i}, \\ \boldsymbol{u}_{ij} = \boldsymbol{o}_{ij}^1}}{\mathbb{E}} \Big[\mathfrak{r}_i(\boldsymbol{\sigma}_i, a_i, \theta_i, \boldsymbol{\sigma}_i') + \sum\limits_{l \in N(i)} \big[\bar{\mathfrak{r}}_{l\to i}(\boldsymbol{h}_i, a_i\theta_i\boldsymbol{u}_i) \big] + \mathfrak{q}_i^*(\boldsymbol{h}_i a_i \theta_i \boldsymbol{u}_i \,|\, \boldsymbol{g}_{ij}^{2:}) \,\Big|\, \boldsymbol{h}_i a_i \theta_i \Big], & a_i = a_{ij}^1 \\ \underset{\substack{\boldsymbol{\sigma}_i, \boldsymbol{\sigma}_i' \in \Sigma_i, \\ \boldsymbol{o}_i \in O_{a,i}}}{\mathbb{E}} \Big[\mathfrak{r}_i(\boldsymbol{\sigma}_i, a_i, \theta_i, \boldsymbol{\sigma}_i') + \sum\limits_{l \in N(i)} \big[\bar{\mathfrak{r}}_{l\to i}(\boldsymbol{h}_i, a_i\theta_i\boldsymbol{o}_i) \big] + \mathfrak{q}_i^*(\boldsymbol{h}_i a_i \theta_i \boldsymbol{o}_i \,|\, \boldsymbol{g}_{ij}) \,\Big|\, \boldsymbol{h}_i a_i \theta_i \Big], & \textit{else} \end{cases} \tag{4.18}$$

Again, Equation 4.16 can return negative infinity in the case where the scenario is “done” but the constraint to fulfill the partial history $\boldsymbol{g}_{ij}$ has not been fulfilled. The use of messages $\mathfrak{m}_{l\to i}(\boldsymbol{h}_{il})$ relies on Assumption 4.13. So at most two agents share a component in their scope. Otherwise for a given future shared history $\boldsymbol{g}_{ij}$, the expected reward for the agents in the sub-tree behind the edge $l \to i$ could also be conditional to a portion of the shared history $\boldsymbol{g}_{ij}$ that also applies to l, i.e. $\Pi_{il}(\boldsymbol{g}_{ij})$. The messages $\mathfrak{m}$ would then be conditioned to this as $\mathfrak{m}_{l\to i}(\Pi_{il}(\boldsymbol{h}_i) \,|\, \Pi_{il}(\boldsymbol{g}_{ij}))$. Assumption 4.13 removes this source of further complexity.

Now a word on the difference between the global Q-value $\mathfrak{q}_i^*$ estimated by the agents i and the messages $\mathfrak{m}_{i\to j}$ between neighboring agents i and j. In accordance with the principles of the GDL described in Section 4.1, it has to be avoided that the agents "mirror back" expected reward that was signaled to them by a neighbor j. Only the expected reward from the other neighbors $N(i) \setminus \{j\}$ is forwarded in the messages. In essence the message contains the expected reward generated in the subtree (according to the agent's neighbor relation) behind the edge $i \to j$ under the assumption of globally optimal decision making by the agents according to their respective Q-value estimation.

Definition 4.18. *The messages exchanged over the edge $i \to j$ between neighboring agents are computed for optimal decisions based on $\mathfrak{q}_i^*$.*

$$\mathfrak{q}_{i\to j}(\boldsymbol{h}_i \,|\, \boldsymbol{g}_{ij}) = \begin{cases} -\infty, & \boldsymbol{h}_i \in \overline{H}_i,\ \boldsymbol{g}_{ij} \neq \varepsilon \\ 0, & \boldsymbol{h}_i \in \overline{H}_i,\ \boldsymbol{g}_{ij} = \varepsilon \\ \mathfrak{q}_{i\to j}(\boldsymbol{h}_i a_i^* \,|\, \boldsymbol{g}_{ij}), & \textit{else} \end{cases} \tag{4.19}$$

where $a_i^* = \operatorname*{arg\,max}_{a_i \in (A_i \setminus A_{ij}) \cup \{a_{ij}^1\}} \mathfrak{q}_i^*(\boldsymbol{h}_i a_i \,|\, \boldsymbol{g}_{ij})$,

$$\mathfrak{q}_{i\to j}(\boldsymbol{h}_i a_i \,|\, \boldsymbol{g}_{ij}) = \begin{cases} \mathfrak{q}_{i\to j}(\boldsymbol{h}_i a_i \theta_{ij}^1 \,|\, \boldsymbol{g}_{ij}), & a_i = a_{ij}^1 \\ \mathfrak{q}_{i\to j}(\boldsymbol{h}_i a_i \theta_i^* \,|\, \boldsymbol{g}_{ij}), & \textit{else} \end{cases} \tag{4.20}$$

where $\theta_i^* = \operatorname*{arg\,max}_{\theta_i \in \Theta_{a,i}} \mathfrak{q}_i^*(\boldsymbol{h}_i a_i \theta_i \,|\, \boldsymbol{g}_{ij})$,

$$\mathfrak{q}_{i\to j}(\boldsymbol{h}_i a_i \theta_i \,|\, \boldsymbol{g}_{ij}) = \begin{cases} \underset{\substack{\boldsymbol{\sigma}_i,\boldsymbol{\sigma}_i' \in \Sigma_i,\\ \boldsymbol{u}_i \in O_{a,i},\\ \boldsymbol{u}_{ij}=\boldsymbol{o}_{ij}^1}}{\mathbb{E}} \Big[\mathfrak{r}_i(\boldsymbol{\sigma}_i, a_i, \theta_i, \boldsymbol{\sigma}_i') + \sum_{l \in N(i)\setminus\{j\}} \big[\bar{\mathfrak{r}}_{l\to i}(\boldsymbol{h}_i, a_i\theta_i\boldsymbol{u}_i)\big] + \mathfrak{q}_{i\to j}(\boldsymbol{h}_i a_i \theta_i \boldsymbol{u}_i \,|\, \boldsymbol{g}_{ij}^{2:}) \,\Big|\, \boldsymbol{h}_i a_i \theta_i \Big], & a_i = a_{ij}^1 \\ \underset{\substack{\boldsymbol{\sigma}_i,\boldsymbol{\sigma}_i' \in \Sigma_i,\\ \boldsymbol{o}_i \in O_{a,i}}}{\mathbb{E}} \Big[\mathfrak{r}_i(\boldsymbol{\sigma}_i, a_i, \theta_i, \boldsymbol{\sigma}_i') + \sum_{l \in N(i)\setminus\{j\}} \big[\bar{\mathfrak{r}}_{l\to i}(\boldsymbol{h}_i, a_i\theta_i\boldsymbol{o}_i)\big] + \mathfrak{q}_{i\to j}(\boldsymbol{h}_i a_i \theta_i \boldsymbol{o}_i \,|\, \boldsymbol{g}_{ij}) \,\Big|\, \boldsymbol{h}_i a_i \theta_i \Big], & \textit{else} \end{cases} \tag{4.21}$$

Finally the message from i *to the neighbor* j *is*

$$\mathfrak{m}_{i\to j}(\boldsymbol{g}_{ij}) = \mathfrak{q}_{i\to j}(\varepsilon \,|\, \boldsymbol{g}_{ij}) \,. \tag{4.22}$$

Decision-making by the agent i depends on optimal decision-making by his neighbors, as communicated in the messages $\mathfrak{m}_{j\to i}$ and vice versa. Astute readers will have noticed a circular dependency between Definition 4.17 and Definition 4.18. Taken together the values are well-defined. Suppose that H is only one level deep. Then the messages $\mathfrak{m}$ can be computed with a forward-backward pass similar to the standard Max-Plus algorithm. Now let H allow two actions in a row. With the same argument, the messages that evaluate the second action choice can be generated. Once these messages are known, the agents can compute the Q-value for the possible first actions. By induction, the message values are well defined on H where all sequences have finite length.

Example 4.3. Suppose that an Original Equipment Manufacturer (OEM) owns two production sites. One in Germany and the other in China. A customer buys a lot of 1000 items to be made to order. The German site can produce the items for \$40 a piece. The Chinese site can fulfill the order for \$35 a piece. But due to uncertainties for the long transport, there is a 10% chance that the products will not reach the OEM headquarters in time for packaging and delivery. For this, the scenario defines three components $C = \{\texttt{oem}, \texttt{de}, \texttt{cn}\}$. Every component also represents an agent $C = I$. There are six actions defined: $A = \{\texttt{pass\_de}, \texttt{prod\_de}, \texttt{prod\_cn}, \texttt{pass\_cn}, \texttt{pass\_oem}, \texttt{deliver}\}$. The "pass" action terminates the scenario for the respective agent. The participants of the action $C_{\texttt{pass\_de}} = \{\texttt{de}\}$, $C_{\texttt{prod\_de}} = \{\texttt{oem}, \texttt{de}\}$, $C_{\texttt{pass\_cn}} = \{\texttt{cn}\}$, $C_{\texttt{prod\_cn}} = \{\texttt{oem}, \texttt{cn}\}$, $C_{\texttt{deliver}} = C_{\texttt{pass\_oem}} = \{\texttt{oem}\}$. The shared actions are $A_{\texttt{oem},\texttt{de}} = \{\texttt{prod\_de}\}$ and $A_{\texttt{oem},\texttt{cn}} = \{\texttt{prod\_cn}\}$. The action prod_cn returns an observation that is either $o_{\texttt{f}}$ or $o_{\texttt{s}}$ indicating either failure or success. None of the actions take a parameter. The reward generated by the actions for the involved agents is stated in Table 4.1.

Action / Reward (in k$)	$\mathfrak{r}_{\texttt{prod\_de}}$	$\mathfrak{r}_{\texttt{prod\_cn}}$	$\mathfrak{r}_{\texttt{prod\_oem}}$
prod_de	-40	–	0
prod_cn	–	-35	0
deliver	–	–	55

Table 4.1: Reward generated by the actions in the supply-chain example.

The possible histories are $(\texttt{prod\_de}, \texttt{deliver})$, $(\texttt{prod\_cn}\, o_{\texttt{f}})$ and $(\texttt{prod\_cn}\, o_{\texttt{s}}, \texttt{deliver})$, as well as the sub-histories with only the first action. The empty action parameters and observations are omitted for readability. From the perspective of global optimisation, the deterministic reward for going with the German production site is $(-40 + 55) = 15$. The expected reward with the production in China is $0.9(-35 + 55) + 0.1(-35) = 14.5$.

We now show some selected examples for the Q-value and the messages between the agents from Definition 4.17 and 4.18. Once the production sites have either produced or passed on the production, the scenario is "done" for them and no further rewards are generated. As de and cn have only one neighbor, no received messages are "mirrored back" in

$\mathfrak{q}_{\texttt{de}\to\texttt{oem}}$ and $\mathfrak{q}_{\texttt{cn}\to\texttt{oem}}$.

$$\mathfrak{q}_{\texttt{de}\to\texttt{oem}}(\texttt{prod\_de}\,|\,\cdot\,) = 0, \quad \mathfrak{q}_{\texttt{de}\to\texttt{oem}}(\texttt{pass\_de}\,|\,\cdot\,) = 0$$
$$\mathfrak{q}_{\texttt{cn}\to\texttt{oem}}(\texttt{prod\_cn}\,|\,\cdot\,) = 0, \quad \mathfrak{q}_{\texttt{cn}\to\texttt{oem}}(\texttt{pass\_cn}\,|\,\cdot\,) = 0$$

Given a either the production-production or the pass-action as a conditional, the optimisation of the actions is trivial for de and cn as there is only one action that can be selected in accordance with the conditional. The Equations 4.19 through 4.21 return the following.

$$\mathfrak{m}_{\texttt{de}\to\texttt{oem}}(\texttt{prod\_de}\,|\,\cdot\,) = \mathfrak{q}_{\texttt{de}\to\texttt{oem}}(\varepsilon\,|\,\texttt{prod\_de}) = -40$$
$$\mathfrak{m}_{\texttt{de}\to\texttt{oem}}(\texttt{pass\_de}\,|\,\cdot\,) = \mathfrak{q}_{\texttt{de}\to\texttt{oem}}(\varepsilon\,|\,\texttt{pass\_de}) = 0$$
$$\mathfrak{m}_{\texttt{cn}\to\texttt{oem}}(\texttt{prod\_cn}\,|\,\cdot\,) = \mathfrak{q}_{\texttt{cn}\to\texttt{oem}}(\varepsilon\,|\,\texttt{prod\_cn}) = -35$$
$$\mathfrak{m}_{\texttt{cn}\to\texttt{oem}}(\texttt{pass\_cn}\,|\,\cdot\,) = \mathfrak{q}_{\texttt{cn}\to\texttt{oem}}(\varepsilon\,|\,\texttt{pass\_cn}) = 0$$

With these messages transferred, the agent oem can continue with the computation of $q^*_{\texttt{oem}}$.

$$\begin{aligned}
\mathfrak{q}^*_{\texttt{oem}}(\texttt{prod\_cn}) = \mathbb{P}(o_{\texttt{s}})\Big[&\mathfrak{r}_{\texttt{oem}}(\texttt{prod\_cn}) + \\
&\bar{\mathfrak{r}}_{\texttt{de}\to\texttt{oem}}(\varepsilon, \texttt{prod\_cn}\,o_{\texttt{s}}) + \\
&\bar{\mathfrak{r}}_{\texttt{cn}\to\texttt{oem}}(\varepsilon, \texttt{prod\_cn}\,o_{\texttt{s}}) + \mathfrak{q}^*_{\texttt{oem}}(\texttt{prod\_cn}\,o_{\texttt{s}})\Big] + \\
\mathbb{P}(o_{\texttt{f}})\Big[&\mathfrak{r}_{\texttt{oem}}(\texttt{prod\_cn}) + \\
&\bar{\mathfrak{r}}_{\texttt{de}\to\texttt{oem}}(\varepsilon, \texttt{prod\_cn}\,o_{\texttt{f}}) + \\
&\bar{\mathfrak{r}}_{\texttt{cn}\to\texttt{oem}}(\varepsilon, \texttt{prod\_cn}\,o_{\texttt{f}}) + \mathfrak{q}^*_{\texttt{oem}}(\texttt{prod\_cn}\,o_{\texttt{f}})\Big] \\
= 0.9 \cdot [0 + 0 - 35 + \mathfrak{r}_{\texttt{oem}}(\texttt{deliver})] &+ \\
0.1 \cdot [0 + 0 - 35 + 0] = 14, 5&
\end{aligned}$$

The message from oem to its neighbors do not contain the reward that was signaled from the neighbor itself.

$$\mathfrak{m}_{\texttt{oem}\to\texttt{de}}(\texttt{prod\_de}) = 55, \quad \mathfrak{m}_{\texttt{oem}\to\texttt{de}}(\texttt{pass\_de}) = 14.5,$$
$$\mathfrak{m}_{\texttt{oem}\to\texttt{cn}}(\texttt{prod\_cn}) = 49.5, \quad \mathfrak{m}_{\texttt{oem}\to\texttt{de}}(\texttt{pass\_cn}) = 15$$

With that, the agents can jointly maximize the expected global reward even though they have only a limited scope to the system state and possible actions.

The messages $\mathfrak{m}_{i\to j}$ assign a value to every node of the shared history tree H_{ij}. Computing the messages $\mathfrak{m}_{i\to j}$ is computationally challenging. Instead of computing the message values directly, they are approximated via Monte-Carlo sampling. See Algorithm 15 for the full specification of the The Distributed Partially-Observable Hybrid Tree Planning (DPOHTP) algorithm.

Similar to the DUCT algorithm, the agents I are exchanging messages as they jointly explore the solution space. Every agent can use his private model for the components in his scope. So DPOHTP does not rely on a central simulator. The agents optimize for the global reward with their action choices. But they keep the reward for the different agents separated for the statistics on expected reward. The $\textsc{Play}_i$ procedure generates playout histories by sampling from the stochastic simulator and using the current reward statistics, exchanged messages and policy π for decision making. The $\textsc{Params}_i$ procedure is used to select parameters for the current action. It uses Optimistic Optimization similarly to StoSOO. But every call to $\textsc{Params}_i$ returns exactly one parameter vector. So several actions with a parameter each can be selected in one playout. The $\textsc{Update}_i$ procedure takes the last playout of the agent i and updates his internal reward statistics as well as the messages sent to the neighbors. Every parameter node stores the empirical direct reward that was generated by the action-parameter combination in a hashmap e_i. The Q-value of the parameter additionally considers the expected reward for optimal decision-making later on. Optimal decision-making here refers to the maximization of the global reward, taking the reward signaled by the neighboring agents via messages into account. Note that the messages are updated with an averaging procedure in lines 14–18 instead of maximizing over the actions and parameters. This is due to the fact that several histories $\boldsymbol{h}_i$ project to the same $\boldsymbol{h}_{ij}$. A maximization under the premise that only the current $\boldsymbol{h}_i$ is considered in the message would distort the message which represents an expectation over the future reward.

Algorithm 15 The Distributed Partially-Observable Hybrid Tree Planning (DPOHTP) algorithm

```
 1: procedure DPOHTP($\underline{\boldsymbol{\sigma}}^0$)
 2:     $m_{i\to j}[\,\cdot\,] \leftarrow 0\;\forall i \in I, j \in N(i)$
 3:     $n_{i\to j}[\,\cdot\,] \leftarrow 0\;\forall i \in I, j \in N(i)$
 4:     $n_i[\,\cdot\,] \leftarrow 0,\; q_i[\,\cdot\,] \leftarrow 0 \quad \forall i \in I$
 5:     $e_i[\,\cdot\,] \leftarrow 0 \quad \forall i \in I$
 6:     $L_i[\,\cdot\,] \leftarrow \{(0,1)\} \quad \forall i \in I$
 7:     $\bar{d} \leftarrow 1$
 8:     while enough time do
 9:         for $i \in I$ do
10:             $\boldsymbol{\sigma}_i \sim \underline{\boldsymbol{\sigma}}_i^0$
11:             $(\boldsymbol{h}_i, \boldsymbol{r}_i) \leftarrow \textsc{Play}_i(\boldsymbol{\sigma}_i^0)$
12:             $\textsc{Update}_i(\boldsymbol{h}_i, \boldsymbol{r}_i)$
13:             $\bar{d} \leftarrow \bar{d} + 1$
14:             if $\bar{d} > \log_2(n_i[\varepsilon]$ then
15:                 $\bar{d} \leftarrow 1$
16:     return $\underset{\substack{a\in A,\;\theta\in\Theta_a,\\ \forall i\in I_a,\; n_i[a\theta]>0}}{\arg\max} \left[\sum_{i\in I} q_i[\Pi_i(a\theta)]\right]$
```

```
 1: procedure $\textsc{Play}_i(\boldsymbol{\sigma}_i, \bar{d})$
 2:     $\boldsymbol{h}_i \leftarrow \varepsilon,\; \boldsymbol{r}_i \leftarrow \varepsilon$
 3:     while not $\textsc{done}(\boldsymbol{\sigma}_i)$ do
 4:         $B \leftarrow \{b_i \in A_i : n[\boldsymbol{h}_i b_i] = 0\}$
 5:         if $B \neq \varnothing$ then
 6:             $(a_i, \theta_i) \leftarrow \pi_i(\boldsymbol{h}_i, B_i)$
 7:             $\bar{d} \leftarrow \bar{d} - 1$
 8:         else
 9:             $a_i \leftarrow \underset{b_i\in A_i}{\arg\max} \Big[ q_i[\boldsymbol{h}_i b_i] + \alpha\sqrt{\frac{\log n_i[\boldsymbol{h}_i]+1}{n_i[\boldsymbol{h}_i b_i]}} + \sum_{j\in N(i)} m_{j\to i}[\Pi_{ij}(\boldsymbol{h}_i b_i)] \Big]$
10:             $\bar{d} \leftarrow \bar{d} - 1$
11:             $\theta_i \leftarrow \textsc{Param}_i(\boldsymbol{h}_i a_i, \bar{d})$
12:             $\bar{d} \leftarrow \bar{d} - \mathrm{depth}(L_i[\boldsymbol{h}_i a_i]$
13:         $(\boldsymbol{\sigma}_i, \boldsymbol{o}_i, v_i) \sim \underline{a}_i^{\theta_i}(\boldsymbol{\sigma}_i)$
14:         $\boldsymbol{h}_i \leftarrow \boldsymbol{h}_i a_i \theta_i \boldsymbol{o}_i,\; \boldsymbol{r}_i \leftarrow (\boldsymbol{r}_i, v_i)$
15:     return $\boldsymbol{h}_i, \boldsymbol{r}_i$
```

```
procedure $\textsc{Param}_i(\boldsymbol{h}_i a_i, đ)$
    if $đ > \mathrm{depth}(L[\boldsymbol{h}_i a_i])$ then
        return $\underset{\substack{\theta:\exists(u,v)\in L_i[\boldsymbol{h}_i a_i],\\ \boldsymbol{x}_i[\boldsymbol{h}_i a_i;u,v]=\theta}}{\arg\max}\; q[\boldsymbol{h}_i a_i \theta]$
    else if $đ < 1$ then
        $G \leftarrow L_i[\boldsymbol{h}_i a_i]$
    else
        $G \leftarrow L_i[\boldsymbol{h}_i a_i; đ]$
    $(u,v) \leftarrow \underset{\substack{(c,j)\in G,\\ \boldsymbol{x}_i[\boldsymbol{h}_i a_i;c,j]=\theta}}{\arg\max} \Big[ q_i[\boldsymbol{h}_i a_i \theta] + \alpha\sqrt{\frac{\log n_i[\boldsymbol{h}_i a_i]+1}{n_i[\boldsymbol{h}_i a_i \theta]}} + \sum_{j\in N(i)} m_{j\to i}(\Pi_{il}(\boldsymbol{h}_i a_i \theta)) \Big]$
    $\theta \leftarrow \boldsymbol{x}_i[\boldsymbol{h}_i a_i; u, v]$
    if $n_i[\boldsymbol{h}_i a_i \theta] < \kappa \wedge l = l^{\max}$ then
        return $\theta$
    $u[\boldsymbol{h}a; l, i] \leftarrow u[\boldsymbol{h}a; l, i] + 1,\ \mu \leftarrow u[\boldsymbol{h}a; l, i]$
    $n \leftarrow \dim(\Theta_a),\ \delta \leftarrow \mathrm{mod}(l, n) + 1$
    $\xi \leftarrow |L[\boldsymbol{h}a; l+1]| + 1$
    $\boldsymbol{x}[\boldsymbol{h}a; l+1, \xi] \leftarrow \theta + (\mu - 2)\boldsymbol{\nu}_\delta \left(\frac{1}{3}\right)^{\lfloor l/n \rfloor + 1}$
    $L[\boldsymbol{h}a] \leftarrow L[\boldsymbol{h}a] \cup \{(l+1, \xi)\}$
    if $\mu = 3$ then
        $L[\boldsymbol{h}a] \leftarrow L[\boldsymbol{h}a] \setminus \{(l, i)\}$
    return $\boldsymbol{x}[\boldsymbol{h}a; l+1, \xi]$
```

```
procedure $\textsc{Update}_i(\boldsymbol{h}_i, \boldsymbol{r}_i)$
    for $k = |\boldsymbol{h}_i|, \ldots, 1$ do
        $(a_i, \theta_i, \boldsymbol{o}_i) \leftarrow h_i^k$
        $\boldsymbol{g}_i \leftarrow \boldsymbol{h}_i^{:k-1}$
        $n_i[\boldsymbol{g}_i] \leftarrow n_i[\boldsymbol{g}_i] + 1,\ n_i[\boldsymbol{g}_i a_i] \leftarrow n_i[\boldsymbol{g}_i a_i] + 1,\ n_i[\boldsymbol{g}_i a_i \theta_i] \leftarrow n_i[\boldsymbol{g}_i a_i \theta_i] + 1$
        $e_i[\boldsymbol{g}_i a_i \theta_i] \leftarrow e_i[\boldsymbol{g}_i a_i \theta_i] + \frac{r_i^k - e_i[\boldsymbol{g}_i a_i \theta_i]}{n_i[\boldsymbol{g}_i a_i \theta_i]}$
        $q_i[\boldsymbol{g}_i a_i \theta_i] \leftarrow e_i[\boldsymbol{g}_i a_i \theta_i] + \sum_{\boldsymbol{o}_i \in O_i} \left[ q_i[\boldsymbol{g}_i a_i \theta_i \boldsymbol{o}_i] \frac{n[\boldsymbol{g}_i a_i \theta_i \boldsymbol{o}_i]}{n[\boldsymbol{g}_i a_i \theta_i]} \right]$
        $\theta_i^* \leftarrow \arg\max_{\substack{\phi_i \in \Theta_{a_i}\\ n[\boldsymbol{g}_i a_i \phi_i] > 0}} \left[ q_i[\boldsymbol{g}_i a_i \phi_i] + \sum_{j\in N(i)} m_{j\to i}(\Pi_{ij}(\boldsymbol{g}_i a_i \phi_i)) \right]$
        $q_i[\boldsymbol{g}_i a_i] \leftarrow q_i[\boldsymbol{g}_i a_i \theta_i^*]$
        $a_i^* \leftarrow \arg\max_{\substack{b_i \in A_i\\ n[\boldsymbol{g}_i b_i] > 0}} \left[ q_i[\boldsymbol{g}_i b_i] + \sum_{j\in N(i)} m_{j\to i}(\Pi_{ij}(\boldsymbol{g}_i b_i)) \right]$
        $q_i[\boldsymbol{g}_i] \leftarrow q_i[\boldsymbol{g}_i a_i^*]$
        for $j \in N(i) : a_i \in A_{ij}$ do
            $\boldsymbol{g}_{ij} \leftarrow \Pi_{ij}(\boldsymbol{g}_i)$
            $n_{i\to j}[\boldsymbol{g}_{ij} a_i \theta_i] \leftarrow n_{i\to j}[\boldsymbol{g}_{ij} a_i \theta_i] + 1,\ n_{i\to j}[\boldsymbol{g}_{ij} a_i] \leftarrow n_{i\to j}[\boldsymbol{g}_{ij} a_i] + 1$
            $q' \leftarrow q_i[\boldsymbol{g}_{ij} a_i \theta_i] + \sum_{l\in N(i)\setminus\{j\}} m_{l\to i}(\Pi_{il}(\boldsymbol{g}_i a_i \theta_i))$
            $m_{i\to j}[\boldsymbol{g}_{ij} a_i \theta_i] \leftarrow m_{i\to j}[\boldsymbol{g}_{ij} a_i \theta_i] + \frac{q' - m_{i\to j}[\boldsymbol{g}_{ij} a_i \theta_i]}{n_{i\to j}[\boldsymbol{g}_{ij} a_i \theta_i]}$
            $q'' \leftarrow q_i[\boldsymbol{g}_{ij} a_i] + \sum_{l\in N(i)\setminus\{j\}} m_{l\to i}(\Pi_{il}(\boldsymbol{g}_i a_i))$
            $m_{i\to j}[\boldsymbol{g}_{ij} a_i] \leftarrow m_{i\to j}[\boldsymbol{g}_{ij} a_i] + \frac{q'' - m_{i\to j}[\boldsymbol{g}_{ij} a_i]}{n[\boldsymbol{g}_{ij} a_i]}$
```

4.5 Evaluation

A preliminary version of DPOHTP was evaluated in [Pfr16a]. The main difference is that this version allows each agent to run simulations in a private simulator. So the agents are completely decoupled besides their message exchange. Existing benchmark examples from the Dec-POMDP literature assume a central simulator.

The scenario from autonomous driving was chosen since it enables a good visual understanding of the effect of agent coordination. Autonomous driving can be considered as one of the activites on the lower levels of the control hierarchy for supply-chain logistics. Furthermore, it shows the ability of the approach to adjust to changing system topologies with agent entering and leaving at runtime. Note that the example does not claim to be a physically accurate simulation for autonomous driving. It is a highly stylized benchmark example and not intended for practical use. Still, the model is based on results from the relevant literature that uses MCTS for planning of car maneuvers [LKK16]. If required, the simulation model can be readily replaced with something more accurate without changing the (implementation of) the GMCTS algorithm used for planning and coordination.

Two versions of the driving scenario are shown in Figures 4.4 and 4.5. Both start from the same initial situation. Three cars (`blue`, `green` and `red`) are driving close to each other at the same speed. They head toward the `grey` car that blocks the left lane after an accident. The cars have only a limited radius of visibility (assume some fog or heavy rain blocking the view). The cars seeing each other are marked with blue and green lines respectively.

The timeline of the scenario is discretized to 100 millisecond periods. In every period the cars may choose between five actions: `nothing`, `accelerate`, `break`, `moveLeft` and `moveRight`. The cars receives a reward at every iteration. A negative reward of -1 is assigned for every action besides `nothing`. This gives an incentive not to over-react. An additional negative reward of -50 is assigned to every car that crosses onto the shoulder of the road. Cars touching each other receive a reward of -1000 each. Each car has an internal simulator to predict future situations given joint actions for all cars in their visibility scope. Cars outside the visibility scope are not considered in the simulator. The simulator is the basis for MCTS-based planning.

In the first figure, the red car “sees” the accident at a very late point when the green car has already nearly passed. Until then, the red car stays in his line and keeps the original speed. Then, to prevent a crash that can now be predicted by the red car, the red car moves behind the blue car. The second figure was produced by identical parameters than the first figure. The only change is the exchange of messages between agents. So they use our DPOMCP algorithm instead of plain MCTS. It can be seen that the red car reacts much sooner to the accident, even before the grey car moves into his scope of visibility. The reason for this change of behavior is as follows: In the second line of Figure 4.5, the green car already sees the accident. Therefore the potential crash of the red and grey cars are predicted by the green car. When the green cars exchanges a message with preferences with the red car, then the danger of the crash is marginalized into the expected reward starting from possible shared action sequences. Thus, the red car learns about the potential strong negative consequences for staying in its line. The red cars therefore moves to the right lane and inserts even before the blue car. The change of exchanging messages leads to a very different – and more safe – outcome compared to the scenario without coordination.

The advantage of our work compared to previous results [FB10] is that it does not require that cars explicitly define coordination groups. Second, the complexity of the coordination algorithm grows exponentially only in the number of neighbors at every agent. Since every agent has a limited scope (and just a few neighbors), scaling to very large systems is easily possible. Third, when agents enter and leave the system at runtime, this has only a very local impact as just the direct neighbours need to coordinate.

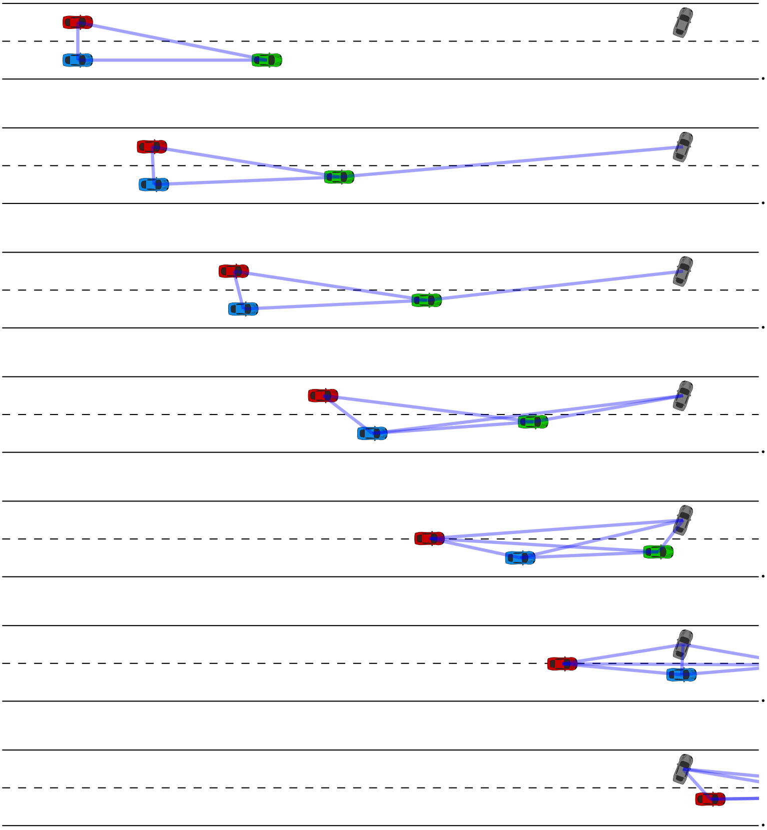


Figure 4.4: Autonomous driving example without coordination. The blue lines indicate neighbor relations.

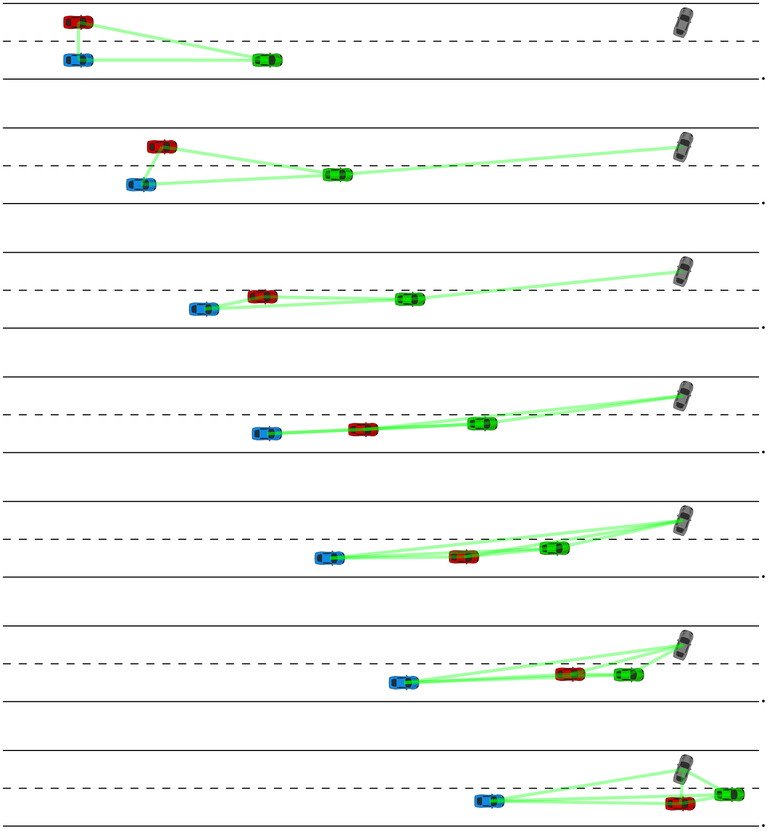


Figure 4.5: Autonomous driving example with coordination. The green lines indicate neighbor relations that are also communication relations.

5 Modeling of Production Skills

> *The tape was a small loop that fed continuously between magnetic pickups. On it were recorded the movements of a master machinist turning out a shaft. [...] Rudy Hertz, an old timer, who had been about ready to retire. And here, now, this little loop [...] was Rudy as Rudy had been to his machine that afternoon. Rudy, the turner-on of power, the setter of speeds, the controller of the cutting tool. This was the essence of Rudy as far as his machine was concerned.*
>
> *Kurt Vonnegut – Player Piano [Von52]*

The model of production systems developed in Chapter 2 demands that every possible behavior of the manufacturing resources is captured in the form of actions with well-defined preconditions and effects. This lays a heavy burden on plant operators to manage the set of available actions. Especially when changes are made to the plant topology and the targeted product lineup. While the actions are conceptually uniform, the complexity of a flexible manufacturing plant manifests itself in the large number of actions required to model the relevant aspects. For tens of considered final products with tens of manufacturing steps each, as well as tooling and transportation, plant operators need to model (and keep up-to-date) thousands of actions.

In addition to just defining the actions, their execution requires custom control code or the parameterization of reusable functionality for the product at hand. This custom effort to foresee, implement and deploy actions that are possibly needed at runtime is then a bottleneck for system integrators and operators who want to introduce changes at a later time. To reduce this burden, we introduce higher-level abstractions for the skills of the system components. Together with a similar description of the production requirements, the low-level actions are then generated automatically or with tool assistance.

Executable actions for planning and runtime control in a production systems can be generated from higher-level descriptions of the production system and the requirements.

Note: Our work on skill modeling [PSB13a; Pfr+14c; PSB13b; Pfr+15b] was published starting in 2013. The more recent literature already references and incorporates these results. This chapter provides a synthesis of our published work together with a novel representation of the concepts in Description Logics. Our own publications are not referenced in the summary of the state-of-the-art.

5.1 Background: Skill Models for Production

Sussman was one of the first to define a "theory of skill" that was not focused on the skilled human but on the computer [Sus73]. He writes:

> [...] a skill is a set of answer procedures, each indexed by a description of the problem types for which it is appropriate, along with a set of pitfalls to avoid when it is necessary to construct a new answer procedure. A skill is acquired by the construction of such a store of "runnable" knowledge – canned answers to problems – by "compiling" it from knowledge of the problem domain supplied in a more "intelligible" form – a form designed more for communication than for use as answers to problems.

This early work already talks about representations for the skill and the problem description, as well as the "compilation" of their combination into an executable form. Since then, work on technical skill definitions has continued mainly in the robotics domain. Many authors have worked on methods for learning of particular robot skills [GFB94; MK97; FRD98]. But we are rather interested in skill definitions that applies across different application domains.

In the last 10 years, interest in using skill definitions specifically for the production domain has considerably gained in importance. Some authors from the production community differentiate between the terms skill and capability. In this text, we will use the terms interchangeably.

The authors from the SIARAS project [Mal+07] use ontologies to store skills and their relations for production. This information is then used for automatic reconfiguration of production systems. Naumann et al. [NWS07] use an ontology to model robot skills and use state-charts for sequences within manufacturing processes. Järvenpää et al. [Jar+11] define an ontology for skills in manufacturing and use it to map resources to manufacturing steps. Kluge [Klu11] uses skill models for assembly planning. Huckaby and Christensen [HC12] provide a taxonomy for assembly tasks and related skill primitives. Constraints specify whether they are executable in a certain situation. Björkelund et al. [Bjö+12] first make use of the PPR (Product, Process, and Resource) concepts in the context of skills. They relate skills to all three views of PPR and represent skills as finite state machines. Keddis et al. [KKZ14] define a description vocabulary for skills of production resources and an accompanying algorithm for production planning and scheduling. Legat et al. [LSV14] use a description of the resource skills to guide engineers in the implementation of field control code. Backhaus et al. [BUR14] present a classification of manufacturing skills to enable task-oriented programming. This concept is expanded in [BR15] and used to generate executable tasks for a welding robot. Malakuti et al. discuss challenges of skill-based production control for practitioners and possible solutions [Mal+18]. The authors of [Jär+18] use semantic inference to determine the capabilities of resource combinations. For example of a robotic manipulator combined with a gripper.

Many interesting approaches to describe the skills of production resources via ontologies and high-level description schemas have been proposed in the literature. All authors report the successful application in the demonstration scenarios for which their description schemas were developed. We argue that modelling semantic knowledge about production systems facilitates the integration of resources in a Plug & Produce fashion. But it is not enough. First, as ontologies for describing manufacturing skills become more detailed, the less general they are. This leads to the difficult situation where

- general skill description schemas require additional information to cover implementation-specific edge-cases and constraints, and

- detailed skill description schemas are applicable to only a narrow subset of resources and will have difficulties in getting the required support across vendors and integration tooling suppliers. A possible scenario is the emergence of a standardized core skill description schema from which domain-specific and sometimes overlapping schemas will branch off.

Second, semantic reasoners were developed to infer further information from an existing knowledge-base. But they are not equipped for reasoning about numerical optimization problems, such as the resource- and time-efficient production of many products on concurrent resources. Therefore, we see the role of semantic skill descriptions primarily for integration and configuration tasks. To make the flexibility of generic resource skills available on the level of large-scale systems, they need to be propagated to dedicated planning and runtime control systems that may use different representations. This also offers the possibility to integrate overlapping and domain-specific skill description schemas if they can be interfaced to a unified abstraction used for planning and runtime control.

5.2 Background: Description Logics

Description Logics (DL) [Baa03] are a family of formal knowledge representation languages developed to represent hierarchical and relational structures and to enable reasoning over these structures. DL are the formal basis of semantic models with ontologies. DL models are defined in terms of *constants*, *concepts* and *roles*. The semantics of a DL is defined in terms of first-order logic. Hence, no DL is more powerful than first-order logic. Let the domain Δ a fixed countably infinite set. The interpretation $\mathcal{I}$ is a function $\cdot^{\mathcal{I}}$ that assigns

- to every constant c an element of Δ so that $c^{\mathcal{I}} \in \Delta$,
- to every concept C members of Δ as $C^{\mathcal{I}} \subseteq \Delta$,
- to every role R binary relations between members of Δ as $R^{\mathcal{I}} \subseteq \Delta \times \Delta$.

The interpretation has to be consistent with respect to assurances that are defined for the model.

Table 5.1 gives an overview on the syntax of the $\mathcal{EL}$ DL [KKS14] with the addition of concrete domains to express and reason about numerical attributes [BH91]. More expressive DL than $\mathcal{EL}$ exist. It was selected for the exposition due to its relative simplicity and the existence of performant solvers. By convention, we write the elements of DL models as follows: `Constants` are written in typewriter font, Concepts in a sans-serif font with a leading uppercase and hasRole definitions in a sans-serif font with a leading lowercase.

Example 5.1. Begin with the constants alice and bob. Both of them are humans and therefore, $\mathsf{Human}(\texttt{Alice})$ and $\mathsf{Human}(\texttt{Bob})$. Every human is either male or female. Therefore $\mathsf{Female} \sqsubseteq \mathsf{Human}$, $\mathsf{Male} \sqsubseteq \mathsf{Human}$ and $\mathsf{Female} \sqcap \mathsf{Male} \sqsubseteq \top$. Suppose `Alice` is the daughter of `Bob`. The role childOf describes the parent-child relation. We assert this relation between our two humans as $\mathsf{childOf}(\texttt{Alice}, \texttt{Bob})$. Daughters are the female children of a human. This can be stated as $\mathsf{daugherOf} \sqsubseteq \mathsf{childOf}$ with the domain $\mathsf{dom}(\mathsf{daughterOf}) \sqsubseteq \mathsf{Human}$ and range $\mathsf{ran}(\mathsf{daughterOf}) \sqsubseteq \mathsf{Human} \sqcap \mathsf{Female}$. From this we can infer that in effect $\mathsf{daughterOf}(\texttt{Alice}, \texttt{Bob})$.

Attributes from concrete domains are also assigned via role relations. Alice is 11 years old. So she has the attributes $\mathsf{hasAge}(\texttt{Alice}, 11)$ and $\mathsf{hasName}(\texttt{Alice}, \text{“Alice”})$. Queries over concrete domains are stated in the form of predicates. The concept $\mathsf{Human} \sqcap \exists \mathsf{hasAge}.(<, 12) \sqcap \exists \mathsf{hasName}.(=, \text{“Alice”})$ contains all humans named Alice and less than 12 years old.

	DL Syntax	Set-Theoretic Semantics
Concepts		
Universal Concept (Top)	$\top$	$\Delta^{\mathcal{I}}$
Empty Concept (Bottom)	$\bot$	$\varnothing$
Concept Assertion	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
Conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Inclusion	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
Restriction	$\exists R.C$	$\{x \mid \exists y : (x, y) \in R^{\mathcal{I}} \land y \in C^{\mathcal{I}}\}$
Roles		
Role Assertion	$R(a, b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$
Domain Restriction	$\mathsf{dom}(R) \sqsubseteq C$	$R^{\mathcal{I}} \subseteq C^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
Range Restriction	$\mathsf{ran}(R) \sqsubseteq C$	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times C^{\mathcal{I}}$
Inclusion	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
Composition	$R_1 \circ R_2 \sqsubseteq S$	$(x, y) \in R_1^{\mathcal{I}} \land (y, z) \in R_2^{\mathcal{I}} \rightarrow (x, z) \in S^{\mathcal{I}}$
Data Types		
Restriction	$\exists F.r$	$\{x \mid \exists v \in \mathcal{D} : (x, v) \in F^{\mathcal{I}} \land r(v)\}$

Table 5.1: Syntax and Semantics of the $\mathcal{EL}_{\bot}^{+}$ Description Logic.

5.3 The PPRS Model for Production Skills

We begin with an informal characterization of the PPRS model. In production, *processes* are used to create *products*. Processes stand for a type of operation, such as welding. The processes are performed by the *resources* of a production system, such as machines and technical equipment in general. *Skills* describe what a component is capable of in general. *Transformations* describe a specific production step that changes the attributes of a workpiece, consumes ingoing workpieces and material to create some output, and so on. Transformations are associated with one or more processes. *Actions* are realizations of a skill. For example to perform a specific transformation of a workpiece. Figure 5.1 gives a high-level overview on the six concepts and their relations. More precise definitions are now given, together with the representation the language of DL.

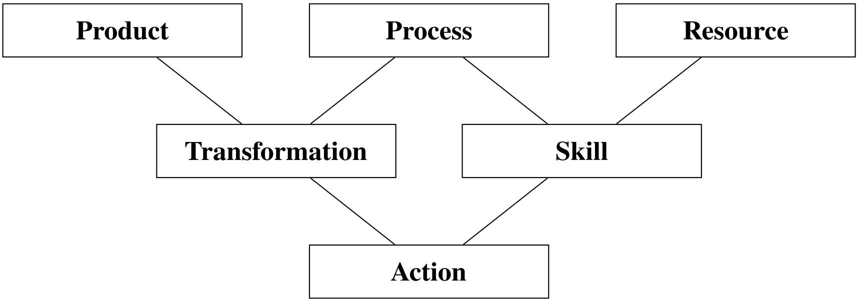


Figure 5.1: Outline of the relations between the PCM concepts.

Example 5.2. The example used for the remainder of this chapter comes from the production of automotive battery systems. The example is heavily simplified to serve as an educational example. From a high-level perspective, battery systems are comprised of battery cells that are welded to a conductive busbar. See [Das+18] for a detailed account of the different joining techniques in battery production.

Definition 5.1 (Product)**.** *A product is a type of marketable good, raw material or intermediate workpiece between production steps. Products describe discrete (countable) entities. Bulk material and fluids must be "packaged" to be considered a discrete product. Whenever product types and product instances need to be differentiated, product instances are denoted as* workpieces *to make the distinction clear.*

In DL, products are represented as constants. The properties of the product type are defined via attributes from concrete domains. The definition corresponds to the product definition from Chapter 2. Workpieces (product instances) of the same product type are interchangeable.

Example 5.3. The products defined for the battery production example are `cell`, `busbar` and `battery`.

$\mathsf{Product}(\texttt{cell})$
$\mathsf{Product}(\texttt{busbar})$
$\mathsf{Product}(\texttt{battery})$

Definition 5.2 (Process)**.** *Processes denote a type of production operation. Every process defines a set of process attributes that are used to characterize instances derived from the process. Processes form a hierarchy where attributes are inherited from parent processes.*

The term process is overloaded and understood differently in the fields of computer science, statistics, workflow management, production, and many more. The definition used here corresponds to the use of the term *manufacturing process* in DIN 8580 [DIN8580]. In DL, processes are represented as concepts. Processes form a hierarchy of concepts derived from the topmost concept Process.

Example 5.4. The processes defined for the battery production example are welding and the more specialized laser-welding. The processes are described by the power used for welding and, for the laser-welding process, the wavelength of the laser.

$\mathsf{LaserWelding} \sqsubseteq \mathsf{Welding} \sqsubseteq \mathsf{Process}$
$\mathsf{dom}(\mathsf{hasPower}) \sqsubseteq \mathsf{Welding}$
$\mathsf{ran}(\mathsf{hasPower}) = P(\mathbb{R}_+)$
$\mathsf{dom}(\mathsf{hasWaveLength}) \sqsubseteq \mathsf{LaserWelding}$
$\mathsf{ran}(\mathsf{hasWaveLength}) = P(\mathbb{R}_+)$

The use of the powerset $P(\mathbb{R}_+)$ as the concrete domain of the hasPower role allows instances of a welding process to refer to ranges of possible temperatures instead of a single scalar. Note that the concrete domains for process attributes may become quite complex, e.g. to describe the possible tool positions and rotations for a 5-axis CNC mill. Implementation may restrict concrete domains for example to one-dimensional ranges, where the representation and verification of predicates on a computer is trivial.

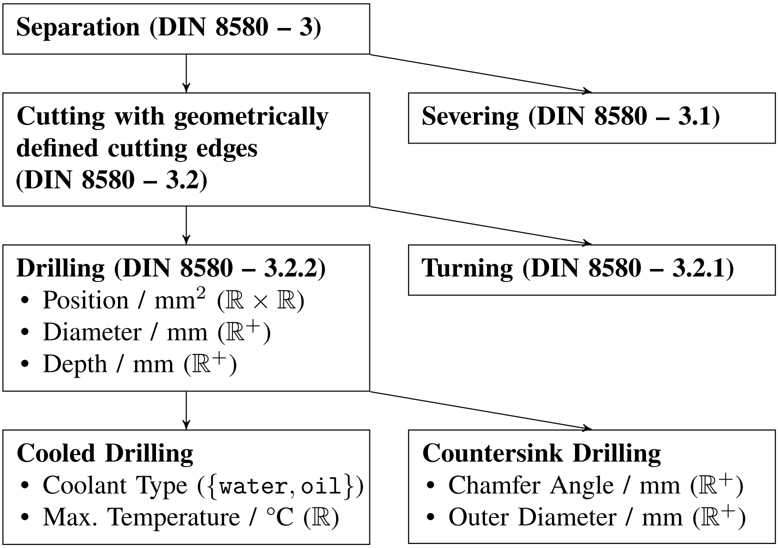

Separation (DIN 8580 – 3)
Cutting with geometrically defined cutting edges (DIN 8580 – 3.2)
Severing (DIN 8580 – 3.1)
Drilling (DIN 8580 – 3.2.2)
• Position / mm^2 ($\mathbb{R} \times \mathbb{R}$)
• Diameter / mm ($\mathbb{R}^+$)
• Depth / mm ($\mathbb{R}^+$)
Turning (DIN 8580 – 3.2.1)
Cooled Drilling
• Coolant Type ($\{\texttt{water}, \texttt{oil}\}$)
• Max. Temperature / °C ($\mathbb{R}$)
Countersink Drilling
• Chamfer Angle / mm ($\mathbb{R}^+$)
• Outer Diameter / mm ($\mathbb{R}^+$)

Figure 5.2: Excerpt from a hierarchy of production processes based on DIN 8580 [DIN8580]. The process attributes are from the indicated concrete domains. Arrows denote an inheritance relation, which is expressed in DL via concept inclusion, such as $\mathsf{CountersinkDrilling} \sqsubseteq \mathsf{Drilling}$.

Figure 5.2 shows a more complete example process hierarchy, specializing processes for drilling from DIN 8580. Processes usually originate from the production and logistics domain. But they can also be auxiliary to the core production operations. For example processes for machine maintenance.

Definition 5.3 (Resource). *Resources denote machines and technical assets in general. Resources are components in the nomenclature of Chapter 2.*

Renaming the components from Chapter 2 to resources may seem unnecessary. This is done in this chapter only to assure the possibility of comparison with existing production skill models from the literature.

In DL, resources are represented as constants. The set of all resources is denoted with the concept $\mathsf{Resource}$. In addition, derived concepts can be used to group resources. For example $\mathsf{5AxisManipulator} \sqsubseteq \mathsf{Manipulator} \sqsubseteq \mathsf{Resource}$. This is useful to query for specific resources. But this grouping of resource has no further purpose in this text.

Example 5.5. In the battery production example, we consider only one resource: The (imginary) LaserWelder200.

$\mathsf{Resource}(\mathtt{LaserWelder200})$

Definition 5.4 (Transformation)**.** *Transformations are production operations that transition one or several input products into one or several output products by the application of production processes.*

In DL, transformations are represented as constants that are instances of the $\mathsf{Transformation}$ concept. Two new roles, $\mathsf{hasInput}$ and $\mathsf{hasOutput}$, both with the range $\mathsf{Transformation}$ and the range $\mathsf{Product}$, are used to model which ingoing products are transformed to which output. The input/output relations between transformations can be used to draw a graph of a bill of processes. Note however, that transformations must act on a product. Auxiliary processes, such as machine tooling and transportation steps, are not considered.

In addition, transformations are also instances of one or more process concepts.

$$\mathsf{Transformation} \sqsubseteq \mathsf{Process}$$

With that, the transformations can assign values to the respective process attributes. The attributes of a transformation describe requirements that need to be fulfilled by implementations of the transformation. As described earlier, process attributes can have powersets for their domain in order to describe subsets, such as ranges. For example, a transformation for welding two workpieces together could indicate the range of supported temperatures in reference to the temperature attribute specified for the welding process.

Example 5.6. To produce a battery, cells are welded together with a busbar. The welding process requires a power to be applied from the range between 2kW and 3kW.

$\mathsf{Transformation}(\mathtt{JoinBatteryCells})$
$\mathsf{hasInput}(\mathtt{JoinBatteryCells}, \mathtt{cell})$

$\mathsf{hasInput}(\texttt{JoinBatteryCells}, \texttt{busbar})$
$\mathsf{hasOutput}(\texttt{JoinBatteryCells}, \texttt{battery})$

$\mathsf{Welding}(\texttt{JoinBatteryCells})$
$\mathsf{hasPower}(\texttt{JoinBatteryCells}, [2.000, 3.000])$

Definition 5.5 (Skill)**.** *Skills describe that a resource can execute a process under constraints defined in terms of the process attributes.*

In DL, skills are represented as constants. Similar to transformations, skills are also instances of processes.

$$\mathsf{Skill} \sqsubseteq \mathsf{Process}$$

Therefore, skills refer to the same attributes used to defined the transformation requirements. But it in the case of skills, the attributes describe the possibility to realize a process for given attributes.

Example 5.7. The `LaserWelder` has the skill to weld with a power between 2.5kW and 5kW. The CO2 laser used has a fixed wavelength of 10.6µm.

$\mathsf{Skill}(\texttt{LW200LaserWeld})$
$\mathsf{hasSkill}(\texttt{LaserWelder200}, \texttt{LW200LaserWeld})$

$\mathsf{LaserWelding}(\texttt{LW200LaserWeld})$
$\mathsf{hasPower}(\texttt{LW200LaserWeld}, [2.500, 5.000])$
$\mathsf{hasWavelength}(\texttt{LW200LaserWeld}, 10.6)$

Now we can formulate a query to find all skills that match with the JoinBatteryCells transformation in terms of supported processes and process attributes. This leads us to the resources that possess said skills. The $\cap$ operator is used to find attributes where the overlap with the indicated range is nonempty.

$\mathsf{SkillMatch} \sqsubseteq \mathsf{Welding} \sqcap \exists \mathsf{hasPower}(\cap, [2.000, 3.000])$
$\mathsf{ResourceMatch} \sqsubseteq \exists \mathsf{hasSkill}.\mathsf{SkillMatch}$

The skill representations cannot capture all details of a production system. Therefore, we can only test whether a resource can perform a transformation in general. For some domains, the attributes might be sufficient to guarantee feasibility of the match. In other domains, additional checks need to be performed either by detailed descriptions of the machine and product geometry or by manual intervention.

However, once a skill has been specialized for a transformation or auxiliary operation, implemented, tested and deployed to the production system, e.g. in the form of PLC control code, then we know the exact conditions under which the concrete operation can be applied.

Definition 5.6 (Action)**.** *An action is an executable process instance with well-defined preconditions and effects. Actions are defined for one or several participating resources. The DL concept* Action *contains instance elements that correspond to the formal action definition from Chapter 2.*

Actions are assumed to encapsulate all the required information to execute on the resource. For example in the form of IEC-61131 PLC code, configuration parameters, and so on. As a consequence, actions can be triggered simply by reference to their identifier, provided that all the preconditions are fulfilled.

Example 5.8. The action `LW200BatteryWeld` is an implementation of the `JoinBatteryCells` transformation. It makes use of the `LW200LaserWeld` skill.

Action(`LW200BatteryWeld`)
implements(`LW200BatteryWeld`, `JoinBatteryCells`)
uses(`LW200BatteryWeld`, `LW200LaserWeld`)

5.4 Assisted Generation of Executable Actions

In a Plug & Produce scenario, at some point the generic skills of machines and equipment need to be specialzied into executable actions with known preconditions and effects. This splits into two steps:

1. Selection of the actions that are required for runtime planning and control.

2. Implementation and deployment of the actions.

The assistend generation of actions was considered as part of the arhitecture of the SkillPro project. See Figure 5.3 for an overview.

The machines and equipment are assigned to a *Skill Execution Engine* (SEE). The SEE provide smart wrappers to the physical resources, which range from conveyor belts with little configurability to complex machine tools and even human workers. Facing towards the underlying resources, the SEE implement domain-specific connectivity, e.g. based on a fieldbus protocol or OPC UA. In case of the human worker, this is accomplished with a tablet-based graphical-user-interface. Also, the SEE may contain domain-specific knowledge on how to derive actions from high-level skill-based descriptions

The *Manufacturing Execution System* (MES) is responsible for orchestrating the available resources in order to achive short- to mid-term manufacturing goals. For this, the MES implements two main features working in lockstep: computing a fine-grained execution plan that accomplishes the manufacturing goals, and the orchestration of the manufacturing resources at runtime.

The *Asset Management System* AMS constitutes the central knowledge base of a manufacturing facility and provides this information to the adjacent components. It contains semantic descriptions of the available resources and their skills, as well as a detailed plant model including topological (e.g. how resources are arranged in work cells) and topographical (e.g. layout and position of the resources) relations. The AMS also holds product models, including drawings, bills of material and bills of processes. The AMS furthermore manages customer orders on a long-term horizon and ensures that the required resources with the right skills are available. Lastly, it interfaces the SkillPro framework with enterprise level ERP (Enterprise Resource Planning) and PDM (Product Data Management) systems.

Assume that a list of product transformations (bill of processes) are initially provided or it could be inferred from the product description itself [TMP92]. The task of determining the right operations to produce a specific product was originally investigated as Computer-Aided Process Planning (CAPP) [ElM93; Kir95]. CAPP deals with finding “a way through the production system”.

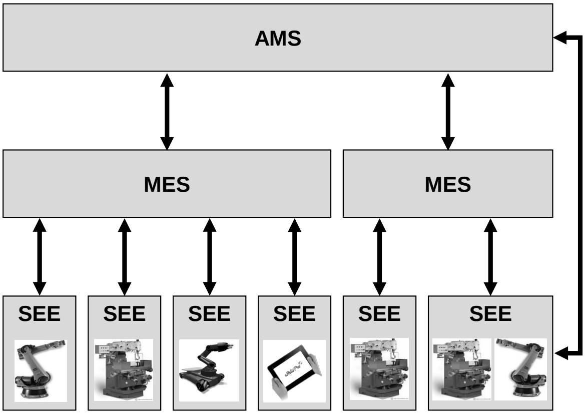


Figure 5.3: Architecture of the SkillPro project [Pfr+15b]

So CAPP is different from production planning and scheduling, where the production of many products on a given plant layout is considered.

The generation of executable actions from high-level skill-based descriptions requires the communication of AMS and SEE framework components. The SEE may internally implement actions with a variety of technologies. Possible ways for implementing actions are:

- Customization of predefined procedures by parametrization, where the defined parameters are either resource- or production-domain specific [ON15]. Note that a description of the entire product in an appropriate format may constitute a parameter in this context. More examples of reasoning on the execution of a high-level task model can be found for example in the RoboEarth project [TB09; Wai+11].
- Automatic code synthesis from a high-level description [VWK05], which can also be either resource-specific or based on a domain specific language [DM97b; MOW01; Mit+05].
- Manually programmed procedures, for example IEC-61131 function blocks.

The architecture from Figure 5.3 was implemented in the SkillPro project. At runtime, every SEE is represented by an OPC UA server that provides a uniform interface. Clients can connect to the OPC UA server and discover the current state of the component, as well as the available actions. So from the perspective of a higher-level control system, the SEE are all uniform. Even if they represent very different types of production equipment. If coordination between components for an action is required (for example the time-synchronization beginning of a procedure), they can also use OPC UA or rely some other communication technology in the background. The latter is rather disapproved of, since this adds technically rigid solution where tool-support for quick adaptations in the sense of Plug & Work do not have as much tool-support.

6 Conclusion

Automated production systems face a tradeoff between efficiency and flexibility. This thesis aims to improve the flexibility of automated production systems by the use of a unified model representation on all levels of the control hierarchy. Recall the postulate from the outset of Chapter 2:

> The same set of modeling principles can represent the relevant properties of production systems on all levels of the control hierarchy.

The thesis developed modeling principles that are able to represent both continuous and discrete production on all levels of the control hierarchy, including concurrency, i.e. parallel operations and the synchronization of system components, as well as uncertainty in stochastic scenarios. To our knowledge, no prior approach was able to encompass all of these properties. In order to make use of such a model for planning and runtime control appropriate decisions need to be derived. Chapter 3 started with the following postulate:

> The same algorithm can be used for planning and runtime control on all levels of the control hierarchy – ranging from continuous dynamics of a physical system to global supply-chain operations – and for both continuous and discrete production.

Accordingly, an algorithm for optimal sequential decision making was developed based on forward-simulation of the model from Chapter 2. Now that a model for all levels of the control hierarchy and a matching planning algorithm exist, the remaining challenge is to speed up planning for production operations at an industrial scale. A range of measures were developed in this thesis to speed up planning.

- Pruning action sequences that are equivalent in a precise sense (Section 3.1).

- The use of Monte-Carlo Tree Search and Optimistic Optimisation to replace trivial tree search and Branch & Bound methods (Section 3.2 and Section 3.3).
- The relaxation of an important special case of the planning problem to a Mixed-Integer Linear Program (Section 3.4).

A way to speed up planning that is orthogonal to the techniques just mentioned is to decompose the planning problem into smaller subproblems to be solved by independent agents. For this, the algorithm from Chapter 3 is extended for multi-agent coordination. The chapter sets out to achieve the following target result:

> Independent agents can jointly perform planning in a production scenario where every agent only has a simulation model of the system part in his visible scope.

The core idea of the developed planning algorithm is to use “utility propagation” for the agent coordination similar to “belief propagation” in probabilistic graphical models. The decomposition into independent agent not only reduces the planning complexity. Companies in a supply-chain can jointly optimize their actions without a central entity that has access to all private information.

But in order to optimise production with planning algorithms, the model representations of the production system need to be accurate. Furthermore, if the resulting plans shall be used for automated control, then the action abstractions used in the model need to available as automated procedures on the actual machines and equipment. Keeping the model and the physical production system synchronized can become quite resource-intensive when the production is flexible and changes over time. It is therefore preferable to automate much of the configuration work as well. The beginning of Chapter 5 postulates the following.

> Executable actions for planning and runtime control in a production systems can be generated from higher-level descriptions of the production system and the requirements.

We put forward a framework to describe the skills of production equipment based on Description Logic (i.e. semantic modeling). The framework also encompasses the description of product transformations and the manufacturing processes involved. These descriptions are then used to match production steps with machines and equipment capable of performaing than. Then, executable actions are generated and deployed for use by the runtime control systems. This is relevant for to achieve flexible production. Instead of manually programming a PLC, higher-level abstractions can be used to generate and parameterized the required control code.

The thesis has answered all four postulates in the affirmative. Example from different production scenarios and from all levels of the control hierarchy have been used to substantiate the claim. But surely this work can only be a stepping stone towards future automated production systems that are both efficient and flexible. To realize this goal, a large research programme is necessary that goes beyond the scope of a single thesis. We now enumerate open research questions and some of the key results that need to be obtained future for the work from different domains to coalesce into a coherent whole.

Derive existing models for production and logistics from a common core The production and logistics domain has developed numerous approaches for modeling operations with varying degree of detail. This text has proposed a high-fidelity model as the common basis for all such models used in production and logistics. To support that claim, examples from several production domains and on several levels of the automation hierarchy have been used for the exposition.

But a more comprehensive treatment is required next to mere examples. It is an open research question which limiting assumptions need to be made on top of the high-fidelity model to recover more coarse-grained models that are already in productive use today. Such an investigation is the basis for an automated treatment of interfaces where subsystems that are controlled by different modeling approaches connect.

Using Machine Learning to Guide Exploration Monte-Carlo Tree Search (MCTS) as a planning algorithm is essentially blind once it reaches a point in the scenario tree that was never visited before. Machine Learning methods can be used to enable MCTS to “learn to see”. The Q-value of the system state (or a belief distribution for the system state) can be approximated from similar experiences in the past that were (reached with a different sequence of actions). In that sense, the model structure is not explicitly represented a-priori but learned from the interaction with the scenario over a series of plays. This has becomes known as Approximate Dynamic Programming [Ber+05; Pow07]. The combination of MCTS with neural networks as function approximators has famously been used in the Deepmind AlphaGo system [Sil+16]. The expectation is that a further reduction of the sample complexity for simulation-based planning with concurrent production system model can be achieved.

There is currently an active community working on Multi-Agent Reinforcement Learning (MARL) [BBD08]. Recent advances in deep learning are now being integrated with MARL. Recent work also lets the agents learn how to communicate instead of predefining the information flow [Foe+16]. "Utility propagation" as used in Chapter 4 could be combined with recent MARL techniques. For example by sending neural networks for Q-value estimation as messages instead of an explicit Q-value representations on a search-tree.

Explicit representation of product attributes The model from Chapter 2 assumes that products are interchangeable. Instead of representing them individually, only a count of products of the same type at the possible locations is considered. But there are of course differences between products of the same product type. This becomes apparent for example when quality issues come up. Quality (for which different the definition depends on the case at hand) is then conditionally dependent on the attributes of incoming raw material and semi-finished products, process settings and the state of the physical process. A complete "theory of production" should therefore encompass product attributes as well.

Open systems with agents entering and leaving at runtime The decentralized planning approach proposed in Chapter 4 allows for agents to enter and leave the system at runtime. Such changes do not have to be communicated within the entire system and only neighboring agents have to be informed. A remaining question is to see how fast the overall system can adjust to the changes.

Domain-specific action generation The generation of executable actions from skill definitions requires domain-specific tool support. In some domains, commercial products are already available that can transform higher-level descriptions to executable definitions. Notably in the robotics domain where clear categories of equipment exist (cf. `https://www.artiminds.com` and `https://www.keba.com`). Another example is the standardized “G-code” for numerically controlled machine tools [ISO82]. G-code can be exported by virtually CAD/CAM tools that target product design. The control synthesis solutions for the individual application domains then have to be integrated. For example to assist the integration of systems that combine robotics for loading and unloading into a machine tool that transforms the work piece and visual inspection for quality control. To us, high-level descriptions based on skill-definitions are the most promising inroad for control synthesis in domains with more variety in the types of machines and equipment as well as a larger range of products to consider.

Bibliography

[Abe+06] E Abele et al. “Globalization and decentralization of manufacturing”. In: *Reconfigurable Manufacturing Systems and Transformable Factories*. Springer, 2006, pp. 3–13.

[ABZ88] Joseph Adams, Egon Balas, and Daniel Zawack. “The shifting bottleneck procedure for job shop scheduling”. In: *Management science* 34.3 (1988), pp. 391–401.

[ACF02] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. “Finite-time analysis of the multiarmed bandit problem”. In: *Machine learning* 47.2-3 (2002), pp. 235–256.

[Agn+14] Alessandro Agnetis et al. *Multiagent scheduling*. Springer, 2014.

[Ali05] Knut Alicke. *Planung und Betrieb von Logistiknetzwerken*. Springer, 2005.

[AM00] Srinivas M Aji and Robert J McEliece. “The generalized distributive law”. In: *IEEE Transactions on Information Theory* 46.2 (2000), pp. 325–343.

[AO15] Christopher Amato and Frans A Oliehoek. “Scalable Planning and Learning for Multiagent POMDPs”. In: *Twenty-Ninth AAAI Conference on Artificial Intelligence*. 2015, pp. 1995–2002.

[Ara+00] T Arai et al. “Agile assembly system by ’plug and produce’”. In: *CIRP Annals-Manufacturing Technology* 49.1 (2000), pp. 1–4.

[Aza+16] Selma Azaiez et al. “Towards Flexibility in Future Industrial Manufacturing: A Global Framework for Self-organization of Production Cells”. In: *Procedia Computer Science* 83 (2016), pp. 1268–1273.

[Baa03] Franz Baader. *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003.

[Bac+92] François Baccelli et al. “Synchronization and linearity: an algebra for discrete event systems”. In: (1992).

[Bad11] Iman Badr. “Agent-based dynamic scheduling for flexible manufacturing systems”. PhD thesis. University of Stuttgart, 2011.

[Bas+75] Forest Baskett et al. “Open, closed, and mixed networks of queues with different classes of customers”. In: *Journal of the ACM (JACM)* 22.2 (1975), pp. 248–260.

[BBD08] Lucian Busoniu, Robert Babuska, and Bart De Schutter. “A comprehensive survey of multiagent reinforcement learning”. In: *IEEE Transactions on Systems, Man, And Cybernetics-Part C: Applications and Reviews, 38 (2), 2008* (2008).

[BCG07] Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing multi-agent systems with JADE*. Vol. 7. John Wiley & Sons, 2007.

[BD08] Ronen I Brafman and Carmel Domshlak. “From One to Many: Planning for Loosely Coupled Multi-Agent Systems.” In: *ICAPS*. 2008, pp. 28–35.

[BDH99] Craig Boutilier, Thomas Dean, and Steve Hanks. “Decision-theoretic planning: Structural assumptions and computational leverage”. In: *Journal of Artificial Intelligence Research* 11.1 (1999), p. 94.

[Bel57] Richard Ernest Bellman. “Dynamic Programming”. In: (1957).

[Ben93] Stuart Bennett. “Development of the PID controller”. In: *IEEE control systems* 13.6 (1993), pp. 58–62.

[Ber+05] Dimitri P Bertsekas et al. *Dynamic programming and optimal control*. Vol. 1. 3. Athena scientific Belmont, MA, 2005.

[Ber+95] Dimitri P Bertsekas et al. *Dynamic programming and optimal control*. Vol. 1. 2. Athena scientific Belmont, MA, 1995.

[Ber38] Daniel Bernoulli. *Hydrodynamica*. Dulsecker, 1738.

[BG01] Blai Bonet and Héctor Geffner. “Planning and control in artificial intelligence: A unifying perspective”. In: *Applied Intelligence* 14.3 (2001), pp. 237–252.

[BH91] Franz Baader and Philipp Hanschke. "A Scheme for Integrating Concrete Domains into Concept Languages". In: *Proceedings of the 12th International Joint Conference on Artificial Intelligence - Volume 1*. IJCAI'91. Sydney, New South Wales, Australia: Morgan Kaufmann Publishers Inc., 1991, pp. 452–457. ISBN: 1-55860-160-0. URL: `http://dl.acm.org/citation.cfm?id=1631171.1631239`.

[Bjö+12] Anders Björkelund et al. "Knowledge for Intelligent Industrial Robots." In: *AAAI Spring Symposium: Designing Intelligent Robots*. 2012.

[BPM18] Lucian Buşoniu, Elődd Páll, and Rémi Munos. "Continuous-action planning for discounted infinite-horizon nonlinear optimal control with Lipschitz values". In: *Automatica* 92 (2018), pp. 100–108.

[BR15] J Backhaus and G Reinhart. "Adaptive and Device Independent Planning Module for Task-Oriented Programming of Assembly Systems". In: *Procedia CIRP* 33 (2015), pp. 545–550.

[Bri03] Douglas Brinkley. *Wheels for the world: Henry Ford, his company, and a century of progress, 1903-2003*. Viking Press, 2003.

[Bro+12] Cameron B Browne et al. "A survey of monte carlo tree search methods". In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (2012), pp. 1–43.

[Bro+84] Jim Browne et al. "Classification of flexible manufacturing systems". In: *The FMS magazine* 2.2 (1984), pp. 114–117.

[BS88] MIRYAM BARAD and Daniel Sipper. "Flexibility in manufacturing systems: definitions and Petri net modelling". In: *International Journal of Production Research* 26.2 (1988), pp. 237–248.

[BUR14] Julian Backhaus, Marco Ulrich, and Gunther Reinhart. "Classification, Modelling and Mapping of Skills in Automated Production Systems". In: *Enabling Manufacturing Competitiveness and Economic Sustainability*. Springer, 2014, pp. 85–89.

[Bus+13] Lucian Busoniu et al. “Optimistic planning for continuous-action deterministic systems”. In: *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2013 IEEE Symposium on*. IEEE. 2013, pp. 69–76.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[BW05] Dimitris Bertsimas and Robert Weismantel. *Optimization over integers*. Dynamic Ideas, Belmont, 2005.

[BZI00] Daniel S Bernstein, Shlomo Zilberstein, and Neil Immerman. “The complexity of decentralized control of Markov decision processes”. In: *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc. 2000, pp. 32–37.

[Cân+11] Gonçalo Cândido et al. “Service-oriented infrastructure to support the deployment of evolvable production systems”. In: *Industrial Informatics, IEEE Transactions on* 7.4 (2011), pp. 759–767.

[Car84] Nancy Cartwright. *How the laws of physics lie*. Oxford University Press, 1984.

[CF69] Pierre Cartier and Dominique Foata. *Problemes combinatoires de commutation et réarrangements*. Vol. 85. Lecture Notes in Mathematics. Springer, 1969.

[Cha+08] Guillaume Chaslot et al. “Monte-Carlo Tree Search: A New Framework for Game AI.” In: *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*. 2008.

[Chr00] Martin Christopher. “The agile supply chain: competing in volatile markets”. In: *Industrial marketing management* 29.1 (2000), pp. 37–44.

[CM15] Felipe Caro and Victor Martínez-de-Albéniz. “Fast fashion: business model overview and research opportunities”. In: *Retail Supply Chain Management*. Springer, 2015, pp. 237–264.

[Coa37] Ronald H Coase. “The nature of the firm”. In: *Economica* 4.16 (1937), pp. 386–405.

[Com93] International Electrotechnical Commission. *IEC 61131-3: Programmable Controllers*. Tech. rep. 1993.

[Cow+99] Robert G Cowell et al. “Probabilistic Networks and Expert Systems”. In: (1999).

[Dan66] Sven Danø. *Industrial production models: A theoretical study*. Springer, 1966.

[Das+18] Abhishek Das et al. “Joining technologies for automotive battery systems manufacturing”. In: *World Electric Vehicle Journal* 9.2 (2018), p. 22.

[De +08] Luciana Moreira Sá De Souza et al. “Socrades: A web service based shop floor integration infrastructure”. In: *The internet of things*. Springer, 2008, pp. 50–67.

[De 70] Morris H De Groot. “Optimal statistical decisions”. In: (1970).

[Dep+16] Torben Deppe et al. “Bidirektionale Kommunikation mit OPC Unified Architecture”. In: *Kommunikation in der Automation - KommA 2016*. Lemgo, Germany, Nov. 2016.

[DF04] David S Dummit and Richard M Foote. *Abstract Algebra*. John Wiley and Sons, 2004.

[Dim15] Todor Dimitrov. “Permanente Optimierung dynamischer Probleme der Fertigungssteuerung unter Einbeziehung von Benutzerinteraktionen”. PhD thesis. Karlsruhe Institute of Technology, 2015.

[DIN18] DIN. *DIN SPEC 16593-1: RM-SA - Reference Model for Industrie 4.0 Service Architectures - Part 1: Basic Concepts of an Interaction-based Architecture*. Tech. rep. 2018.

[DIN8580] *DIN8580: Manufacturing processes - Terms and definitions, division*. Standard. Deutsches Institut für Normung, 2003.

[DM97a] Volker Diekert and Yves Métivier. “Partial commutation and traces”. In: *Handbook of formal languages* 3 (1997), pp. 457–533.

[DM97b] Donald Dragomatz and Stephen Mann. “A classified bibliography of literature on NC milling path generation”. In: *Computer-Aided Design* 29.3 (1997), pp. 239–247.

[Dor+17] Kirill Dorofeev et al. “Device adapter concept towards enabling plug&produce production environments”. In: *Emerging Technologies and Factory Automation (ETFA), 2017 22nd IEEE International Conference on*. IEEE. 2017, pp. 1–8.

[DP87] Neil A Duffie and Rex S Piper. “Non-hierarchical control of a flexible manufacturing cell”. In: *Robotics and computer-integrated manufacturing* 3.2 (1987), pp. 175–179.

[DT98] Alberto De Toni and Stefano Tonchia. “Manufacturing flexibility: a literature review”. In: *International journal of production research* 36.6 (1998), pp. 1587–1617.

[Dür+12] Lars Dürkop et al. “Towards autoconfiguration of industrial automation systems: A case study using Profinet IO”. In: *Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on*. IEEE. 2012, pp. 1–8.

[Dür+14] Lars Dürkop et al. “A field level architecture for reconfigurable real-time automation systems”. In: *Factory Communication Systems (WFCS), 2014 10th IEEE Workshop on*. IEEE. 2014, pp. 1–10.

[DW91] Thomas L Dean and Michael P Wellman. *Planning and control*. Morgan Kaufmann Publishers Inc., 1991.

[Dyc03] Harald Dyckhoff. “Neukonzeption der Produktionstheorie”. In: *Zeitschrift für Betriebswirtschaft* 73.7 (2003), pp. 705–732.

[Dyc06] Harald Dyckhoff. *Produktionstheorie: Grundzüge industrieller Produktionswirtschaft*. Springer-Verlag, 2006.

[ElM93] Hoda A ElMaraghy. “Evolution and future perspectives of CAPP”. In: *CIRP Annals-Manufacturing Technology* 42.2 (1993), pp. 739–751.

[Fär88] Rolf Färe. *Fundamentals of production theory*. Springer, 1988.

[FB10] Christian Frese and Jürgen Beyerer. “Planning cooperative motions of cognitive automobiles using tree search algorithms”. In: *Annual Conference on Artificial Intelligence*. Springer. 2010, pp. 91–98.

[FD16] Pascal Faure and Philippe Darmayan. “Le plan français «Industrie du futur»”. In: *Annales des Mines-Réalités industrielles*. 4. FFE. 2016, pp. 57–60.

[FFW07] Victor Fung, William Fung, and Yoram Jerry Wind. *Competing in a flat world: building enterprises for a borderless world*. Wharton School Publishing, 2007.

[Fis99] Klaus Fisher. “Agent-based design of holonic manufacturing systems”. In: *Robotics and autonomous Systems* 27.1-2 (1999), pp. 3–13.

[Foe+16] Jakob Foerster et al. “Learning to communicate with deep multi-agent reinforcement learning”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 2137–2145.

[FRD98] Holger Friedrich, Oliver Rogalla, and Rüdiger Dillmann. “Integrating skills into multi-agent systems”. In: *Journal of Intelligent Manufacturing* 9.2 (1998), pp. 119–127.

[FT63] H. Fisher and G.L. Thompson. “Probabilistic learning combinations of local job-shop scheduling rules”. In: *Industrial Scheduling*. Ed. by J.F. Muth and G.L. Thompson. Prentice Hall, 1963, pp. 225–251.

[Fuj98] Richard M Fujimoto. “Time management in the high level architecture”. In: *Simulation* 71.6 (1998), pp. 388–400.

[Fur18] Kai Furmans. *Material Handling and Production Systems Modelling-Based on Queuing Models*. Springer, 2018.

[FYK92] Katsuhisa Furuta, M Yamakita, and S Kobayashi. “Swing-up control of inverted pendulum using pseudo-state feedback”. In: *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering* 206.4 (1992), pp. 263–269.

[Gar18] Gartner. *Market Share Analysis: ERP Software, Worldwide, 2017*. Tech. rep. 2018. URL: `https://www.gartner.com/doc/3879510/market-share-analysis-erp-software`.

[Ger93] Donald Gerwin. “Manufacturing flexibility: a strategic perspective”. In: *Management science* 39.4 (1993), pp. 395–410.

[Ger99] Gary Gereffi. “International trade and industrial upgrading in the apparel commodity chain”. In: *Journal of international economics* 48.1 (1999), pp. 37–70.

[GF08] Harald Gleißner and J Christian Femerling. *IT in der Logistik*. Springer, 2008.

[GFB94] Vijaykumar Gullapalli, Judy A Franklin, and Hamid Benbrahim. “Acquiring robot skills via reinforcement learning”. In: *IEEE Control Systems* 14.1 (1994), pp. 13–24.

[GG89] Yash P Gupta and Sameer Goyal. “Flexibility of manufacturing systems: concepts and measurements”. In: *European journal of operational research* 43.2 (1989), pp. 119–135.

[Gie04] Ronald N Giere. “How models are used to represent reality”. In: *Philosophy of science* 71.5 (2004), pp. 742–752.

[GJS76] Michael R Garey, David S Johnson, and Ravi Sethi. “The complexity of flowshop and jobshop scheduling”. In: *Mathematics of operations research* 1.2 (1976), pp. 117–129.

[GLK98] Ling Gou, Peter B Luh, and Yuji Kyoya. “Holonic manufacturing scheduling: architecture, cooperation mechanism, and implementation”. In: *Computers in Industry* 37.3 (1998), pp. 213–231.

[Gol99] Jonathan S Golan. *Semirings and their applications*. Kluwer Academic Publishers, 1999.

[GPP15] Sten Grüner, Julius Pfrommer, and Florian Palm. “A RESTful extension of OPC UA”. In: *Factory Communication Systems (WFCS), 2015 IEEE World Conference on*. IEEE, 2015, pp. 1–4.

[GPP16] Sten Grüner, Julius Pfrommer, and Florian Palm. “RESTful Industrial Communication With OPC UA”. In: *IEEE Transactions on Industrial Informatics* 12.5 (2016), pp. 1832–1841.

[Gro05] Ignacio Grossmann. “Enterprise-wide optimization: A new frontier in process systems engineering”. In: *AIChE Journal* 51.7 (2005), pp. 1846–1857.

[Gro08] Donald Gross. *Fundamentals of queueing theory*. John Wiley & Sons, 2008.

[GT60] Bernard Giffler and Gerald Luther Thompson. “Algorithms for solving production-scheduling problems”. In: *Operations research* 8.4 (1960), pp. 487–503.

[Gur16] Gurobi Optimization, Inc. *Gurobi Optimizer Reference Manual*. 2016. URL: `http://www.gurobi.com`.

[GVM16] Jean-Bastien Grill, Michal Valko, and Rémi Munos. “Blazing the trails before beating the path: Sample-efficient Monte-Carlo planning”. In: *Advances in Neural Information Processing Systems*. 2016, pp. 4680–4688.

[HC12] Jacob Huckaby and Henrik I Christensen. “A taxonomic framework for task modeling and knowledge transfer in manufacturing robotics”. In: *Workshops at 26th AAAI Conference on Artificial Intelligence*. 2012.

[Hem66] J Hemelrijk. “Underlining random variables”. In: *Statistica Neerlandica* 20.1 (1966), pp. 1–7.

[HL05] Willy Herroelen and Roel Leus. “Project scheduling under uncertainty: Survey and research potentials”. In: *European journal of operational research* 165.2 (2005), pp. 289–306.

[Hou85] David Hounshell. *From the American system to mass production, 1800-1932: The development of manufacturing technology in the United States*. 4. JHU Press, 1985.

[HU79] John E Hopcroft and Jeffrey D Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley, 1979.

[ISO82] ISO. *ISO 6983-1: Automation systems and integration – Numerical control of machines – Program format and definitions of address words – Part 1: Data format for positioning, line motion and contouring control systems*. 1982.

[Jac+07] F Robert Jacobs et al. “Enterprise resource planning (ERP)—A brief history”. In: *Journal of Operations Management* 25.2 (2007), pp. 357–363.

[Jar+11] E Jarvenpaa et al. “Presenting capabilities of resources and resource combinations to support production system adaptation”. In: *Assembly and Manufacturing (ISAM), 2011 IEEE International Symposium on*. IEEE. 2011, pp. 1–6.

[Jär+18] Eeva Järvenpää et al. “Utilizing SPIN rules to infer the parameters for combined capabilities of aggregated manufacturing resources”. In: *IFAC-PapersOnLine* 51.11 (2018), pp. 84–89.

[Jay03] Edwin T Jaynes. *Probability theory: The logic of science*. Cambridge university press, 2003.

[Jel06] Mohieddine Jelali. “An overview of control performance assessment technology and industrial applications”. In: *Control engineering practice* 14.5 (2006), pp. 441–466.

[JM98] Anant Singh Jain and Sheik Meeran. *A state-of-the-art review of job-shop scheduling techniques*. Tech. rep. Department of Applied Physics, Electronic and Mechanical Engineering, University of Dundee, Dundee, Scotland, 1998.

[Joh87] David G Johnson. *Programmable controllers for factory automation*. Marcel Dekker, Inc., 1987.

[JPS93] Donald R Jones, Cary D Perttunen, and Bruce E Stuckman. “Lipschitzian optimization without the Lipschitz constant”. In: *Journal of Optimization Theory and Applications* 79.1 (1993), pp. 157–181.

[JS05] François Jammes and Harm Smit. “Service-oriented paradigms in industrial automation”. In: *Industrial Informatics, IEEE Transactions on* 1.1 (2005), pp. 62–70.

[Jun+17] Jieun Jung et al. “Design of smart factory web services based on the industrial internet of things”. In: *Proceedings of the 50th Hawaii International Conference on System Sciences*. 2017.

[KBT17] Ilya Kovalenko, Kira Barton, and Dawn Tilbury. “Design and implementation of an intelligent product agent architecture in manufacturing systems”. In: *Emerging Technologies & Factory Automation (ETFA)*. IEEE, 2017, pp. 1–8.

[Ken15] Scott Kennedy. “Made in China 2025”. In: *Center for Strategic and International Studies* (2015).

[KF09] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[KFL01] Frank R Kschischang, Brendan J Frey, and H-A Loeliger. “Factor graphs and the sum-product algorithm”. In: *Information Theory, IEEE Transactions on* 47.2 (2001), pp. 498–519.

[KHL08] Hanna Kurniawati, David Hsu, and Wee Sun Lee. “SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces.” In: *Robotics: Science and systems*. Vol. 2008. 2008.

[Kir95] Dimitris Kiritsis. “A review of knowledge-based expert systems for process planning. Methods and problems”. In: *The International Journal of Advanced Manufacturing Technology* 10.4 (1995), pp. 240–262.

[KKS14] Yevgeny Kazakov, Markus Krötzsch, and František Simančık. “The incredible ELK”. In: *Journal of automated reasoning* 53.1 (2014), pp. 1–61.

[KKZ14] Nadine Keddis, Gerd Kainz, and Alois Zoitl. “Capability-based planning and scheduling for adaptable manufacturing systems”. In: *Emerging Technology and Factory Automation (ETFA), 2014 IEEE*. IEEE. 2014, pp. 1–8.

[KLC98] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. “Planning and acting in partially observable stochastic domains”. In: *Artificial intelligence* 101.1 (1998), pp. 99–134.

[Klu11] Stefan Kluge. “Methodik zur fähigkeitsbasierten Planung modularer Montagesysteme”. PhD thesis. Universität Stuttgart, 2011.

[KMN02] Michael Kearns, Yishay Mansour, and Andrew Y Ng. “A sparse sampling algorithm for near-optimal planning in large Markov decision processes”. In: *Machine learning* 49.2-3 (2002), pp. 193–208.

[Koe68] Arthur Koestler. “The ghost in the machine.” In: (1968).

[KS06] Levente Kocsis and Csaba Szepesvári. “Bandit based monte-carlo planning”. In: *European conference on machine learning*. Springer. 2006, pp. 282–293.

[Kuh62] Thomas S Kuhn. *The structure of scientific revolutions*. University of Chicago Press, 1962.

[KV05] Jelle R Kok and Nikos Vlassis. “Using the max-plus algorithm for multiagent decision making in coordination graphs”. In: *Robot Soccer World Cup*. Springer. 2005, pp. 1–12.

[KWH13] H Kagermann, W Wahlster, and J Helbig. “Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0–Abschlussbericht des Arbeitskreises Industrie 4.0”. In: *Forschungsunion im Stifterverband für die Deutsche Wissenschaft. Berlin* (2013).

[Lam78] Leslie Lamport. “Time, clocks, and the ordering of events in a distributed system”. In: *Communications of the ACM* 21.7 (1978), pp. 558–565.

[LaV06] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[Lei09] Paulo Leitão. “Agent-based distributed manufacturing control: A state-of-the-art survey”. In: *Engineering Applications of Artificial Intelligence* 22.7 (2009), pp. 979–991.

[Lep+11] Wilfried Lepuschitz et al. “Toward self-reconfiguration of manufacturing systems using automation agents”. In: *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 41.1 (2011), pp. 52–69.

[Lib11] Daniel Liberzon. *Calculus of variations and optimal control theory: a concise introduction*. Princeton University Press, 2011.

[Lim15] Li&Fung Limited. *Li&Fung Annual Report 2015*. 2015. URL: `https://www.lifung.com/investors/financial-reports-presentations/2015/`.

[Lit+63] John DC Little et al. “An algorithm for the traveling salesman problem”. In: *Operations research* 11.6 (1963), pp. 972–989.

[LK08] J-H Lee and C-O Kim. “Multi-agent systems applications in manufacturing systems and supply chain management: a review paper”. In: *International Journal of Production Research* 46.1 (2008), pp. 233–265.

[LK15] Paulo Leitão and Stamatis Karnouskos. *Industrial Agents: Emerging Applications of Software Agents in Industry*. Morgan Kaufmann, 2015.

[LKK16] David Lenz, Tobias Kessler, and Alois Knoll. “Tactical cooperative planning for autonomous highway driving using Monte-Carlo Tree Search”. In: *Intelligent Vehicles Symposium (IV), 2016 IEEE*. IEEE. 2016, pp. 447–453.

[LL94] Tim C Lueth and Thomas Laengle. “Task description, decomposition, and allocation in a distributed autonomous multi-agent robot system”. In: *Intelligent Robots and Systems’ 94.’Advanced Robotic Systems and the Real World’, IROS’94. Proceedings of the IEEE/RSJ/GI International Conference on*. Vol. 3. IEEE. 1994, pp. 1516–1523.

[LMV13] Paulo Leitão, Vladimír Mařík, and Pavel Vrba. “Past, present, and future of industrial agent applications”. In: *IEEE Transactions on Industrial Informatics* 9.4 (2013), pp. 2360–2372.

[Los+11] Matthias Loskyll et al. “Semantic service discovery and orchestration for manufacturing processes”. In: *Emerging Technologies & Factory Automation (ETFA)*. IEEE. 2011, pp. 1–8.

[Los+12] Matthias Loskyll et al. “Context-based orchestration for control of resource-efficient manufacturing processes”. In: *Future Internet* 4.3 (2012), pp. 737–761.

[LP12] Gisela Lanza and Steven Peters. “Integrated capacity planning over highly volatile horizons”. In: *CIRP Annals-Manufacturing Technology* 61.1 (2012), pp. 395–398.

[LPW97] Hau L Lee, Venkata Padmanabhan, and Seungjin Whang. “Information distortion in a supply chain: The bullwhip effect”. In: *Management science* 43.4 (1997), pp. 546–558.

[LS92] Grace Yuh-jiun Lin and James J Solberg. “Integrated shop floor control using autonomous agents”. In: *IIE transactions* 24.3 (1992), pp. 57–71.

[LSV14] Christoph Legat, Daniel Schütz, and Birgit Vogel-Heuser. “Automatic generation of field control strategies for supporting (re-) engineering of manufacturing systems”. In: *Journal of Intelligent Manufacturing* 25.5 (2014), pp. 1101–1111.

[Lue69] David G Luenberger. *Optimization by vector space methods*. John Wiley & Sons, 1969.

[LV15] Christoph Legat and Birgit Vogel-Heuser. “An Orchestration Engine for Services-Oriented Field Level Automation Software”. In: *Service Orientation in Holonic and Multi-agent Manufacturing*. Ed. by Theodor Borangiu, André Thomas, and Damien Trentesaux. Vol. 594. Springer International Publishing, 2015, pp. 71–80.

[Mac+06] C Matthew MacKenzie et al. *OASIS Reference model for service oriented architecture 1.0*. Tech. rep. 2006.

[Mac02] Jan Marian Maciejowski. *Predictive control: with constraints*. Pearson education, 2002.

[Mal+07] J. Malec et al. “Knowledge-Based Reconfiguration of Automation Systems”. In: *Automation Science and Engineering, 2007. CASE 2007. IEEE International Conference on*. 2007, pp. 170–175.

[Mal+18] Somayeh Malakuti et al. “Challenges in Skill-based Engineering of Industrial Automation Systems”. In: *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*. Vol. 1. IEEE. 2018, pp. 67–74.

[Maz77] Antoni Mazurkiewicz. “Concurrent program schemes and their interpretations”. In: *DAIMI Report Series* 6.78 (1977).

[MB00] Duncan C McFarlane and Stefan Bussmann. “Developments in holonic production planning and control”. In: *Production Planning & Control* 11.6 (2000), pp. 522–536.

[Mes13] VDI/VDE - Gesellschaft für Mess und Automatisierungstechnik (GMA). *Cyber-Physical Systems: Chancen und Nutzen aus Sicht der Automation*. Tech. rep. 2013.

[MF70] Burton G Malkiel and Eugene F Fama. “Efficient capital markets: A review of theory and empirical work”. In: *The journal of Finance* 25.2 (1970), pp. 383–417.

[Mit+05] S Mitsi et al. “Off-line programming of an industrial robot for manufacturing”. In: *The International Journal of Advanced Manufacturing Technology* 26.3 (2005), pp. 262–267.

[MK97] J Daniel Morrow and Pradeep K Khosla. “Manipulation task primitives for composing robot skills”. In: *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*. Vol. 4. IEEE. 1997, pp. 3354–3359.

[ML99] Manfred Morari and Jay H Lee. “Model predictive control: past, present and future”. In: *Computers & Chemical Engineering* 23.4 (1999), pp. 667–682.

[MLD09] Wolfgang Mahnke, Stefan-Helmut Leitner, and Matthias Damm. *OPC Unified Architecture*. Springer, 2009.

[Mon14] László Monostori. “Cyber-physical production systems: Roots, expectations and R&D challenges”. In: *Procedia Cirp* 17 (2014), pp. 9–13.

[MOW01] Swee M Mok, Kenlip Ong, and Chi-haur Wu. “Automatic generation of assembly instructions using STEP”. In: *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*. Vol. 1. IEEE. 2001, pp. 313–318.

[Mun+14] Rémi Munos et al. "From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning". In: *Foundations and Trends in Machine Learning* 7.1 (2014), pp. 1–129.

[Mun11] Rémi Munos. "Optimistic optimization of a deterministic function without the knowledge of its smoothness". In: *Advances in neural information processing systems*. 2011, pp. 783–791.

[MVK06] László Monostori, József Váncza, and Soundar RT Kumara. "Agent-based systems for manufacturing". In: *CIRP Annals-Manufacturing Technology* 55.2 (2006), pp. 697–720.

[MW59] Born Max and Emil Wolf. *Principles of optics*. Pergamon Press, 1959.

[MWL11] Christopher R Mansley, Ari Weinstein, and Michael L Littman. "Sample-Based Planning for Continuous Action Markov Decision Processes." In: *ICAPS*. 2011.

[Nas51] John Nash. "Non-cooperative games". In: *Annals of mathematics* (1951), pp. 286–295.

[Neu55] John von Neumann. *The impact of recent developments in science on the economy and on economics*. Speech to the National Planning Association, Washington D.C., reprinted in *Collected Works* (Pergamon Press, 1963). 1955.

[NHR99] Andrew Y Ng, Daishi Harada, and Stuart Russell. "Policy invariance under reward transformations: Theory and application to reward shaping". In: *ICML*. Vol. 99. 1999, pp. 278–287.

[NW10] Peter Nyhuis and Hans-Peter Wiendahl. "Ansatz zu einer Theorie der Produktionstechnik". In: *ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb* 105.1-2 (2010), pp. 15–20.

[NWS07] Martin Naumann, Kai Wegener, and Rolf Dieter Schraft. "Control architecture for robot cells to enable Plug'n'Produce". In: *Robotics and Automation, 2007 IEEE International Conference on*. IEEE. 2007, pp. 287–292.

[OHN14] Jens Otto, Steffen Henning, and Oliver Niggemann. "Why cyber-physical production systems need a descriptive engineering approach–a case study in plug & produce". In: *Procedia Technology* 15 (2014), pp. 295–302.

[Ohn88] Taiichi Ohno. *Toyota production system: beyond large-scale production*. CRC Press, 1988.

[ON15] Jens Otto and Oliver Niggemann. "Automatic Parameterization of Automation Software for Plug-and-Produce". In: *The AAAI-15 Workshop on Algorithm Configuration (AlgoConf 2015), Austin, Texas* (2015).

[Ono+12] Mauro Onori et al. "The IDEAS project: plug & produce at shop-floor level". In: *Assembly automation* 32.2 (2012), pp. 124–134.

[Oue+99] Djamila Ouelhadj et al. "A multi-contract net protocol for dynamic scheduling in flexible manufacturing systems (FMS)". In: *Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on*. Vol. 2. IEEE. 1999, pp. 1114–1119.

[Pal+14] Florian Palm et al. "open62541 - der offene OPC UA-Stack". In: Lemgo, Nov. 2014.

[Par87] H Van Dyke Parunak. "Manufacturing experience with the contract net". In: *Distributed Artificial Intelligence, Volume I*. Elsevier, 1987, pp. 285–310.

[Pea84] Judea Pearl. "Heuristics: intelligent search strategies for computer problem solving". In: (1984).

[Pea88] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., 1988.

[Pfr+14a] Julius Pfrommer et al. "Begrifflichkeiten um Industrie 4.0 – Ordnung im Sprachwirrwarr". In: *Tagungsband zu Entwurf komplexer Automatisierungssysteme (EKA) 2014*. Ed. by Ulrich Jumar and Christian Diedrich. Magdeburg, May 2014.

[Pfr+14b] Julius Pfrommer et al. “Dynamic vehicle redistribution and online price incentives in shared mobility systems”. In: *IEEE Transactions on Intelligent Transportation Systems* 15.4 (2014), pp. 1567–1578.

[Pfr+14c] Julius Pfrommer et al. “Modelling and Orchestration of Service-Based Manufacturing Systems via Skills”. In: *Emerging Technologies & Factory Automation (ETFA), 2014 IEEE 19th Conference on*. Barcelona, Spain, Sept. 2014.

[Pfr+15a] Julius Pfrommer et al. “Plug & Produce by Modelling Skills and Service-Oriented Orchestration of Reconfigurable Manufacturing Systems”. In: *at Automatisierungstechnik* 10.63 (2015).

[Pfr+15b] Julius Pfrommer et al. “Plug & produce by modelling skills and service-oriented orchestration of reconfigurable manufacturing systems”. In: *at-Automatisierungstechnik* 63.10 (2015), pp. 790–800.

[Pfr+16a] Julius Pfrommer et al. “A common core for information modeling in the Industrial Internet of Things”. In: *at-Automatisierungstechnik* 64.9 (2016), pp. 729–741.

[Pfr+16b] Julius Pfrommer et al. “Deploying software functionality to manufacturing resources safely at runtime”. In: *Emerging Technologies and Factory Automation (ETFA), 2016 IEEE 21st International Conference on*. Berlin, Germany: IEEE, Sept. 2016, pp. 1–7.

[Pfr+18] Julius Pfrommer et al. “Optimisation of manufacturing process parameters using deep neural networks as surrogate models”. In: *Proceedings of the 51st CIRP Conference on Manufacturing Systems*. Stockholm: CIRP, 2018.

[Pfr14] Julius Pfrommer. *Information and Control in Cyber-Physical Production Systems*. Tech. rep. 2014, pp. 61–74.

[Pfr15] Julius Pfrommer. *Distributed Constraint Optimization over Constrained Communication Topologies*. Tech. rep. 2015, pp. 77–87.

[Pfr16a] Julius Pfrommer. “Graphical Partially Observable Monte-Carlo Planning”. In: *Workshop on Learning, Inference and Control of Multi-Agent Systems, Conference on Neural Information Processing Systems (NIPS)*. Dec. 2016.

[Pfr16b] Julius Pfrommer. “Semantic Interoperability at Big-Data Scale with the open62541 OPC UA Implementation”. In: *2nd International Workshop on Interoperability and Open-Source Solutions for the Internet of Things (InterOSS-IoT)*. Stuttgart, Germany, Nov. 2016.

[Pfr16c] Julius Pfrommer. *Towards Graphical Partially Observable Monte-Carlo Planning*. Tech. rep. 2016, pp. 113–125.

[PGP16] Julius Pfrommer, Sten Grüner, and Florian Palm. “Hybrid OPC UA and DDS: Combining architectural styles for the industrial internet”. In: *Factory Communication Systems (WFCS), 2016 IEEE World Conference on*. Aveiro, Portugal: IEEE, May 2016, pp. 1–7.

[Pin08] Michael L Pinedo. “Scheduling: Theory, Algorithms, and Systems”. In: (2008).

[PLM13] Jani Puttonen, Andrei Lobov, and Jose L Martinez Lastra. “Semantics-based composition of factory automation processes encapsulated by web services”. In: *Industrial Informatics, IEEE Transactions on* 9.4 (2013), pp. 2349–2359.

[Pow07] Warren B Powell. *Approximate Dynamic Programming: Solving the curses of dimensionality*. Vol. 703. John Wiley & Sons, 2007.

[Pro+17] Stefan Profanter et al. “OPC UA for plug & produce: Automatic device discovery using LDS-ME”. In: *Proceedings of the IEEE International Conference on Emerging Technologies And Factory Automation (ETFA)*. IEEE. 2017.

[PSB13a] Julius Pfrommer, Miriam Schleipen, and Jürgen Beyerer. “Fähigkeiten adaptiver Produktionsanlagen”. In: *atp-edition* 55 (11) (2013).

[PSB13b] Julius Pfrommer, Miriam Schleipen, and Jürgen Beyerer. “PPRS: Production skills and their relation to product, process, and resource”. In: *Proceedings of the 2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*. IEEE. Cagliari, Italy, 2013.

[Put94] Martin L Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

[RD05] Paul Renteln and Alan Dundes. “Foolproof: A sampling of mathematical folk humor”. In: *Notices of the AMS* 52.1 (2005), pp. 24–34.

[Rei+10] G Reinhart et al. “Automatic configuration (plug & produce) of industrial ethernet networks”. In: *Industry Applications (INDUSCON), 2010 9th IEEE/IAS International Conference on*. IEEE. 2010, pp. 1–6.

[Rei91] J Francis Reintjes. *Numerical control: making a new technology*. Oxford University Press, Inc., 1991.

[Ros+08] Stéphane Ross et al. “Online planning algorithms for POMDPs”. In: *Journal of Artificial Intelligence Research* 32 (2008), pp. 663–704.

[Sal+10] Yves Sallez et al. “The lifecycle of active and intelligent products: The augmentation concept”. In: *International Journal of Computer Integrated Manufacturing* 23.10 (2010), pp. 905–924.

[Sch+11] Günther Schuh et al. “Developing a production engineering based theory of production”. In: *Concurrent Enterprising (ICE), 2011 17th International Conference on*. IEEE. 2011, pp. 1–9.

[Sch+15a] Miriam Schleipen et al. “Requirements and concept for plug-and-work”. In: *at-Automatisierungstechnik* 63.10 (2015), pp. 801–820.

[Sch+15b] Günther Schuh et al. “Hypotheses for a Theory of Production in the Context of Industrie 4.0”. In: *Advances in Production Technology*. Springer, 2015, pp. 11–23.

[Sch+16] Miriam Schleipen et al. “OPC UA & Industrie 4.0 - enabling technology with high diversity and variability”. In: *49th CIRP Conference on Manufacturing Systems (CIRP-CMS 2016)*. Stuttgart, Germany: CIRP, May 2016.

[Sch+17] Günther Schuh et al. “Towards a technology-oriented theory of production”. In: *Integrative Production Technology*. Springer, 2017, pp. 1047–1079.

[Sch04] Christoph Schneeweiß. “On the empirical validity of production theory”. In: *Central European Journal of Operations Research* 12.2 (2004), p. 107.

[Sch34] Erich Schneider. *Theorie der Produktion*. J. Springer, 1934.

[Sch86] Christoph Schneeweiß. *Einführung in die Produktionswirtschaft*. Springer, 1986.

[She+07] Weiming Shen et al. “An agent-based service-oriented integration architecture for collaborative intelligent manufacturing”. In: *Robotics and Computer-Integrated Manufacturing* 23.3 (2007), pp. 315–325.

[She03] Khalid Sheikh. *Manufacturing resource planning (MRP II): with introduction to ERP, SCM and CRM*. McGraw-Hill New York, NY, 2003.

[She71] Ronald William Shepherd. *Theory of cost and production functions*. Princeton University Press, 1971.

[Sil+16] David Silver et al. “Mastering the game of Go with deep neural networks and tree search”. In: *nature* 529.7587 (2016), p. 484.

[Sil+17] David Silver et al. “Mastering the game of Go without human knowledge”. In: *Nature* 550.7676 (2017), p. 354.

[SKB97] Ashraf Saad, Kazuhiko Kawamura, and Gautam Biswas. “Performance evaluation of contract net-based heterarchical scheduling for flexible manufacturing systems”. In: *Intelligent Automation & Soft Computing* 3.3 (1997), pp. 229–247.

[Sko04] Sigurd Skogestad. “Control structure design for complete chemical plants”. In: *Computers & Chemical Engineering* 28.1 (2004), pp. 219–234.

[Smi80] Reid G Smith. “The contract net protocol: High-level communication and control in a distributed problem solver”. In: *IEEE Transactions on computers* 12 (1980), pp. 1104–1113.

[SP97] Rainer Storn and Kenneth Price. “Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces”. In: *Journal of global optimization* 11.4 (1997), pp. 341–359.

[SS73] Richard D Smallwood and Edward J Sondik. “The optimal control of partially observable Markov processes over a finite horizon”. In: *Operations research* 21.5 (1973), pp. 1071–1088.

[SS90] Andrea Krasa Sethi and Suresh Pal Sethi. “Flexibility in manufacturing: a survey”. In: *International journal of flexible manufacturing systems* 2.4 (1990), pp. 289–328.

[Sus73] Gerald J Sussman. “A computational model of skill acquisition”. PhD thesis. 1973.

[SV10] David Silver and Joel Veness. “Monte-Carlo planning in large POMDPs”. In: *Advances in neural information processing systems*. 2010, pp. 2164–2172.

[SWH06] Weiming Shen, Lihui Wang, and Qi Hao. “Agent-based distributed manufacturing process planning and scheduling: a state-of-the-art survey”. In: *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* 36.4 (2006), pp. 563–577.

[SZW17] Jan-Philipp Schmidt, Andreas Zeller, and Michael Weyrich. “Modellgetriebene Entwicklung serviceorientierter Anlagensteuerungen”. In: *at-Automatisierungstechnik* 65.1 (2017), pp. 26–36.

[TB09] Moritz Tenorth and Michael Beetz. “KnowRob: knowledge processing for autonomous personal robots”. In: *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. IEEE. 2009, pp. 4261–4266.

[TMP92] HK Tönshoff, E Menzel, and HS Park. “A knowledge-based system for automated assembly planning”. In: *CIRP Annals-Manufacturing Technology* 41.1 (1992), pp. 19–24.

[UPS14] Thomas Usländer, Julius Pfrommer, and Miriam Schleipen. “Das Internet der Dinge in der Automation - Anforderungen und Technologien”. In: *5. Jahreskolloquium "Kommunikation in der Automation" (KommA 2014)*. Lemgo, 2014.

[Van+98] Hendrik Van Brussel et al. “Reference architecture for holonic manufacturing systems: PROSA”. In: *Computers in industry* 37.3 (1998), pp. 255–274.

[VCM13] Michal Valko, Alexandra Carpentier, and Rémi Munos. “Stochastic simultaneous optimistic optimization”. In: *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*. 2013, pp. 19–27.

[VHL03] Guilherme E Vieira, Jeffrey W Herrmann, and Edward Lin. “Rescheduling manufacturing systems: a framework of strategies, policies, and methods”. In: *Journal of scheduling* 6.1 (2003), pp. 39–62.

[Von52] Kurt Vonnegut. *Player Piano*. Charles Scribner’s Sons, 1952.

[VWK05] Birgit Vogel-Heuser, Daniel Witsch, and Uwe Katzke. “Automatic code generation from a UML model to IEC 61131-3 and system configuration tools”. In: *Control and Automation, 2005. ICCA’05. International Conference on*. Vol. 2. IEEE. 2005, pp. 1034–1039.

[Vya11] Valeriy Vyatkin. “IEC 61499 as enabler of distributed and intelligent automation: State-of-the-art review”. In: *IEEE transactions on Industrial Informatics* 7.4 (2011), pp. 768–781.

[Wai+11] M. Waibel et al. “RoboEarth”. In: *IEEE Robotics Automation Magazine* 18.2 (2011), pp. 69–82.

[Wal12] Mark John Walker. “The programmable logic controller: its prehistory, emergence and application”. PhD thesis. The Open University, 2012.

[War93] Hans-Jürgen Warnecke. *The Fractal Company—A Revolution in Corporate Culture*. Springer, 1993.

[WD51] Marshal K. Wood and George B. Dantzig. "The programming of interdependent activities: general discussion". In: *Activity analysis of production and allocation*. Ed. by Tjalling C. Koopmans. John Wiley & Sons, Inc., 1951.

[Wel03] Lloyd R Welch. "Hidden Markov models and the Baum-Welch algorithm". In: *IEEE Information Theory Society Newsletter* 53.4 (2003), pp. 10–13.

[Wey+14] Michael Weyrich et al. "Flexibilisierung von Automatisierungssystemen - Systematisierung der Flexibilitätsanforderungen von Industrie 4.0". In: *wt Werkstattstechnik online* 104.3 (2014), pp. 106–111.

[Wie+07] H-P Wiendahl et al. "Changeable manufacturing-classification, design and operation". In: *CIRP Annals–Manufacturing Technology* 56.2 (2007), pp. 783–809.

[Wie48] Norbert Wiener. "Cybernetics; or control and communication in the animal and the machine". In: (1948).

[Wig81] Oliver W Wight. *MRP II: Unlocking America's productivity potential*. Omneo, 1981.

[WJ+08] Martin J Wainwright, Michael I Jordan, et al. "Graphical models, exponential families, and variational inference". In: *Foundations and Trends® in Machine Learning* 1.1–2 (2008), pp. 1–305.

[WNH10] H-P Wiendahl, P Nyhuis, and W Hartmann. "Should CIRP develop a Production Theory? Motivation, Development Path, Framework". In: *43rd CIRP International Conference on Manufactoring Systems*. CIRP. 2010.

[Woo95] Robert Simpson Woodward. "An Historical Survey of the Science of Mechanics". In: *Science* 1.6 (1895), pp. 141–157.

[WS11] Stephan M Wagner and Victor Silveira-Camargos. "Decision model for the application of just-in-sequence". In: *International Journal of Production Research* 49.19 (2011), pp. 5713–5736.

[WS15] Stephanie Wong and Paula Sailes. “Wal-Mart Takes Back Some Goods Sourcing From Li & Fung”. In: (2015). URL: `https://www.bloomberg.com/news/articles/2015-05-22/wal-mart-takes-back-some-goods-sourcing-business-from-li-fung`.

[XL08] Wei Xiang and Heow Pueh Lee. “Ant colony intelligence in multi-agent dynamic manufacturing scheduling”. In: *Engineering Applications of Artificial Intelligence* 21.1 (2008), pp. 73–85.

[YFW01] Jonathan S Yedidia, William T Freeman, and Yair Weiss. “Bethe free energy, Kikuchi approximations, and belief propagation algorithms”. In: *Advances in neural information processing systems* 13 (2001).

[ZR89] Gilad Zlotkin and Jeffrey S Rosenschein. “Negotiation and task sharing among autonomous agents in cooperative domains”. In: *Proceedings of the 11th International Joint Conference on Artificial Intelligence (IJCAI)*. Morgan Kaufmann Publishers Inc. 1989, pp. 912–917.

[Zur14] Richard Zurawski. *Industrial communication technology handbook*. CRC Press, 2014.