October 2023

Cyber-Physical Multi-Robot Systems in a Smart Factory: A Networked AI Agents Approach

Zixiang Nie
University of South Florida

Cyber-Physical Multi-Robot Systems in a Smart Factory

A Networked AI Agents Approach

by

Zixiang Nie

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Electrical Engineering
College of Engineering
University of South Florida

Date of Approval:
September 21, 2023

Keywords: Multi-agent System, Cyber-Physical System, Multi-agent Coordination, Multi-agent Collaboration, Autonomous AI

Dedication

To my husband, Andrew Gettings, whose unwavering support was pivotal in the completion of this dissertation; to my mother, Xuejun Zhu; my father, Zhicun Nie; and my cherished grandfather, Jin Zhu - your encouragement has been a driving force throughout this journey. Dr. Kwang-Cheng Chen, my Ph.D. supervisor, has been a monumental source of inspiration. The ideas birthed in this dissertation are largely a reflection of our enlightening conversations. I am deeply thankful for the unwavering support from my family and friends; their presence made the challenges of this Ph.D. study surmountable. Lastly, I dedicate this work to the memory of my beloved grandmother, Pinfang Yao, with whom I spent my formative years, and to my paternal grandparents Yucai Nie and Jinglian Zhang. I hope that, wherever they are, they look upon this achievement with pride.

Acknowledgments

The research covered in this dissertation is funded by Cyber Florida and Qsancus Inc. The completion of this dissertation is funded by the Dissertation Completion Fellowship from the Office of Graduate Studies at the University of South Florida.

Table of Contents

List of Tables

List of Figures

Abstract

This dissertation focuses on addressing the technical challenges of non-stationarity in smart factories through the use of cyber-physical AI agents. Industry 4.0 and smart manufacturing with smart factories as a central role, have a growing demand for Just-in-Time (JIT) and on-demand production, as well as mass customization—all while maintaining high productivity, resource efficiency and resilience. This research positions Multi-Robot Systems (MRS)-driven smart factories. The heterogeneous production and transportation robots in an MRS collaborate to form multiple real-time adjusted production flows achieving the flexibility to accommodate such on-demand, mass customization.

However, the implementation of MRS introduces new sets of challenges, including the need for a coordination mechanism with superior agility, productivity, and energy efficiency. Further complexities arise in ensuring system resilience under unpredictable production demands and cyber-physical errors, which contribute to the non-stationary nature of smart factories. To address these issues, this dissertation proposes the cyber-physical AI agent system approach. This approach proposed is based on autonomous Artificial Intelligence (AI) agents that emphasize knowledge-based rationality, and AI cognitions including reasoning, prediction, and planning. Additionally, the AI agents learn to adapt to non-stationarity through data-driven computations while interacting with the environment.

This dissertation offers both theoretical and applicational contributions to the electrical engineering field. On the theoretical side, it solidates knowledge-based problem-solving of AI research, using graph models as domain models for the AI cognitions. More importantly, when multiple AI agents unite into a system, with wireless information exchange and social learning, the networked AI agents achieve collaboration toward collective objectives through communications. On the practical side, this work addresses real-world challenges in

smart factories, including real-time Multi-Robot Task Allocation (MRTA), resilient operation of production robots in the presence of physical errors, predictive path coordination for transportation robots, and maintaining the operational integrity of MRS in the presence of cyber-physical errors.

Chapter 1: Introduction

Wireless networked Multi-Robot Systems (MRSs) are rapidly emerging as a center role in smart factories, Industry 4.0, smart manufacturing, lean manufacturing, and future generations of manufacturing paradigms. The inherent qualities of MRSs align well with the objectives of these paradigms, which include enhanced productivity, energy efficiency, agility, flexibility, and sustainability [104, 88, 19, 25, 111, 96].

The application and expansion of these systems have been largely driven by Artificial intelligence (AI) as well as advances in other key technologies. Edge computing [110, 109] that performs system-wise decision-making with complex computation including production and transportation task scheduling and assignment, predictive maintenance is where central AI is located. Distributed AI computing and mobile AI computing [19, 14, 33] enables MRS to make real-time collaborative coordination and adjustments in task executions; Cyber-Physical Systems (CPS) [33] provide a framework for the integration of both types of AI computing with physical processes, allowing the application of data-driven Machine Learning (ML) techniques; Information Communication Technologies (ICT) [19] and wireless communication and networks [25, 20] facilitate data-driven AI decision-making and communication between smart factory components. When implemented within the framework of a smart factory, wireless networked MRS can handle heterogeneous process types, facilitate multiple real-time adjusted production flows, and support dynamic temporal-spatial collaborations. These functionalities allow them to effectively cater to Just-in-Time (JIT) or on-demand and mass-customized production.

1.1 Industry 4.0: The Emergence of the Smart Factory

The history of industrialization can be traced back to the late 18th century with the emergence of the First Industrial Revolution, marked by the mechanization of textile production and the application of steam power. This era significantly amplified the productivity of individual workers and laid the foundational framework for modern manufacturing. However, at this stage, the concept of automation and intelligent systems was almost entirely absent; labor was still predominantly manual, and machine assistance was rudimentary.

The subsequent industrial phases, characterized by the proliferation of electricity and the rise of mass production techniques, increased efficiencies but did not significantly diversify the capabilities of industrial systems. The Third Industrial Revolution, fueled by advancements in computer technology and automation, began to show the potential for intelligence within the manufacturing sector. Yet, despite these developments, the scope was primarily limited to single, autonomous units, fixed automated assembly lines or simple networked systems that executed repetitive, predefined tasks.

Fast forward to the 21st century, the term "Industry 4.0" incorporates the latest transformation in manufacturing, characterized by the seamless integration of physical and digital technologies. Within the paradigm of Industry 4.0, Smart Factories emerge as centers of an ecosystem connected not just by machinery but by data and analytics. These systems go beyond isolated tasks to incorporate adaptability, scalability, and learning into the production cycle. It is here that Wireless Networked Multi-Intelligent Robot Systems play a critical role in augmenting the abilities of a Smart Factory.

The automated manufacturing systems in Industry 3.0 were primarily engineered toward singular tasks. Robots on an assembly line could weld car doors or paint exteriors, but they couldn't adapt or switch between tasks efficiently. They lacked the intelligence to coordinate with other machines, adapt to new manufacturing configurations, or respond in real time to external changes.

Wireless Networked Multi-Intelligent Robot Systems represent extensive changes in this regard. By enabling high levels of coordination, these systems make it possible to have multiple robots work in unison, each aware of the other's position, status, and function. This level of integration provides a new range of flexibility and efficiency elaborated as follows that was unimaginable in earlier industrial stages.

- Scalability: MRS can be easily scaled up or down to adapt to the demands of production. Additional robots can be seamlessly integrated into the industrial wireless network as required.

- Adaptability: These robots can switch roles or tasks in real time based on the demands of the manufacturing process. This is especially crucial for custom manufacturing and just-in-time or on-demand productions.

- Optimization: AI algorithms bring maximized productivity, increased overall efficiency and minimized downtime through optimized task scheduling and assignments.

- Resilience: The system is less vulnerable to breakdowns; if one robot fails, tasks can be dynamically reallocated among the remaining robots so that MRS remains operating under errors.

- Data-Driven: Continuous data collection from MRS's interaction with the production environment and analytics with Machine Learning (ML) techniques make these systems self-improving, allowing them to adapt and become more efficient over time.

- Machine Intelligent Collaboration: Robots not only communicate with each other but can also make collaborative decisions toward system-wide and collective objectives.

Therefore, Wireless Networked Multi-Intelligent Robot Systems offer capabilities that dramatically break through the limitations of traditional industrial setups. By offering unprecedented levels of scalability, adaptability, and intelligence, these systems are defining the future landscape of smart factories and pushing the boundaries of new possibilities in the modern industry.

1.2 Research Motivation, Objectives and Organization

Building on the foundation set by Section 1.1, the emerging generation of the industry, inclusive of smart factories, is evolving to autonomously accommodate fluctuating production demands. This eliminates the need for product-specific automation programming.

This research introduces a method that employs a heterogeneous and collaborative Multi-Robot System (MRS) to achieve this objective. This approach aligns with the discrete production paradigm, as discussed in recent literature [132, 38].

Discrete production pertains to the manufacturing of distinct items that are countable and tangible. This mode of production revolves around assembling individual components or parts to produce complete goods. It stands in contrast to process production, characterized by continuous flows or batches, common in the creation of chemicals or beverages. For instance, products from discrete manufacturing include automobiles, furniture, airplanes, toys, smartphones, and defense systems. These items are distinguishable by their ability to be deconstructed into individual components and reassembled.

Tasks within the discrete production realm often encompass actions such as cutting, drilling, or assembling to yield the final products. Given the unique nature of each product, discrete manufacturing processes are generally more adaptable than their continuous counterparts. Such adaptability allows manufacturers to seamlessly transition between different products or tailor products in response to market demand.

Employing a heterogeneous, collaborative Multi-Robot System (MRS) comprised of both production and transportation robots, each robot type can be dedicated to specific production processes. By assigning them time-varying tasks that pinpoint the necessary processes, the MRS is capable of establishing multiple, real-time adjusted production flows. This setup efficiently meets the Just-In-Time (JIT) and mass-customized production demands, eliminating the need for product-specific programming.

In the context of a smart factory, this efficiency is realized through AI computations on a centralized edge server paired with a network of interconnected robots. These robots fetch task assignments directly from the edge computing system. Based on these assignments, they self-reconfigure, setting up new, tailored production flows as needed.

The implementation of MRS within smart factories presents a myriad of challenges, several of which remain unaddressed in current literature:

- *Productivity and Energy Efficiency*: A fully automated MRS is anticipated to surpass traditional production lines in both productivity and energy efficiency. While the integration of Machine Learning (ML) techniques and AI computing at both the edge and robotic levels is expected to autonomously optimize these performance metrics, this remains an unresolved issue in current research.

- *Uncertainty in JIT Production*: By nature, JIT production isn't wholly predictable, introducing non-stationarities to MRS operations. This unpredictability challenges prevalent ML techniques in literature, particularly those rooted in statistical assumptions.

- *Operational Resilience of MRS*: A novel aspect of smart factories is the operational resilience of MRS. Enhanced by ML and AI applications, MRS can continuously function even amidst cyber-physical errors. In contrast, previous industrial generations required scheduled maintenance and calibrations, which inevitably interrupted operations. Moreover, despite decentralized functioning, point failures within a smart factory's MRS can be identified and localized, allowing for predictive maintenance without halting production flows. This resilience, integral to modern smart factories, has yet to be thoroughly explored in academic literature.

- *Data-Driven Approaches*: Cutting-edge ML techniques have ushered in data-driven methodologies. However, questions remain about the nature and methods of data collection from the MRS operations.

- *AI & ML in MRS Automation*: While AI and ML have been subjects of extensive research, the methodology to incorporate them effectively in automating MRS operations is still in its infancy. This leads us to the primary technical proposal of this research: the cyber-physical AI agent system approach. While AI and ML represent a toolkit of techniques, a cyber-physical AI agent actively engages with its surroundings across both digital and physical realms. These agents also collaborate with fellow robots, aiming to achieve individual and collective objectives.

The contribution of this lies in both the technical and practical aspects of employing Multi-Robot Systems (MRS) in a smart factory, aiming to identify and address new challenges including the need for an effective coordination mechanism with superior agility, productivity, and energy efficiency under the complex multi-robot interactions, as well as the operational resilience and integrity under unpredictable production demands and cyber-physical errors.

From a technical perspective, a fundamental principle of my research is the application of Artificial Intelligence (AI) and Machine Learning (ML) to cope with the complex challenges raised by MRS-driven smart factories. A cyber-physical AI agent system approach is proposed, which resides at the intersection of foundational AI research, as outlined by [89]. Furthermore, this research employs graph theory as an essential tool for modeling the MRS and its operations. This mathematical technique offers a robust framework to map out complex systems and their interactions, serving as a bridge to the fast adaptability of AI and ML solutions. This modeling approach is elaborated in Section 3.1.

The proposed approach integrates the strengths of the rational agent framework, as presented by [89], with augmented cybernetic attributes. This synthesis transforms robots into holistic entities capable of advanced communication, perception, reasoning, predictive planning, and decision-making informed by domain knowledge. The graph model serves as this domain knowledge, effectively acting as the "world model" posited by [58]. These innovations substantially enhance AI and ML efficiencies, such as learning sample efficiency, computational time, and accelerated learning. Moreover, this framework supports MRS interoperability and communication efficiency, as detailed in Section 3.2 and 3.3.

In practical terms, as an electrical engineer and researcher, my overarching aim is to actualize the theoretical aspirations of smart factories with MRS. I explore how the broader goals of smart factories—namely flexibility, agility, productivity, energy efficiency, scalability, and resilience—can be mapped onto specific MRS objectives. These include task assignment, collaborative task execution, maximized production yield, and the maintaining of these performance indices even under the presence of cyber-physical errors - resilient MRS operations. The detailed MRS modeling is proposed in Section 1.3.

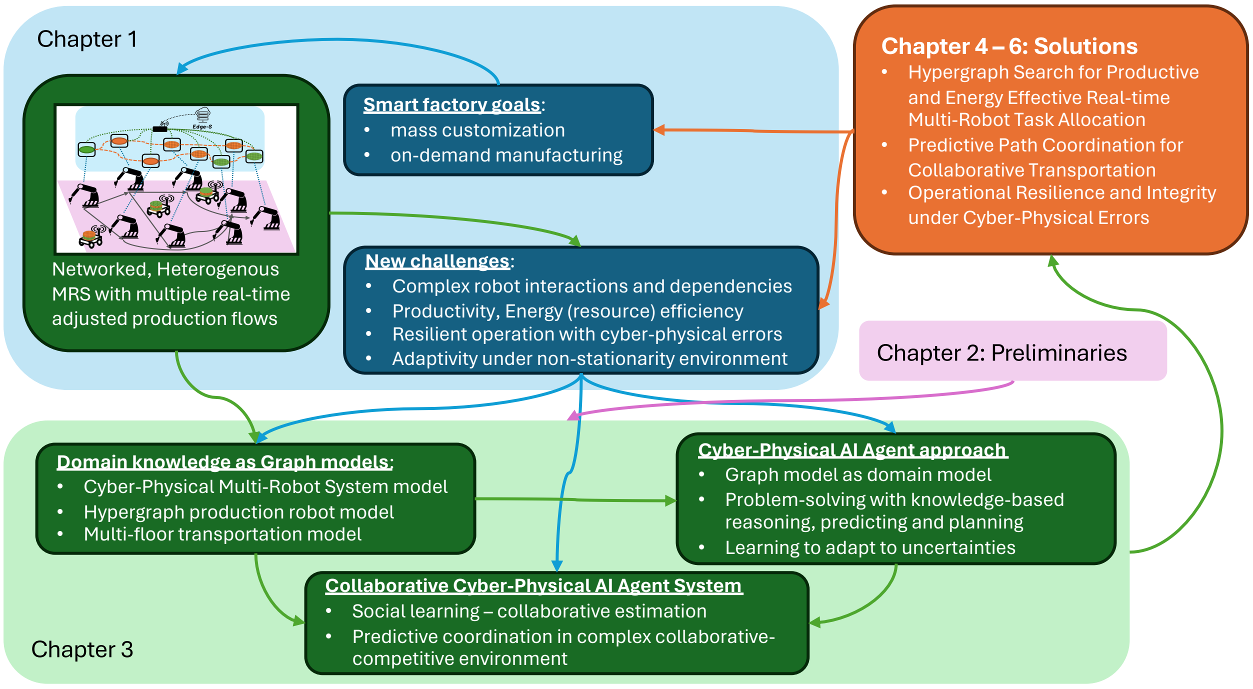

Chapter 1
Networked, Heterogenous MRS with multiple real-time adjusted production flows
Smart factory goals:
• mass customization
• on-demand manufacturing
New challenges:
• Complex robot interactions and dependencies
• Productivity, Energy (resource) efficiency
• Resilient operation with cyber-physical errors
• Adaptivity under non-stationarity environment
Chapter 4 – 6: Solutions
• Hypergraph Search for Productive and Energy Effective Real-time Multi-Robot Task Allocation
• Predictive Path Coordination for Collaborative Transportation
• Operational Resilience and Integrity under Cyber-Physical Errors
Chapter 2: Preliminaries
Domain knowledge as Graph models:
• Cyber-Physical Multi-Robot System model
• Hypergraph production robot model
• Multi-floor transportation model
Cyber-Physical AI Agent approach
• Graph model as domain model
• Problem-solving with knowledge-based reasoning, predicting and planning
• Learning to adapt to uncertainties
Collaborative Cyber-Physical AI Agent System
• Social learning – collaborative estimation
• Predictive coordination in complex collaborative-competitive environment
Chapter 3

Figure 1.1 The organization of the dissertation.

The organizational structure of the dissertation is outlined as follows.

In Chapter 1, with the previous sections' elaboration of smart factory goals, Section 1.3 aligns these goals with MRS and real-time adjusted multiple production flows. On top of the challenges summarized earlier in this section, Section 1.4 discusses the new technical challenges associated with implementing MRS in a smart factory and offers a comprehensive literature review. Chapter 3 delves into the networked AI agents approach, integrating domain models to facilitate intelligent decision-making to address the aforementioned challenges. This chapter also underscores the cyber-physical feature, which allows intelligent agents to improve their behavior through active interaction with their environment. Subsequent sections focus on specific methodologies: Section 3.1 explains how graph modeling serves as the foundational technique for mathematically representing domain knowledge; Section 3.2 proposes how a single AI agent utilizes both domain models and AI cognition for problem-solving; and Section 3.3 elaborates on how multi-AI agents collaborate through communications. Chapters 4 to 6 validate these methodological innovations with particular engineering problems, solutions, computational experiments and discussions. Figure 1.1 illustrates the above organization of the dissertation.

1.3 Multi-Robot System-Driven Smart Factories

Introduced by [19], [76], and [75], wireless networked Multi-Robot Systems (MRSs), which comprise both production and transportation robots, are conceptualized as Cyber-Physical Multi-Robot System (CPMRS) to be detailed in Section 3.1.1. As depicted in Figure 1.2, both the cyber and physical domains utilize the same set of production robots as nodes. However, edge computing is exclusive to the cyber domain. The cyber domain embodies a wireless network that integrates distributed robot computing with edge computing. In contrast, the physical domain represents a dynamic production flow network [126] that encompasses production robots, transportation robots, and raw materials.

In this configuration, edge computing serves as a central AI controller that dynamically adapts the Multi-Robot Systems (MRS) to meet fluctuating production demands. It does so through real-time task assignment and scheduling. These task assignments are centrally orchestrated by the real-time Multi-Robot Task Allocation (MRTA) algorithm. Both produc-

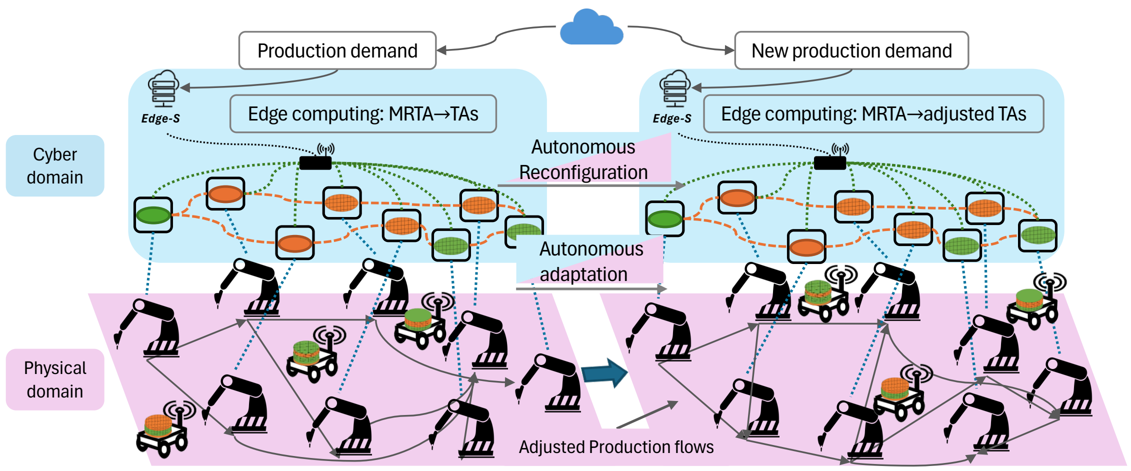


Figure 1.2 An MRS-driven smart factory with edge computing governing production robots and transportation robots to form dynamic production flows.

tion and transportation robots are recipients of these centrally assigned tasks. However, the actual execution of these tasks is done collaboratively and is decentralized, sometimes even distributed across the network of robots. Consequently, the MRS operates in a dual manner: decentralized in task execution and temporally collaborative to adapt to time-varying production flows.

The CPMRS model grants smart factories significant flexibility and agility. Each robot is uniquely equipped to execute specific tasks, allowing for frequent reconfiguration of production flows. Edge computing addresses dynamic, on-demand production requirements by calculating TAs in real time, which factors in productivity, resource efficiency, and resilience. Consequently, multiple time-varying production flows exit the physical domain, highlighting the unique position of edge computing in the cyber domain as it oversees and influences the physical domain.

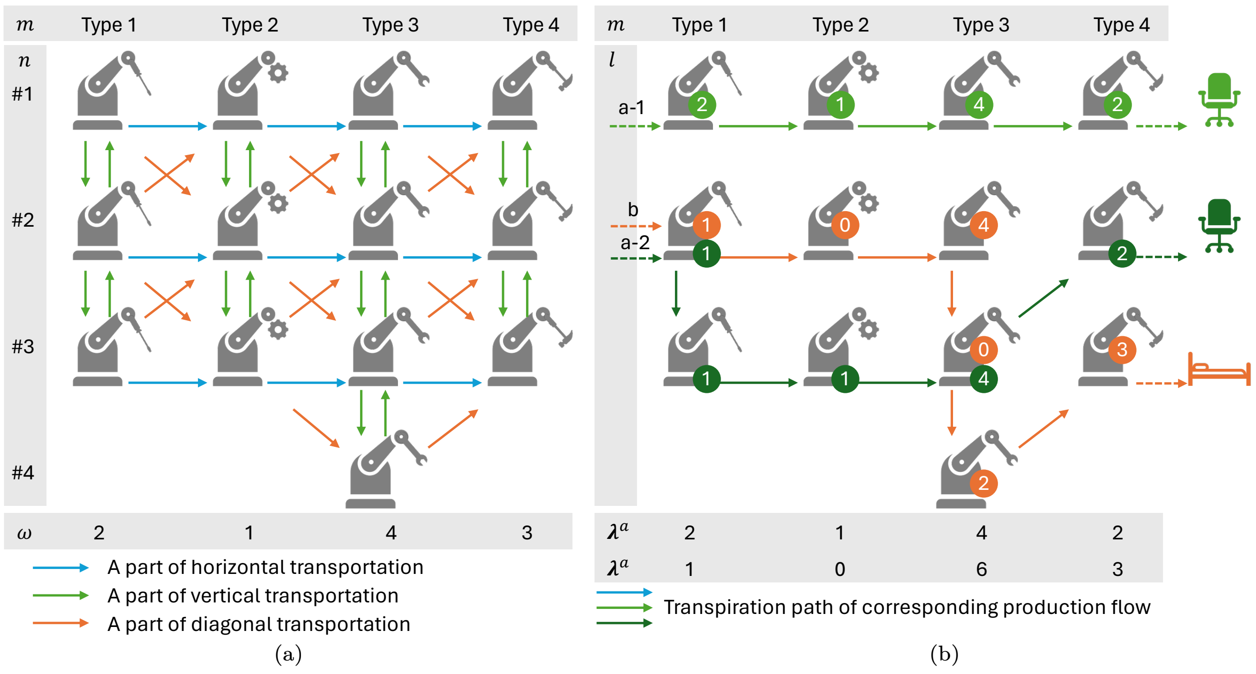
The concept of time slots is introduced to manage the task execution in Multi-Robot Systems (MRS). In MRS, each robot is allocated specific tasks and possesses unique capabilities, as well as differing multiplexing capacities. Production robots operate synchronously by leveraging their multiplexing capabilities. Despite the variability in time required for different production tasks, the synchronization is achieved by assigning specific multiplexing capacities to each robot [19]. As a result, within a single time slot, production robots can handle multiple production flows. This is referred to as executing one task per time slot with multiplexing capability. In contrast, transportation robots function asynchronously at the task assignment level. These robots are allocated new transportation tasks as they complete preceding ones, leading to continuous task execution [69]. Nonetheless, during the actual task execution, transportation robots move in discrete navigation steps that are synchronized. This synchronous movement allows for real-time adjustments, facilitating collision-free navigation paths.

An MRS that combines production and transportation robots constitutes a heterogeneous system, where the robots cooperate in specific robot-specific tasks and collaborate in the overall production flows. Edge computing assigns each production or transportation task to a corresponding production or transportation robot. In the physical domain, these robots collaborate in a temporal-spatial manner: a production robot completes a task, producing a semi-finished product, which a transportation robot then conveys to the next production robot for further processing, and the cycle continues. Simultaneously, in the cyber domain, the robots collaborate to make real-time adjustments to their task executions, optimizing for productivity.

1.3.1 Production Robots for Customized Production Demands

Given a production demand that defines the product type, task type, and the number of tasks needed to fabricate a product [30], edge computing is employed to execute the MRTA (Multi-Robot Task Allocation) algorithm. This process generates task assignments and then transmits reconfiguration directives, coupled with these assignments, to each robot via wireless communications.

As demonstrated in Figure 1.3, this research assumes a heterogeneous MRS, meaning that it involves multiple types of robots, with each type exclusively dedicated to a specific production task. Practical production flows align with the task assignments provided to



m
Type 1
Type 2
Type 3
Type 4
n
#1
#2
#3
#4
ω
2
1
4
3
A part of horizontal transportation
A part of vertical transportation
A part of diagonal transportation
(a)
l
a-1
b
a-2
λ^a
2
1
4
2
λ^a
1
0
6
3
Transpiration path of corresponding production flow
(b)

Figure 1.3 Temporal-spatial production robot model.

fixed-location production robots. The subsequent paths, indicated by arrows leading to transportation robots, form a temporal-spatial operational model.

As depicted in Figure 1.3, an MRS comprising M types of production robots can be represented by the tuple $(M, \mathbf{N}, \boldsymbol{\omega}, \varepsilon_h, \varepsilon_v, \varepsilon_d)$.

Here:

- $\{\mathbf{N}\} = (N_1, N_2, \dots, N_m, \dots, N_M)$ signifies that N_m type-m robots are positioned and aligned at the top of the mth column, dedicated to type-m tasks.

- $\boldsymbol{\omega} = (\omega_1, \omega_2, \dots, \omega_m, \dots, \omega_M)$ denotes the multiplexing capability of a production robot. Specifically, type-m production robots can execute ω_m type-m tasks concurrently within a time slot, hinting at Multi-Task robots [35].

- Transportation is constrained to spatially adjacent production robot neighbors, whether they are in columns, rows, or diagonals [41, 78]. The respective energy consumptions for these transports are quantified by ε_h, ε_v, and ε_d.

This setup distinguishes the spatial characteristics of the MRS based on the placements of production robots, transportation constraints, and quantifications of energy consumption. The temporal characteristic, on the other hand, is differentiated by the robot's multiplexing capabilities within a time slot.

Each production robot is identified by the label $R_{m,n}$ where $m = 1, 2, \dots, M$ and $n = 1, 2, \dots, N_m$.

Figure 1.3 showcases an MRS characterized by the following tuple, where the three unique colored arrows distinguish the three types of transportation.

$$(M = 4, \mathbf{N} = (3, 3, 4, 3), \boldsymbol{\omega} = (2, 1, 4, 3), \varepsilon_h, \varepsilon_v, \varepsilon_d)$$

As illustrated in Figure 1.3, the production demands are represented by the vector set $\{\boldsymbol{\lambda}^l\}$, where l specifies the product type. Given the established relationship between the type of a production robot and the type of production task, manufacturing product-l can be decomposed into a sequence of production tasks. These tasks are executed by the production robots in ascending order and can be denoted by the vector $\boldsymbol{\lambda}^l = (\lambda_1^l, \lambda_2^l, \dots, \lambda_m^l, \dots, \lambda_M^l)$. Each element in $\boldsymbol{\lambda}^l$ indicates the number of corresponding tasks needed to complete product-l.

It's worth noting that while product-l might not necessitate every type of task for its production, any given λ_m^l can be zero. However, the aggregate of all M elements in $\boldsymbol{\lambda}^l$, represented as $\sum_{m=1}^{M} \lambda_m^l$, should exceed 0 to be valid.

Figure 1.3 presents the same MRS as seen previously with production demands given by $\{\boldsymbol{\lambda}^l\} = \{(2, 1, 4, 2), (1, 0, 6, 3)\}$. This illustration also includes a feasible, albeit non-optimal, task assignment showcasing two chair-production flows (denoted as $a(1)$ and $a(2)$) and a single bed-production flow (represented as b). A production flow is visualized using a sequence of arrows, indicating transportation, and circled numbers, indicating task execution. Different product types and production flows are differentiated using colors and shades, respectively.

Interestingly, while both $a(1)$ and $a(2)$ meet the production demand $\boldsymbol{\lambda}^a = (2, 1, 4, 3)$, they differ in terms of the number of production robots used and the transportation pathways. Specifically, $a(2)$ employs one additional production robot and an extra transportation segment, necessitating an additional time slot for production. This makes $a(1)$ a more optimized production compared to $a(2)$. The flexibility inherent in the temporal-spatial MRS model is evident from the involvement of certain production robots in multiple production flows (e.g., $R_{3,3}$) and the segmented transportation paths based on layout.

The evolution of manufacturing processes, as outlined in the model, illustrates a shift from static flows in traditional automated manufacturing systems to a flexible, temporal, and sequential task execution process within MRS. The MRTA problem, as presented in this study, aims to satisfy the production demands $\{\boldsymbol{\lambda}^l\}$ by allocating production tasks to individual production robots $R_{m,n}$ while adhering to task sequence constraints when determining transportation paths.

Given the intricate dynamics of real-time application scenarios, the diversity of robot and product types, the capabilities of multi-task robots [35], and the interwoven temporal constraints, this unique MRTA problem can be categorized under "Complex Dependencies [MT-SR-IA]" as described by [55]. Consequently, this categorization underscores the need for a nuanced approach to effectively solve and optimize this problem, bearing in mind time complexity considerations.

1.3.2 Transportation Robots Under Constrained Operation

When employing transportation robots to advance industrial autonomous transportation in a smart factory the autonomous navigation of transportation-MRS comprising autonomous mobile robots (AMRs) is formulated as discrete navigation facilitated by decentralized AI decision-making [3, 108]. As depicted in In Figure 1.2-(a) and (b), edge computing in a smart factory performs real-time multi-robot task allocation (MRTA) [76] and dynamically assigns production tasks to production-MRS according to production demands, which leads to dynamic transportation task requirements to transportation-MRS. It allows the production flows to change from a and b to c and d between time t_a and t_b, which is known as an MRS reconfiguration [76]. Transportation task changes from MRS reconfigurations cannot be reasonably predicted by an AMR since they involve executive-level decisions and customer demands [116]. Potential collisions will happen when multiple AMRs are located at the same location and intend to take the same navigation step, which can be resolved or avoided in advance with the assistance of discrete navigation and thus referred to as potential collisions. Consequently, 11 autonomous mobile robots (AMRs) in Figure 1.2-(a) and (b) are assigned distinct tasks in a real-time, end-to-end manner. Therefore, breaking down the navigation paths of AMRs into a series of smaller, discrete steps allows for more adaptable, flexible, and efficient coordination [23, 67], as the AMRs can quickly adjust to avoid potential collisions by re-routing their paths in real-time.

A transportation-MRS flexibly accomplishes transportation tasks by path planning capable of making real-time adjustments according to the requirement of production-MRS, to achieve agile and flexible production in a smart factory. Contrasting to conventional industrial autonomous transportation, techniques such as conveyor belts, assembly lines, and Autonomous Guided Vehicles (AGVs) are designed to handle static production flows. Once such systems are tuned to satisfy efficiency and consistency, they lose the adaptability to dynamic changes in production demands [46]. Moreover, unlike warehouse logistics with AMRs, transportation tasks in smart factories do not begin or end at a single fixed location and have precise due time because they are parts of production flows [7]. Literature such as [1] and [122] formulate smart factory transportation as multi-path scheduling that optimizes the delay and energy consumption. Yet, they cannot address the dynamic production flows with the interaction of two MRSs. Further, [68] and [134] formulate the path following control for industrial AGVs but cannot address the path constraints from transportation tasks and thus are inapplicable for smart factories.

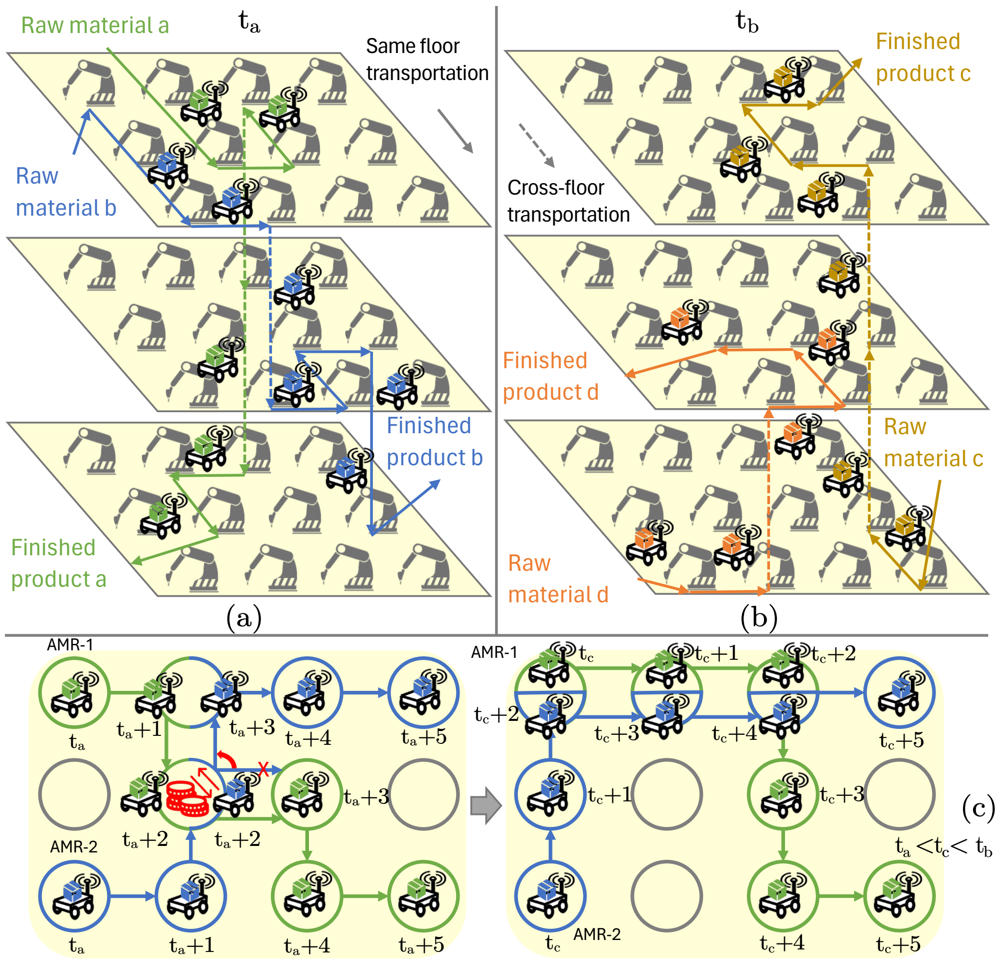


Figure 1.4 The smart factory transportation is formulated as discrete navigations in a multi-floor model.

1.4 Technical Challenges and Literature Review

The challenges of employing multi-robot systems in a smart factory are identified in this Section. First, superior flexibility brings complexity. While ideally, a multi-robot system can execute time-varying tasks to form multiple real-time adjusted production flows, classical task allocations or scheduling methods face the challenge of large complexity from heterogeneity and inter-dependencies. Second, such complexity also leads to decentralized task execution for the sake of scalability in terms of the number of robots. Such decentralization brings new challenges of collaboration. Third, the cyber-physical nature of the

AI computing-enabled multi-robot systems brings new challenges of operational resilience, integrity and privacy preservation.

1.4.1 Complexity in Coordination

Section 1.3 introduces the flexible heterogeneous MRS to account for customized, time-varying production demands. However, it introduces multiple dimensions of complexity in coordination.

Firstly, the MRS under consideration is heterogeneous, comprising various types of both production and transportation robots. This diversity necessitates addressing the coordination challenges posed by the Multi-Robot Task Allocation (MRTA) problem, with an emphasis on sequential coordination to meet the specific production processes.

Historically, MRTA has been a critical theoretical facet of multi-robot coordination. Much of the existing research on MRTA has been centered around static optimization objectives, particularly under assumptions of task independence which negate the imposition of ordering constraints on tasks [35]. Traditional approaches to MRTA have framed it as a combinatorial optimization problem. Consequently, solutions have ranged from linear programming techniques, exhaustive search algorithms [29], genetic algorithms [2], to distributed auction algorithms [44]. While these methods have demonstrated efficacy and optimality for their intended objectives, they fall short when applied to smart factories. The inherent need for task sequencing in production flows and real-time adaptability render static MRTA suboptimal for such contexts.

Secondly, with the production demands being customized and time-varying, MRTA must be adaptive enough to reconfigure multiple production flows. There have been attempts to enhance the Multi-Robot Task Allocation (MRTA) taxonomy, taking into account the interconnected utilities and constraints. This enhanced perspective posits that MRTA encompasses the intertwined challenges of task decomposition, task allocation, and scheduling [55]. This viewpoint aligns more congruently with the dynamics of smart factory settings.

[105] employs a graph-partitioning technique, aiming to optimize temporal efficiency, factoring in spatial conflicts and computational time constraints. Meanwhile, [106] leverages genetic algorithms to address temporal sequencing of tasks and to ensure collision-free path planning. Further, [70] introduces a distributed multi-agent system, wherein agents engage through a negotiation protocol, optimizing both localized and overarching objectives.

While the aforementioned studies address time and space constraints in various capacities, none offer a holistic solution tailored for smart factories. Such a solution would encompass dynamic production demands, flexible production flows, energy efficiency, and robust coordination—the central concerns of the present study.

Thirdly, MRTA is required to provide dual-agent, dual-objective, comprehensive optimization that focuses on both production robot productivity and transportation robot energy efficiency. These optimizations should be considered in real-time application contexts, factoring in the intricacies of wireless communication dynamics [19, 22].

[112] puts forward a dynamic multi-agent-based real-time task allocation strategy, drawing on a bargaining-game-based negotiation mechanism. However, this approach is tailored for a job-shop environment, rendering it inadequate for managing flexible production flows.

On the other hand, [26] presents a computationally effective stochastic conflict-based allocation strategy that considers both temporal and spatial constraints. Yet, its reliance on a single-agent policy tree search fails to meet the system-level performance optimization that smart factories necessitate.

Meanwhile, [49] proposes a robust, negotiation-based distributed dynamic scheduling mechanism for a multi-agent system. However, its robustness, as a feature, is not thoroughly addressed or convincingly demonstrated.

Given the integration of edge computing and the diverse manufacturing dynamics across sectors, it's crucial to acknowledge that there isn't a definitive balance between optimality and computational time. Moreover, for optimal task assignments to be viable, their computation must be achieved in real time. This ensures that computational delays don't threaten the stable functioning of the system. Addressing these manifold challenges is the primary objective of the present study.

1.4.2 Collaborative Task Execution Under Decentralization

In the MRS-driven smart factory, which is discussed in Section 1.3, there is a unique blend of centralized task assignments through edge computing and decentralized task executions. While this decentralization allows for distributed activities across the robot network, the system still upholds temporal collaboration to fulfill larger goals. However, this structural duality introduces several challenges, especially when ensuring coordination between production and transportation robots. This has sparked the need for a new multi-intelligent robot system architecture.

First, adaptation to dynamic production flows. The smart factories of today require real-time adjustments to dynamic production flows. Production robots often switch between working concurrently and sequentially with different robot groups, catering to customized, on-demand production needs. Traditional deterministic control or optimization methods fall short in these dynamic environments [10, 16]. Current research primarily either hones in on deterministic optimization strategies [74, 82] or ignores the dynamic aspect of production flows entirely [13, 136]. A robust MRS should address several key areas including handling dynamic production flows, maintaining scalable wireless communications with minimal redundancy, regulating information exchange, providing accurate estimations, and fostering inter-robot collaboration during tasks. Hence, an adaptive coordination mechanism is crucial for these time-sensitive environments.

Second, edge computing and dynamic task assignments. Edge computing's role is pivotal in moderating the interactions between the production and transportation sectors of the MRS. The nature of dynamic production flows can result in unexpected task assignment changes. Therefore, the MRS must collectively make decisions on-the-fly to maintain consistent performance. Though the Markov Decision Process (MDP) is renowned for modeling sequential decision challenges, its intricacy rises with the inclusion of more robots. Traditional multi-dimensional MDPs struggle with variable task durations and multiple conflicting goals, leading to visibility issues. Several studies, like [81, 91, 7], employ MDPs for discrete navigation but don't thoroughly optimize transportation task performance.

Third, challenges of decentralized task execution. With decentralized task execution, individual robots often work without regard to their peers' activities. This can be problematic, given the importance of temporal collaboration. For production robots, the diverse range of their abilities further complicates matters, making it a challenge to anticipate other robots' actions. Transportation robots face their own set of issues. With vast shop floors, a multitude of optimal or near-optimal paths are available. Predicting paths for all Autonomous Mobile Robots (AMRs) becomes unfeasible, especially in multi-floor layouts, given the computational restrictions. Some existing studies, such as [24] and [125], tackle multiagent pickup and delivery challenges without the advantage of agent-based modeling. This results in exponential growth in time complexity as the number of AMRs increases, complicating the optimization process.

1.4.3 Reliability, Resilience and Privacy Preservation

As indicated by existing literature such as [119], as well as industry reports, the primary concerns in the implementation of smart factories are reliability, resilience, and privacy. The concept of reliability in this context refers to the ability of wireless networked multi-intelligent robot systems to maintain operations even when errors occur, either in the physical or cyber domain. Resilience pertains to the system's capacity for error detection and correction, ensuring that the performance of the Multi-Robot System (MRS) can be preserved and restored. Lastly, privacy preservation is concerned with securing the data and models used in Machine Learning (ML) and Artificial Intelligence (AI) techniques, protecting them from unauthorized access.

Particularly for production robots, estimating accuracy degradation poses a challenge. This is due to the lack of immediate feedback from production flows and the global heterogeneity inherent to MRS. Requiring immediate feedback from downstream robots to upstream robots within every production flow imposes NP-hardness on wireless networked MRS [94]. Moreover, because MRS are tasked with a variety of production functions, they experience differing rates and directions of accuracy degradation [36], which are not readily known to the robots themselves. This heterogeneity creates significant technical challenges for smart factories employing diverse MRS. Errors in task execution propagate along the production flows, undermining the effectiveness of directly measuring unfinished products to gauge a robot's accuracy. Similarly, relying on edge computing to evaluate the quality of finished products is insufficient for accurately assessing individual robot performance. Even general stochastic optimization approaches, as discussed in [16] and [36], prove inadequate for such coordination complexities. Thus, an intelligent MRS should collaboratively and decentrally estimate the accuracy of peer production robots. This estimation can be accomplished through localized direct measurements and the exchange of information among the robots within the MRS.

Moreover, even when MRS employs on-device machine learning techniques like Federated Learning (FL) to circumvent the need for explicit data transmission, privacy concerns remain. For instance, model inversion attacks can compromise privacy [133]. Attackers can intercept the confidential data collected for local model updates by sniffing wirelessly transmitted models. Once intercepted, these models can be used to predict original input data, thus breaching privacy [47]. While unauthorized access to these models is a prerequisite for such attacks, defending against them is challenging due to the difficulty in obfuscating the models' predictions [40]. This makes the encryption of both global and local FL models crucial for preserving privacy during model exchanges.

Additionally, the operational integrity of MRS-driven smart factories faces threats from cyber attacks such as Sybil attacks [63]. In such attacks, fictitious clients are created to send malicious updates to edge nodes. These malicious ML updates are then aggregated with updates from legitimate clients, resulting in a compromised global model [48]. Consequently, Sybil attacks can effectively compromise FL operations even without gaining direct control over the robots or edge nodes. When robots execute tasks using compromised models, it not only jeopardizes the MRS operation but also introduces cyber errors that disrupt the MRS. Further complications arise from inaccurate task execution parameters, which can lead to defects in the finished products. Given that errors in flow-based manufacturing are cumulative and propagate through the production line, identifying a compromised robot—or even the spurious devices generating the malicious updates—becomes an immense challenge.

Chapter 2: Preliminaries

This chapter serves as an introduction to the basis of knowledge that underpins the dissertation. These concepts are not original contributions but are essential to the research conducted. The chapter is divided into three primary sections: smart factory task environments, problem-solving graphical algorithms, and stochastic optimization coupled with machine learning.

The first section delves into the task environment of smart factories. It outlines the multidisciplinary considerations needed for modeling Multi-Robot System (MRS)-driven smart factories, incorporating elements from robotics, automation, and industrial engineering.

The second section focuses on problem-solving graphical algorithms. These algorithms were at the forefront of AI before the rise of machine learning and deep learning technologies. The insights gained from these algorithms continue to offer valuable perspectives on problem-solving within AI.

The third section addresses stochastic optimization and machine learning. These are core theories that underlie data-driven and general-model-based learning approaches. Their relevance extends to a wide array of applications, including but not limited to, robotics and factory automation.

2.1 Smart Factory Task Environment

Task environments are essentially the "problems" to which rational agents are the "solutions". Here are some of the key factors that need to be considered in the task environment of an MRS-driven smart factory in terms of performance, environment, actuators, and sensors [89].

The performance of an MRS in a smart factory is determined by a number of factors, including the number of robots, the capabilities of the robots, the complexity of the tasks, and the environment in which the robots are operating. In general, more robots will lead to better performance, as they can collaborate to complete tasks more quickly and efficiently. However, the additional robots also add complexity and cost to the system. The capabilities of the robots are also important, as they must be able to perform the tasks required of them. For example, if the robots are required to lift heavy objects, they will need to be equipped with strong actuators. The complexity of the tasks also affects the performance of the system. Simple tasks can be easily completed by a single robot, but more complex tasks may require the cooperation of multiple robots. The environment in which the robots are operating also plays a role in their performance. For example, robots that are operating in a dusty environment will need to be equipped with sensors that can detect and avoid obstacles.

The environment in which a MRS is operating can have a significant impact on its performance. The environment can affect the robots' ability to move around, the accuracy of their sensors, and the reliability of their communication systems. Some of the factors that need to be considered when designing a MRS for a particular environment include:

- The size and layout of the environment [130]: The robots need to be able to move around the environment without colliding with each other or with obstacles.

- The cyber-physical error [80]: In the physical domain, the accuracy and precision of actuators and sensors of the robots degrade over time, which can lead to defects; in the cyber domain, the correctness of cyber operations can be compromised because of bugs or even malicious attacks.

- The presence of other robots or humans [90]: The robots need to be able to safely operate in an environment that is shared with other robots or humans. Actuators

Actuators are devices that convert energy into motion. They are used to move the robots around and to perform tasks such as picking and placing objects. The type of actuators that are used in an MRS depends on the capabilities of the robots and the tasks that they are required to perform. Some of the common types of actuators used in MRSs include:

- Electric motors [52]: Electric motors are the most common type of actuator used in robots. They are relatively inexpensive and reliable, and they can be used to power a variety of movements.

- Hydraulic actuators [77]: Hydraulic actuators are more powerful than electric motors, but they are also more expensive and complex. They are often used in robots that require a lot of force, such as robots that are used to lift heavy objects.

- Pneumatic actuators [5]: Pneumatic actuators are powered by compressed air. They are less powerful than hydraulic actuators, but they are also lighter and less expensive. They are often used in robots that need to be lightweight, such as robots that are used in space exploration.

Sensors are devices that detect and measure physical quantities such as the position, velocity, and acceleration of the robots. They are also used to detect obstacles and other objects in the environment. The type of sensors that are used in a MRS depends on the tasks that the robots are required to perform. Some of the common types of sensors used in MRSs include:

- Cameras [93]: Cameras are used to detect objects and to track the position and movement of the robots.

- LiDAR sensors [45]: LiDAR sensors use laser light to measure the distance to objects. They are used to create a 3D map of the environment, which can be used by the robots to plan their movements.

- Proximity sensors [52]: Proximity sensors detect objects that are close to the robots. They are used to prevent the robots from colliding with obstacles.

- Inertial measurement units (IMUs) [9]: IMUs measure the orientation and acceleration of the robots. They are used to track the robots' movements and to maintain their balance.

The task environment of a MRS-driven smart factory is a complex and challenging one. The factors that affect the performance of the system are numerous and varied. By carefully considering all of these factors, it is possible to design a system that is capable of performing tasks efficiently and effectively.

2.2 Problem-solving Graphical Algorithms

Artificial Intelligence (AI) represents an interdisciplinary field at the intersection of computer science, mathematics, and engineering, with the aim to develop intelligent agents capable of performing tasks that would otherwise require human intelligence. A core area of focus within AI is problem-solving, which encompasses methods for searching through a large space of potential solutions to find one that meets the specified objective. This section focuses on four fundamental graphical algorithms that I employed throughout my research: shortest path search, random walks, maximum flow heuristic and constraint satisfaction problem.

2.2.1 Shortest Path Search

A graph G is defined as $G = (V, E)$, where V is a set of vertices (or nodes) and E is a set of edges (or links) connecting these vertices. Each edge e may have an associated weight ε. The concepts of *simple path*, *path* and *walk* are important to interpret this dissertation. A walk is a sequence of nodes and links in a graph. The links in a walk can be repeated, and the nodes can be repeated, but the first and last nodes must be different. A transportation robot's task execution is a walk since it might include repeated nodes and links but the first and last nodes are different. A path is a walk in which no vertex is repeated. This means that no links can be repeated either since an edge connects two nodes. A simple path is a path in which the nodes are also all distinct. This means that no vertex appears more than once in the path. The production flow of a smart factory is a simple path since it does not include repeated nodes and links.

Here are some examples of walks, paths, and simple paths:

- The sequence (A, B, C, B, A) is a walk, but it is not a path because the vertex B appears twice.
- The sequence (A, B, C, D) is a path, but it is not a simple path because the vertex A appears twice.
- The sequence (A, B, C, D, E) is a simple path.

A shortest path search algorithm is a method for finding the shortest path between two nodes in a graph. The graph can represent a physical space, such as a city map, or a more abstract space, such as a set of possible states in a problem-solving domain. Some well-known shortest-path search algorithms are introduced as follows

Breadth-first search (BFS) explores all the nodes at the present depth before moving on to the nodes at the next level of depth. It is generally implemented using a queue. BFS is complete and guarantees a shortest-path solution when all edge costs are uniform. Depth-first search (DFS) explores as far as possible along each branch before backtracking. Implemented using a stack, DFS is not guaranteed to find the shortest path. It is, however, memory-efficient compared to BFS. Dijkstra's algorithm finds the shortest path from a source vertex to all other vertices in a weighted graph. It is complete and optimal but cannot handle graphs with negative edge weights. A* (A-star) algorithm is an informed search algorithm that uses a heuristic to estimate the cost from the current node to the goal. This allows A*

Table 2.1 Comparison of shortest path algorithms

Algorithm	Completeness	Optimality	Time Complexity	Space Complexity
BFS	Yes	Yes	$O(V+E)$	High
DFS	Yes	No	$O(V+E)$	Low
Dijkstra's	Yes	Yes	$O(V \log V + E)$	Moderate
A*	Yes	Yes	$O(V \log V)$	Moderate

to search in the direction of the goal, thereby often achieving faster solutions than BFS or Dijkstra's Algorithm.

2.2.2 Random Walks

Random Walk algorithms, particularly in the domain of graph-based models, have gained significant importance in various fields like network analysis, machine learning, and artificial intelligence [121, 129]. One of the well-known and employed algorithms in this dissertation is *Node2Vec* [37], which utilizes random walks to create feature representations for nodes in a graph. The aim is to generate embeddings that could be useful for a variety of graph analysis tasks, such as classification, clustering, and link prediction.

In graph theory, a random walk is a stochastic process comprising movements from one node to another in a graph, generally with equal probability. A simple random walk on a graph $G = (V, E)$ could start from an initial node v and move to a neighboring node chosen at random. This walk can continue indefinitely or stop after a set number of steps.

Node2Vec extends the generic Skip-Gram model in Natural Language Processing (NLP) to the realm of graphs. It considers each random walk as a sentence and each node as a word within that sentence. The algorithm aims to optimize the embeddings such that the nodes that are more likely to co-occur in these walks are closer in the feature space.

Given a node u, the objective is to maximize the log-likelihood of the neighborhood, formulated as (2.1) Here, $f(u)$ and $f(v)$ are the feature representations of nodes u and v, respectively, and $N(u)$ is the neighborhood of u.

$$\max_f \sum_{u \in V} \log \prod_{v \in N(u)} \frac{e^{f(u) \cdot f(v)}}{\sum_{n \in V} e^{f(u) \cdot f(n)}} \tag{2.1}$$

Node2Vec introduces two hyperparameters, p and q, to guide the random walk. While p controls the likelihood of revisiting the previous node, q influences the likelihood of exploring distant parts of the graph.

The Node2Vec algorithm typically involves the following steps:

```
1. Random Walk Generation:
     Generate random walks starting from each node, guided by
     parameters p and q
2. Optimization:
     Use the generated random walks as input sequences for
     a Skip-Gram model and optimize the objective function to
     learn the node embeddings
```

Node2Vec provides a flexible and efficient methodology for learning feature representations of nodes in a graph. By leveraging the foundation of random walks and Skip-Gram models, it creates meaningful embeddings that can be used in a myriad of machine learning and network analysis tasks.

2.2.3 Maximum Flow Problem

The Maximum Flow problem [27] is an optimization problem that deals with finding the largest possible flow in a flow network from a source node to a sink node. This problem has been studied extensively and is applicable to a variety of domains including transportation, logistics, and telecommunications. Various algorithms exist for solving this problem, such as the Ford-Fulkerson and Edmonds-Karp algorithms.

Given a directed graph $G = (V, E)$ with source node s and sink node t, each edge (u, v) has an associated capacity $c(u, v)$. The goal is to find a flow $f(u, v)$ such that:

1. Capacity Constraints: $0 \leq f(u, v) \leq c(u, v)$
2. Conservation of Flow: $\sum_{u \in V} f(u, v) = \sum_{u \in V} f(v, u)$ for all $v \in V \setminus \{s, t\}$
3. Maximize $f = \sum_{v \in V} f(s, v)$

The Ford-Fulkerson algorithm iteratively augments the flow along feasible paths from the source to the sink until no such paths remain.

```
1. Initialize flow f(u, v) = 0 for all (u, v) in E
2. While there exists a path p from s to t in the residual graph:
    a. Find bottleneck capacity b on path p
    b. Augment f along path p by b
3. Return f
```

The term "heuristic" in Maximum Flow generally refers to techniques that offer practical solutions by trading off optimality for computational efficiency. For instance:

- Preflow-Push Algorithms: Instead of finding augmenting paths, these algorithms maintain a "preflow" and iteratively push flow towards the sink.
- Capacity Scaling: Prioritize paths with higher capacities to minimize the number of iterations.

The Maximum Flow heuristic provides an efficient and practical approach to solving a myriad of problems in various domains. Its versatility and adaptability make it a cornerstone in the realm of optimization algorithms.

2.2.4 Constraint Satisfaction Problem

The Constraint Satisfaction Problem (CSP) [89] has been recognized as a fundamental concept in artificial intelligence, computer science, and operations research. The versatility of CSPs makes them applicable in various domains such as scheduling, planning, vehicle routing, and natural language processing.

A Constraint Satisfaction Problem is formally defined as a triple $CSP = (X, D, C)$, where:

- $X = \{x_1, x_2, \dots, x_n\}$ is a set of variables.
- $D = \{D_1, D_2, \dots, D_n\}$ is a set of domains, one for each variable.
- $C = \{C_1, C_2, \dots, C_m\}$ is a set of constraints.

A domain, D_i, consists of a set of allowable values $\{v_1, \dots, v_k\}$, for variable X_i. Different variables can have different domains of different sizes. A constraint C_i is a relation involving a subset of variables that specifies the allowable combinations of their values. Constraints can be represented as equations, inequalities, or tables of allowed values.

Backtracking is a depth-first search algorithm for solving CSPs. It incrementally builds solutions by choosing values for one variable at a time and backtracks when a variable has no legal moves. Algorithms like Hill-Climbing and Simulated Annealing can be used for solving CSPs. These algorithms start with an initial assignment and move to neighboring states by altering the value of one or more variables. Techniques like Forward Checking and Arc Consistency are used to reduce the domain of the variables, thereby simplifying the problem.

Constraint Satisfaction Problems offer a general framework for modeling and solving combinatorial problems. They have numerous applications in various fields, making them a critical subject in the study of optimization and artificial intelligence.

2.3 Stochastic Optimization and Machine Learning

Stochastic optimization and machine learning are interdisciplinary fields that focus on finding optimal solutions in systems with uncertainty. In machine learning, one of the paradigms that heavily employs stochastic optimization is Reinforcement Learning (RL).

2.3.1 Stochastic Optimization

In optimization, the objective function is the quantity that we are trying to minimize or maximize. The constraints are the restrictions that the solution must satisfy. Stochastic optimization deals with optimizing objectives that are subject to stochastic (random) variations. This is in contrast to deterministic optimization, where the objective function and constraints are known exactly. Formally, a stochastic optimization problem can be defined as (2.2), where $f : \mathcal{X} \times \Xi \rightarrow \mathbb{R}$ is the objective function, $\mathbf{x}$ is the decision variable, $\boldsymbol{\xi}$ is a random variable with a known probability distribution, and $\mathcal{X}$ is the feasible set for $\mathbf{x}$.

$$\min_{\mathbf{x} \in \mathcal{X}} \mathbb{E}[f(\mathbf{x}, \boldsymbol{\xi})] \tag{2.2}$$

In stochastic optimization, Gradient Descent (GD) and its variant, Stochastic Gradient Descent (SGD), are widely used algorithms. Gradient descent and stochastic gradient descent are both iterative optimization algorithms used to find the minimum of a function. Gradient descent works by starting with an initial guess for the minimum of the function and then iteratively updating the guess in the direction of the negative gradient of the function. The gradient of a function is a vector that points in the direction of the steepest ascent of the function. By moving in the direction of the negative gradient, we are moving in the direction of the steepest descent of the function. Stochastic gradient descent is a variant of gradient descent that uses only a single data point at a time to update the guess for the minimum of the function. This makes it more efficient than gradient descent, but it can also be less accurate. The updated rule for GD and SGD are given by (2.3a) and (2.3b), respectively, where $\boldsymbol{\xi}_t$ is a sample drawn from the distribution of $\boldsymbol{\xi}$ at iteration t.

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \alpha \nabla f(\mathbf{x}_t) \tag{2.3a}$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - \alpha \nabla f(\mathbf{x}_t, \boldsymbol{\xi}_t) \tag{2.3b}$$

2.3.2 Machine Learning

Machine learning algorithms often involve optimization problems, where the objective function $J(\boldsymbol{\theta})$ represents a loss or cost, and $\boldsymbol{\theta}$ are the parameters to be optimized.

Reinforcement Learning (RL) [100] is a subfield of machine learning focused on learning optimal policies in environments with stochastic dynamics. In RL, learning agents improve their behavior in an environment by trial and error. The agent interacts with the environment and receives rewards or punishments for its actions. The RL problem is usually modeled as a Markov Decision Process (MDP), defined by the tuple (S, A, P, R), where S is the state space, A is the action space, P is the state transition probability, and R is the reward function.

Stochastic optimization is a key component of reinforcement learning. The agent uses stochastic optimization algorithms to update its policy, which is a mapping from states to actions. The policy is updated so that the agent is more likely to take actions that lead to high rewards.

One popular algorithm in RL is Q-learning, which aims to learn the action-value function $Q(s, a)$ that represents the expected return (cumulative discounted reward) of taking action a in state s. The update rule for $Q(s, a)$ is:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right) \tag{2.4}$$

Another important class of algorithms in RL is policy gradient methods. These algorithms directly parameterize the policy $\pi_{\boldsymbol{\theta}}(a|s)$ and update the parameters $\boldsymbol{\theta}$ to maximize the expected return (2.5), where τ is a trajectory and $R(\tau)$ is the return of τ. The policy gradient is computed as (2.6).

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} [R(\tau)] \tag{2.5}$$

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} \left[\sum_{t=0}^{\infty} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a_t | s_t) R(\tau) \right] \tag{2.6}$$

Actor-Critic methods combine the best aspects of value-function methods like Q-learning and policy-based methods like policy gradients. The "Actor" updates the policy in the direction that increases the expected return, while the "Critic" estimates the value function that guides the Actor's updates. Mathematically, the Actor updates the policy $\pi_{\boldsymbol{\theta}}(a|s)$ using the Critic's estimation $V(s)$ or $Q(s, a)$ of the state-value or action-value function by (2.7a).

The Actor's policy $\pi_{\boldsymbol{\theta}}(a|s)$ is updated using gradients that consider the Critic's value by (2.7b) The Critic's value function $V_{\mathbf{w}}(s)$ or $Q_{\mathbf{w}}(s, a)$ is updated using Temporal Difference (TD) error by (2.7c) By combining the strengths of both value-function and policy-based methods, Actor-Critic algorithms are often more stable and efficient than either approach alone.

$$\nabla J(\boldsymbol{\theta}) = \mathbb{E}_{s \sim \rho, a \sim \pi} \left[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(a|s) Q_{\mathbf{w}}(s, a) \right] \tag{2.7a}$$

$$\delta_t = r_t + \gamma V_{\mathbf{w}}(s_{t+1}) - V_{\mathbf{w}}(s_t) \tag{2.7b}$$

$$V_{\mathbf{w}}(s_t) \leftarrow V_{\mathbf{w}}(s_t) + \alpha \delta_t \tag{2.7c}$$

Stochastic optimization serves as the backbone for many machine learning algorithms, particularly in reinforcement learning, where the aim is often to find optimal policies in uncertain environments. Various algorithms like Q-learning, policy gradient methods, and Actor-Critic methods offer different approaches to solving these challenging problems.

Chapter 3: Fast Adaptive AI Agent System with Domain Model

When it comes to Artificial Intelligence, a commonly acknowledged definition is around *rationality* [89]. Multiple approaches are explored, such as the Turing test approach, cognitive modeling approach, and the "laws of thought" approach. This study focuses on the *cyber-physical AI agent* approach, which is coherent with the production robots and transportation robots introduced in Chapter 1 and Section 3.1.

The concept of *cyber-physical AI agent* intersects between *cybernetic agent* and *rational agent*. An agent is an entity existing in either the physical domain or cyber domain (virtual agent) that acts to the environment through a form in its domain. A cybernetic agent approach is the interdisciplinary study of systems, control, and communication in animals, machines, and organizations [15]. A rational agent approach [89] involves decision-making (acting) to achieve the best outcome or the best-expected outcome under uncertainties. Thus, a cyber-physical AI agent focuses on the adaptive interactions between an agent and its changing environment with cyber-physical interactions achieving the best-expected outcomes considering factors such as beliefs, desires, and intentions. It perceives the environment through sensors and communications and makes rational action decisions to the environment through actuators for the best-expected outcomes and receives causal feedback. This study refers to "cyber-physical AI agent" as "AI agent", or just "agent", "robot" for simplicity.

The central AI cognitive functions explored in this study are *knowledge*, *perception*, *reasoning*, *predicting*, *planning*, *learning*, *decison-making* and *communication*. These topics will be elaborated upon in the subsequent sections. But first, it is important to understand the concept of the domain model and its relationship with knowledge-based agents. Although a majority of AI research emphasizes the *accuracy* and *optimality* of an agent's decision-making capabilities to achieve the best-expected outcomes, the reality of practical AI applications requires a broader view. Specifically, factors such as *efficiency*, *effectiviy* and *reliability* also come into play. As a result, traditional training techniques often employed in AI, whether they are online methods like model-free reinforcement learning or offline methods such as imitation learning and offline reinforcement learning, may not be fully adequate for practical implementations. These approaches may fall short when the requirements extend beyond mere accuracy and optimality to include scalability, computational efficiency and real-world robustness.

Online training techniques like model-free reinforcement learning methods, including multi-agent reinforcement learning [131], offer the advantage of not needing a pre-defined domain model. They learn directly from interactions with the environment, granting them a degree of flexibility especially useful when the environment isn't fully understood. However, these online methods often have slower learning curves, particularly in intricate and complex settings. They are also more susceptible to data noise and bias, which can hold back their performance. In the early stages of training, agents are prone to making errors. In a smart factory setting, such mistakes could have severe consequences, ranging from equipment damage and compromised product quality to potential injuries to workers. Furthermore, an inadequately trained agent may fail to discover the optimal policy for navigating its environment, leading to issues like diminished productivity and increased operational costs.

Offline reinforcement learning methods, as surveyed in [84], learn from a dataset of previously recorded experiences, eliminating the need for real-time interaction with the environment. This can be more efficient than online training techniques. However, the effectiveness of offline methods heavily depends on the quality and representativeness of the dataset. If the dataset is either insufficient or not reflective of the actual operational environment, the trained agent may fail to learn the optimal policy for action. Moreover, gathering data from real-world factories presents its own set of challenges. The factory setting is typically noisy and chaotic, making accurate data collection a complicated task. In addition, security concerns around operational data may deter factories from participating in data collection efforts, as this information is often sensitive and with restricted access. This can lead to expensive and time-consuming data collection processes, further complicating the training of agents for practical applications.

The domain model approach underscores the necessity of leveraging context-specific or even expert knowledge to develop efficient and reliable AI applications. A domain model serves as a comprehensive representation of various elements relevant to a specific context. This can include knowledge about the environment, state space (perception), action space, training trajectories (sequences of actions), and other agents in MAS or MRS. The domain model embodies various entities, concepts, relationships, processes, rules, and constraints. These components together facilitate the various cognitive functions of AI agents, such as reasoning, predicting, planning, learning, decision-making, and communication. In the context of smart factories and advanced manufacturing, a domain model might encapsulate a range of crucial information. This could include the shop floor layout, the diverse types of robots and machinery in operation, the products under manufacturing, and the overall production processes and flows. By doing so, it provides a structured framework that aids in the complex task of maintaining and optimizing smart factory operations.

The efficiency of involving domain models in AI agents, particularly in environments where collecting large datasets is impractical or too costly, can be a significant advantage. Some of the specific benefits this approach offers, especially in the context of MRS-driven are as follows:

- *Model Size and Complexity*: Deep learning models, particularly those at the state-of-the-art level, are inherently resource-intensive, necessitating substantial computational power for both training and inference phases. This poses a challenge in Multi-Robot Systems (MRS) environments, where computational resources are often constrained. In contrast, domain models offer a more lightweight alternative. These models incorporate predefined rules and constraints, thereby facilitating the design of smaller neural networks. Consequently, fewer training steps are needed to capture the uncertainties that are not explicitly handled by the domain models.

- *Faster Training*: Deep learning models typically require large amounts of data and extensive training time to reach peak performance. In contrast, a domain model can often start producing reasonable, near-optimal and even optimal solutions. This is particularly useful in a smart factory setting where quick adaptability to changes is crucial.

- *Rule-based Constraints*: Domain models are excellent at incorporating hard and soft constraints into decision-making processes. This is incredibly useful in an MRS context where, for example, robots might have to respect spatial constraints or production timelines. Deep learning models can struggle to incorporate such rule-based logic without complex architecture designs or loss functions.

- *Scalability*: As mentioned in Chapter 1, scalability is a challenge when it comes to applying large models in MRS. The size and complexity of deep learning models often make them difficult to deploy in a distributed system comprising multiple agents with limited computational resources. Domain models can more easily scale with the system's needs, offering a more resource-efficient solution.

- *Transparency and Interpretability*: Domain models often provide more transparent and interpretable decisions compared to deep learning models. In environments like smart factories, where safety and reliability are critical, being able to understand and trace the decision-making process can be vital.

The trade-off is often in the fine-grained, nuanced decision-making that large, well-trained neural networks can provide. However, for many practical applications in smart factories, the "good enough" solutions provided by a well-designed domain model might be entirely sufficient, especially factoring in the non-stationarity. The efficiency gains can make this approach more practical for real-world implementation.

The challenges of using domain models in various applications largely arise from their inherent complexity, the necessity for regular updates, and their risk of becoming obsolete if foundational assumptions change. However, adopting graph models as domain models could serve as a viable solution to mitigate some of these challenges. One of the most compelling features of graph models is their ability to address complexity effectively. Unlike monolithic domain models, which often necessitate a deep understanding of numerous nuances, graph models break down the complex system into a network of nodes (representing entities) and links (representing relationships). This modular structure allows for easier management and the ability to add, modify, or remove specific components without disrupting the overall system.

In addition to simplifying complexity, graph models offer the advantage of being dynamically updatable. Traditional domain models may require extensive modifications in response to environmental changes, but graph models facilitate more straightforward updates—sometimes as simple as adding or removing a node or link. This adaptability is particularly beneficial in fast-evolving settings, such as smart factories, where the ability to swiftly adjust to new conditions is crucial.

Here are some specific examples of how domain models with ML and AI methods can be used to resolve MRSs or MASs problems in smart factories or smart manufacturing:

- *Path planning*: ML can be used to learn the optimal paths for robots to take through the factory, taking into account the obstacles, the other robots, and the current state of the production processes.

- *Collaboration*: AI can be used to develop algorithms that allow robots to collaborate with each other to complete tasks. For example, robots could be programmed to share information about their locations and tasks, so that they can avoid collisions and optimize their movements.

- *Scheduling*: AI can be used to develop scheduling algorithms that can take into account the uncertainty and dynamicity of the factory environment. For example, algorithms could be developed that can re-schedule tasks if a robot breaks down or if a new production demand arrives.

- *Reactive control*: AI can be used to develop reactive control algorithms that allow robots to respond to unexpected events in the factory environment. For example, robots could be programmed to detect and avoid obstacles, or to react to changes in the production process.

This chapter is extended from domain knowledge modeled as graphs, to single intelligent robots, and then to the cyber-physical AI agent system that addresses the challenges in smart factories.

3.1 Domain Knowledge of MRS as Graph Models

Graph modeling is a technique that uses graphs [11] to represent, analyze, and interpret complex systems and networks. In a graph model, nodes (or nodes) represent entities, and edges (or links) represent relationships or interactions between entities. This form of modeling is particularly useful for systems in which the structure and relationship between components are important features.

Meanwhile, both use graphs, a commonly known concept of graphical modeling is a technique in statistics and machine learning where graphs are used to represent probabilistic relationships among a set of variables. The graph serves as a compact and intuitive representation of the conditional dependencies and independencies between variables. This differs from graph modeling, which is more general and can represent any kind of relationship between entities, not just probabilistic ones.

Although the author employs both in research, this chapter's scope is using graph and graph theory to mathematically represent the MRS-driven smart factory. Graph models can be used to represent the interactions and dependencies between robots, the environment, and other entities, and to analyze, reason and predict the system's behavior. However, there are a number of challenges in using graph models for MRS and MAS in smart factories.

Graph models face several challenges when applied to MRS and MAS, particularly in the context of smart factories [98]. Firstly, scalability becomes a major issue. As the number of robots and other elements within the system expands, the graph model also grows exponentially in size. This increased complexity presents challenges in terms of storage, manipulation, and analysis of the model. Secondly, environmental uncertainty adds another layer of complexity [51]. The ever-changing conditions in a smart factory can make it challenging to encapsulate the nuanced interactions between robots and their surroundings within a graph-based representation. Lastly, traditional graph models are often inadequate in capturing the dynamics of MRS and MAS [98]. While they excel in representing static relationships, they fall short in portraying temporal variations and dynamic interactions. To overcome these challenges, several strategies are being explored. One approach involves the development of lightweight graph models tailored for scalability and computational efficiency. Another alternative is the use of probabilistic graph models capable of accounting for environmental uncertainties. Researchers are also working on novel graph models designed to capture the dynamic aspects of MRS and MAS.

Three graph models are presented in this chapter. They are based on the MRS-driven smart factory's main components, production robots, transportation robots and wireless networks. These three models are proposed to resolve multiple engineering problems including finding the optimal task assignments for both production robots and transportation robots with optimized productivity and energy efficiency; facilitating communication for collaboratively production task execution against actuator errors; and facilitating optimal and near-optimal transportation task execution planning.

These models allow mathematical representation of the spatial environment of smart factories, complex heterogeneous MRS and its multiple dynamic production flows in operation.

These graph models can be further vectorized and used for ML and AI techniques to solve the engineering problems mentioned above. On the other hand, they allow analysis of multiple production flows in the physical domain, wireless communications and topology in the cyber domain and their interactions in the cyber-physical domain.

3.1.1 Cyber-Physical Multi-Robot System (CPMRS) Model

This section introduces the Multi-Robot Systems (CPMRS) model, a framework designed to enable effective coordination within smart factories featuring Multi-Robot Systems (MRS). As discussed in Section 1.3, the MRS in smart factories exhibit inherent complexities such as time-dynamic production flows and system heterogeneity. These complexities introduce non-stationarity into the system, making deterministic solutions less effective. Therefore, a non-deterministic, decentralized AI computing approach is required. CPMRS aims to facilitate MRS coordination in a decentralized manner. It does so by dynamically leveraging information from edge computing and the MRS itself. By regulating the gathering and exchange of this real-time data, CPMRS makes it possible to perform effective coordination via AI computing.

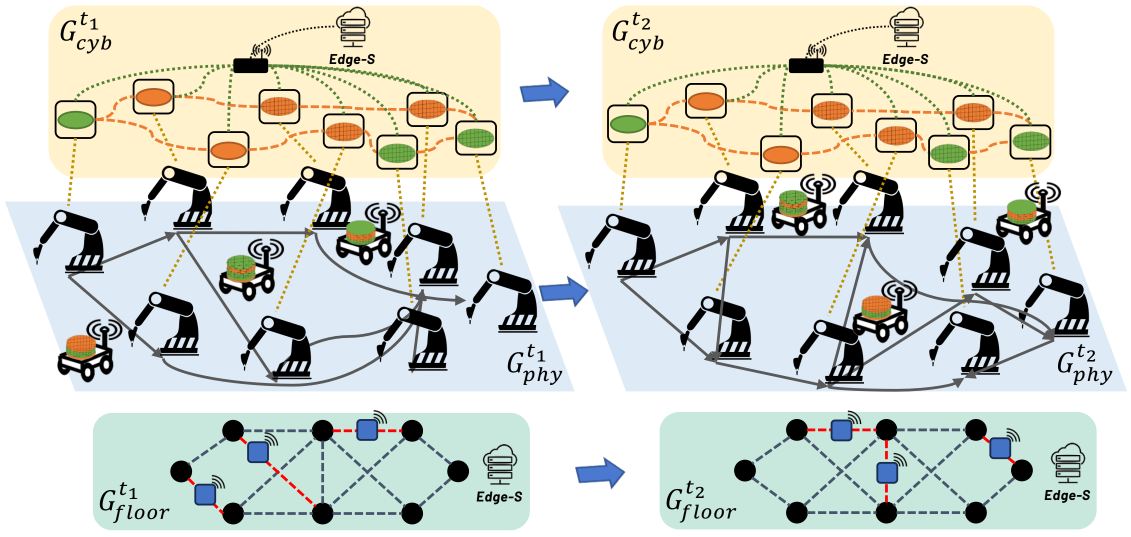


Figure 3.1 An example of CPMRS with an edge server, 8 production robots and 3 transportation robots.

The CPMRS model features a dual-layer network architecture comprising:

- *Physical Domain G_{phy}:* Represents real-time physical interactions among MRS, including the Multi-Robot Task Allocation (MRTA) to handle time-dynamic production flows. In this domain, an upstream robot's output serves as input for a downstream robot.

- *Cyber Domain G_{cyb}:* Focuses on wireless network communications and social communications among the MRS for information exchange and AI computing.

As illustrated in Figure 3.1, both the physical and cyber domains share a common set of nodes, representing heterogeneous production robots. This forms a cohesive, dual-layer model. To enrich the model with additional domain-specific information, an extra layer, denoted as G_{floor}, can be incorporated. As shown in Figure 3.1, this third layer consists of both production and transportation robots, thereby capturing the dynamics of transportation task execution. Furthermore, CPMRS can exhibit time-varying behavior, as indicated by the superscript t in G^t_{phy}, G^t_{cyb}, and G^t_{floor}. This temporal notation allows the model to adapt its topology in response to changes in MRS operations.

In the physical domain, multiple, time-varying production flows serve as crucial components. Referring to the various transportation path segments outlined in Section 1.3, each production flow consists of an alternating sequence of nodes and links. The configuration of these flows can be dynamically adjusted through MRTA mechanisms.

The cyber domain is largely dependent on wireless networks. The network topology could be either fully connected or partially connected, depending on the specific engineering challenges at hand. A *fully connected topology* implies that every node is directly linked to every other node in the network. While this offers the advantage of direct communication between all pairs of nodes, it also has downsides. Namely, the implementation and maintenance costs are high, and the number of required links grows quadratically with the addition of nodes, making the topology impractical for large networks. On the other hand, a *partially connected* *social topology* only necessitates that some nodes are directly connected. Such topologies are prevalent in real-world social networks like Facebook and Twitter, where not all users are interconnected. The degree of connectivity can vary, ranging from nodes connected to only a few other nodes to nodes with many connections.

In the physical domain, dynamic, time-varying production flows serve as critical components. As elaborated in Section 1.3, each production flow is characterized by an alternating sequence of nodes and links. These flows can be flexibly configured using MRTA to adapt to real-time needs.

The cyber domain predominantly relies on wireless networks, the topology of which could be either fully connected or partially connected based on the engineering challenges being addressed. A *fully connected topology* signifies that every node in the network is directly linked to all other nodes. Although this setup facilitates immediate communication between any two nodes, it comes with drawbacks such as elevated implementation and maintenance costs. Additionally, the number of required links increases quadratically as nodes are added, rendering it unsuitable for large-scale networks.

Contrastingly, a *partially connected social topology* requires only some nodes to be directly linked. This type of topology is commonly found in real-world social networks like Facebook and Twitter, where not all users are interconnected. The degree of connectivity can be adjusted, with some nodes having only a few connections and others having many.

Considering the scalability limitations and impracticality of maintaining a fully connected topology for complex MRS operations, subsequent chapters advocate for adopting a partially connected social topology within the CPMRS model. This approach aligns well with the inherent characteristics and capabilities of production robots, featuring a stochastic, partially connected framework that allows for adaptable and flexible robot interactions.

In the CPMRS framework, wireless communications are subject to specific timing and informational constraints. These constraints effectively transform the communication process into a social interaction paradigm among AI-enabled robots.

3.1.1.1 Physical Domain

Initially introduced in Section 1.3, a heterogeneous MRS is characterized by $(M, \mathbf{N}, \boldsymbol{\omega})$, with individual production robots denoted by $R_{m,n}$, where $m = 1, \ldots, M$, $n = 1, \ldots, N_m$, and $N_m \in \mathbf{N}$. The parameter M distinguishes heterogeneity based on the robots' properties, such that type-m robots only execute type-m production tasks. $\mathbf{N} = (N_1, \ldots, N_m, \ldots, N_M)$ characterizes the number of each type of production robot, with two robots considered adjacent if their numbers differ by one.

A finished product is processed from raw material and transported along a series of heterogeneous production robots by transportation robots in smart factories, forming a discrete production flow. Consequently, the completion of products involves the collaboration of various groups of production robots in MRS in a temporal-spatial manner. MRTA composes such production flows to ensure required sequential processes for product completion, while dynamically adjusting production flows to enhance productivity and energy efficiency. Compared to factory automation, which employs dedicated robots for static production flows a heterogeneous MRS endows a smart factory with flexibility and agility.

The physical domain network, $G^t_{phy} = (R_{m,n}, \mathbf{E}^t_{phy})$, treats each production robot as a node and time dynamic production flows (3.1a) as links, given by (3.1b). Time-dynamic production flows are therefore time-dynamic simple paths, or physical domain topology, distinguished by time slot t in G^t_{phy}, which is vectorizable domain knowledge for AI computing from edge computing. The MRS operates with time-slotted synchrony in the unit time slot denoted by t, which represents the smallest unit of task execution and production flow adjustments.

$$\boldsymbol{\rho}^l = (R_{m_1,n_1}, R_{m_2,n_2}, \ldots, R_{m_{i-1},n_{i-1}}, R_{m_i,n_i}) \tag{3.1a}$$

$$G^t_{phy} = (\{R_{m,n}\}, \mathbf{E}^t_{phy} \leftarrow \{\boldsymbol{\rho}^1, \boldsymbol{\rho}^2, \ldots, \boldsymbol{\rho}^l, \ldots\}) \tag{3.1b}$$

Figure 3.2 illustrates a physical domain network consisting of 12 robots capable of performing four types of production tasks. Governed by MRTA, these robots collaborate to manufacture a variety of products. As a production flow change occurs at t_2, the links in $G_{phy}^{t_1}$ are altered to the links in $G_{phy}^{t_2}$. From $t = t_1$ to $t = t_2$, $R_{2,2}$ participates in two production flows at both time periods but collaborates with different groups of robots and processes distinct products. Simultaneously, although $G_{phy}^{t_1}$ and $G_{phy}^{t_2}$ share the same "bed" product, the production flow is adjusted to accommodate the new "couch" production demand. These complex physical time dynamics are achieved through MRTA and are investigated by [76]. Consequently, in this chapter, we do not distinguish product denotation l with a time index.

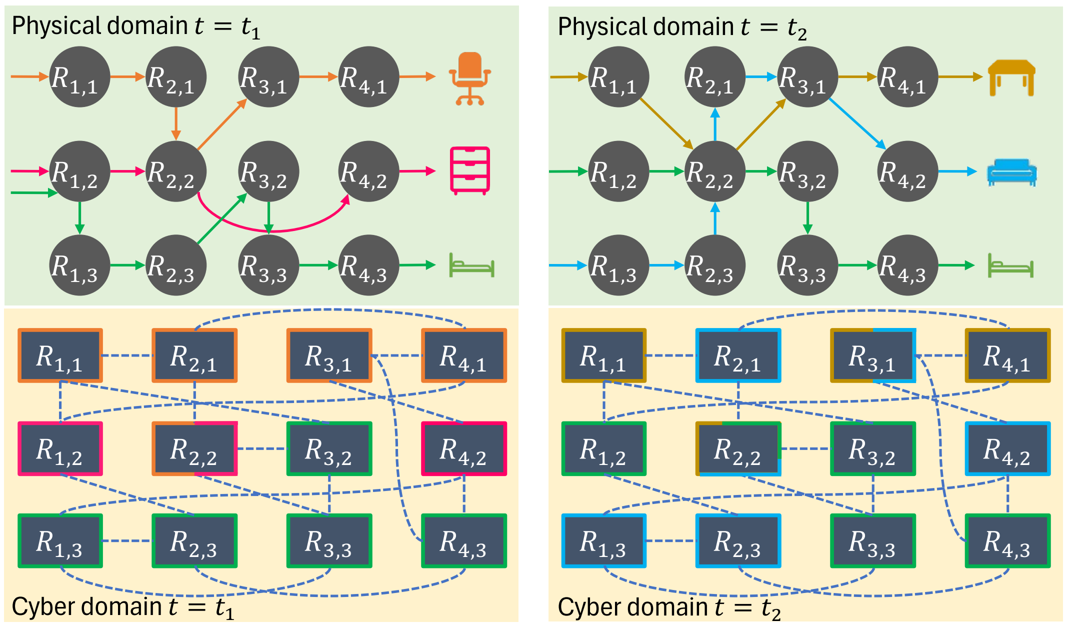


Figure 3.2 A CPMRS with a physical domain consisting of multiple time-varying production flows and a cyber domain consisting of a partially connected social topology.

3.1.1.2 Cyber Domain

In the cyber domain of the proposed CPMRS model, production robots engage in information exchange through Robot-to-Infrastructure-to-Robot (R2I2R) communications [42] and receive task assignments from the edge computing via Robot-to-Infrastructure (R2I) communications. This process enables adaptive and collaborative coordination through AI computing.

Due to the decentralization and agent-based modeling inherent in MRS, as well as the scalability concerns that prevent edge computing from accessing sensor data from every robot, real-time centralized robot manipulation that provides the edge computing with complete information for estimating the accuracy of all robots, imposes a heavy computational burden on the edge computing and wireless resources while also limiting scalability. Moreover, exhaustive feedback of each task execution from the downstream robots to the upstream robots can achieve decentralized accuracy estimation, but this approach presents NP-hardness [72] and is thus computationally intractable for production robots with limited computation capability.

Consequently, in order to propose a novel collaborative MRS coordination approach in Section 6.1 via social learning, leveraging information available from individual robots and social communications within a partially connected wireless topology in the CPMRS, as detailed. Social learning can be both computationally lightweight and effective, even in the context of incomplete observations [124]. A partially connected network has been demonstrated to offer high efficiency [43, 71]. Wireless message passing is conducted among a subset of robots, rather than all robots, reducing redundant communications and unnecessary message passing that do not enhance observability. This approach also improves communication delays and optimizes the use of wireless resources.

The cyber domain network, denoted as G_{cyb}, shares the same node set $R_{m,n}$ with the physical domain network G^{t}_{phy}. However, the links in G_{cyb} are derived stochastically and do not need to be changed unless the node set changes. Algorithm 3.1 provides a unified-degree, stochastic network formulation algorithm that outputs the stochastic cyber network topology, which is executed by the edge computing.

Algorithm 3.1 iteratively considers each robot $R_{m,n}$ as the root, with robot type difference as a constraint, and links d robots as cyber neighbors through random choices. The numerical parameter k represents the ratio of the number of intra-robot-type links over the total number of links d. The cyber network degree is thus a composite degree denoted by $\mathcal{D}_{cyb} = (d, kd, (1-k)d)$, meaning that each robot in the cyber network G_{cyb} has d links, of which kd links are to intra-type robots while $(1-k)d$ links are to inter-type robots. Therefore, a small set of fully connected robots (denoted as cyber neighbors in Algorithm 3.1) forms a cyber cluster, and a robot can be included in multiple clusters, ensuring that the cyber network is connected and messages can be passed between clusters.

Algorithm 3.1: Stochastic Cyber Topology Formulation

```
   Data: $G_{phy}(\{R_{m,n}\}, \mathbf{E}_{phy}), \mathcal{D}_{cyb} = (d, kd, (1-k)d)$
1  Initialization:
2  $G_{cyb}(\{R_{m,n}\}, \mathbf{E}_{cyb}) \leftarrow G_{phy}$
3  $\{R_{m,n}$.Cyber_Neighbor$\} \leftarrow \emptyset$
4  for $R_{m,n}$ in $\{R_{m,n}\}$ :
5      while $R_{m,n}$.Cyber_Neighbor $< d$ :
6          if $R_{m,n}$.Cyber_Neighbor.same_type $< kd$ :
7              Candidates $\leftarrow \{R_{m\pm1,1}, \dots, R_{m\pm1,N_m}\}$
8              $R_{m,n}$.Cyber_Neighbor.same_type $\leftarrow$
9              Random_Choose (Candidates)
10         else:
11             Candidates $\leftarrow \{R_{m,n}\} \setminus R_{m\pm1,1}, \dots, R_{m\pm1,N_m}$
12             $R_{m,n}$.Cyber_Neighbor.different_type $\leftarrow$
13             Random_Choose (Candidates)
```

Social communication in a multi-robot system (MRS) refers to the exchange of information among linked nodes, commonly termed cyber neighbors. This concept is also known as the diffusion strategy in the context of social MRS [94]. Social communication primarily governs the wireless interactions within the cyber domain network. Unlike general communications, which may involve all nodes, social communications are restricted to occur only between directly linked nodes.

These social communications are scheduled to take place at each discrete time slot. In contrast, communications between the MRS and edge computing infrastructure are event-driven, typically occurring during adjustments to the production flows. These adjustments can span varying time intervals, ranging from hours to days, as highlighted in previous studies [76, 75].

Importantly, social communications within the MRS are designed to be efficient in a partially connected cyber network. The system avoids retransmission and back-forwarding mechanisms to conserve wireless resources, thereby ensuring an economical use of the network bandwidth.

Figure 3.2 shows the cyber domain network with composite degree $\mathcal{D}_{cyb} = (3, 1, 2)$ of the same 12 robots. Although production flows in $G_{phy}^{t=t_1}$ are adjusted at $t = t_2$, the cyber domain network does not change with such physical adjustments. The randomness of the stochastic cyber network is produced from the sequence of robots in line 4 and function `Random_Choose` in lines 9 and 13 in Algorithm 3.1 in which a robot is unified drawn to link from the candidate list.

3.1.2 Hypergraph MRS Model

As indicated by [62], smart factories can provide real-time responses to dynamic production demands with a time resolution as fine as 10 milliseconds, potentially serving as the length of a time slot in the model. While the reconfiguration of the MRS is not expected to occur at every time slot, the time required to perform multi-robot task allocation (MRTA) must be compatible with these real-time operational needs.

However, solving MRTA in the context of the temporal-spatial MRS model described in Section 1.3 is computationally challenging, being an NP-hard problem. The additional objectives of optimizing productivity and energy efficiency further complicate the matter, rendering conventional techniques like linear programming and exhaustive search impractical for real-time operations.

Although genetic algorithms have been widely adopted to handle interrelated constraints and deliver sub-optimal solutions, they suffer from scalability issues related to time complexity. As a result, they are often considered a secondary option for this application.

To address these challenges, this section introduces hypergraph modeling specifically tailored for the temporal-spatial MRS model discussed in Section 1.3. This approach aims to solve MRTA with constant time complexity, thereby offering sub-optimal solutions in both productivity and energy efficiency suitable for real-time operations in smart factories.

In graph theory, a hypergraph serves as a generalization of a traditional graph. While a standard graph has links that link exactly two nodes, a hypergraph features hyperedges capable of connecting any number of nodes, potentially more than just two. Hypergraphs exhibit several distinctive properties, elaborated as follows:

- *Incidence Matrix:* Similar to standard graphs, hypergraphs can also be represented using incidence matrices. However, these matrices tend to be more complex and may require careful interpretation.

- *Degree:* In a hypergraph, the degree of a node is defined as the number of hyperedges in which that node participates.

- *Duality:* For a given hypergraph H, its dual hypergraph essentially reverses the roles of nodes and hyperedges. In other words, each node in the dual hypergraph corresponds to a hyperedge in H, and vice versa.

- *Connectivity:* The concept of connectivity in hypergraphs is less straightforward than in standard graphs. Various approaches have been proposed to extend the traditional notions of connectivity to the realm of hypergraphs.

Integrated with the temporal-spatial MRS model, the hypergraph model represents temporal resources as hypernodes and spatial resources as weighted links. Each hypernode comprises multiple blocks, which are interconnected by these weighted links.

The hypergraph search algorithm, detailed in Chapter 4, addresses both the ascending task order constraints and the production demands constraints. It aims to optimize objectives such as throughput and energy efficiency.

Additionally, the hypergraph search algorithm outputs the production flow as a simple path within the hypergraph model. It also provides specific production tasks for the production robots and outlines the transportation paths for the transportation robots. This comprehensive output serves as a valuable input for higher-level decision-making processes.

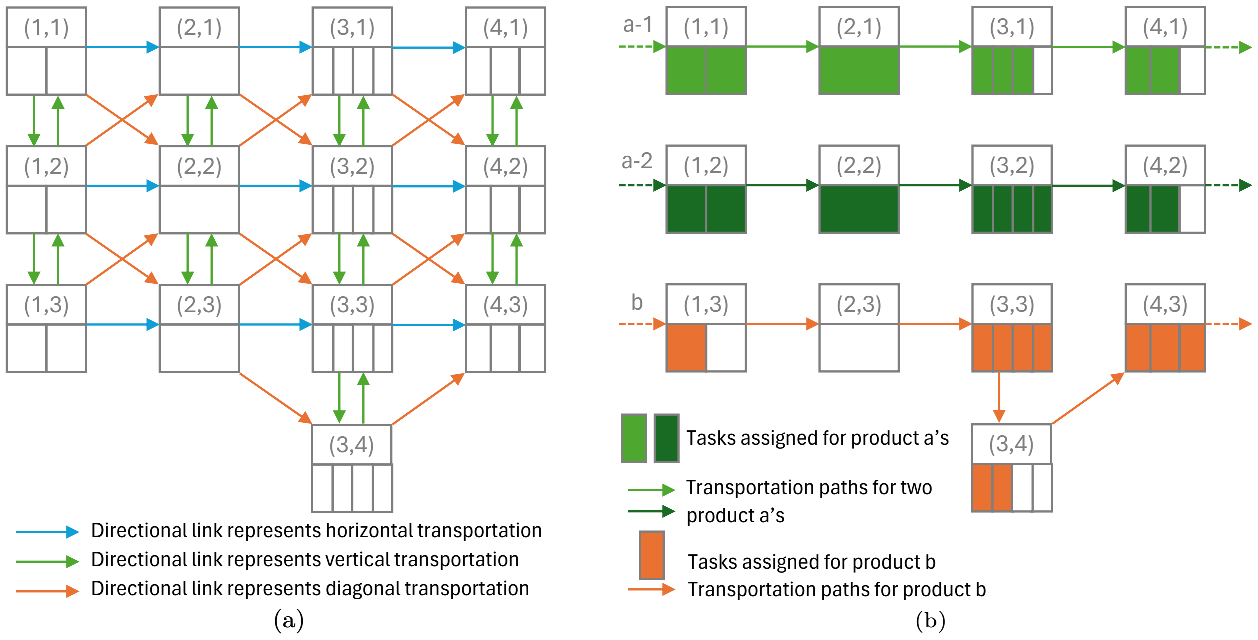

(1,1)
(2,1)
(3,1)
(4,1)
(1,2)
(2,2)
(3,2)
(4,2)
(1,3)
(2,3)
(3,3)
(4,3)
(3,4)
Directional link represents horizontal transportation
Directional link represents vertical transportation
Directional link represents diagonal transportation
(a)
a-1
a-2
b
Tasks assigned for product a's
Transportation paths for two product a's
Tasks assigned for product b
Transportation paths for product b
(b)

Figure 3.3 Hypergraph production robot and MRTA model.

A hypergraph $H(\{R_{m,n}\}, E)$, where $m = 1, 2, \dots, M$ and $n = 1, 2, \dots, N_m$, is defined as a directed hypergraph. In this hypergraph, each hypernode represents a production robot, and the set of weighted links E denotes horizontal, vertical, and diagonal transportation paths. These links are weighted based on the corresponding energy consumption metrics. The notation for hypernodes, $R_{m,n}$, inherits the robot type m, spatial distribution, and multiplexing capability ω_m from the temporal-spatial MRS model.

Following the concept of a hyperedge as defined in [12], a hypernode can be viewed as a set of variables, as expressed in Equation (3.2). Each element $\beta^b_{m,n}$ indicates the single-task execution capability of the robot $R_{m,n}$. A value of $\beta^b_{m,n} = 0$ denotes an unassigned block, while $\beta^b_{m,n} = l$ implies that the b-th task execution capability of the robot $R_{m,n}$ is designated for manufacturing product-l. Consequently, the number of elements in the set $R_{m,n}$ is equal to its multiplexing capability ω_m.

$$R_{m,n} = \{\beta^1_{m,n}, \beta^2_{m,n}, \dots, \beta^{\omega_m}_{m,n}\} \tag{3.2}$$

In the hypergraph, each weighted directed link in E connects two neighboring robots. Specifically, $E = \{e_{i,j:i'j'}\} = \{(R_{m_i,n_j}, R_{m_{i'},n_{j'}})\}$, where R_{m_i,n_j} and $R_{m_{i'},n_{j'}}$ form a neighboring hypernode pair. Figure 3.3 illustrates this hypergraph model, which corresponds to the production robot system shown in Figure 1.3.

For example, within the hypernode $R_{1,1}$, the presence of two blocks indicates that its multiplexing capability is $\omega_1 = 2$. The link $(R_{1,1}, R_{2,1})$ serves as a directed link with the hypernode $R_{1,1}$ as the head, $R_{2,1}$ as the tail, and the link itself carrying a weight of ε_h.

3.1.3 Multi-floor Model of a Smart Factory

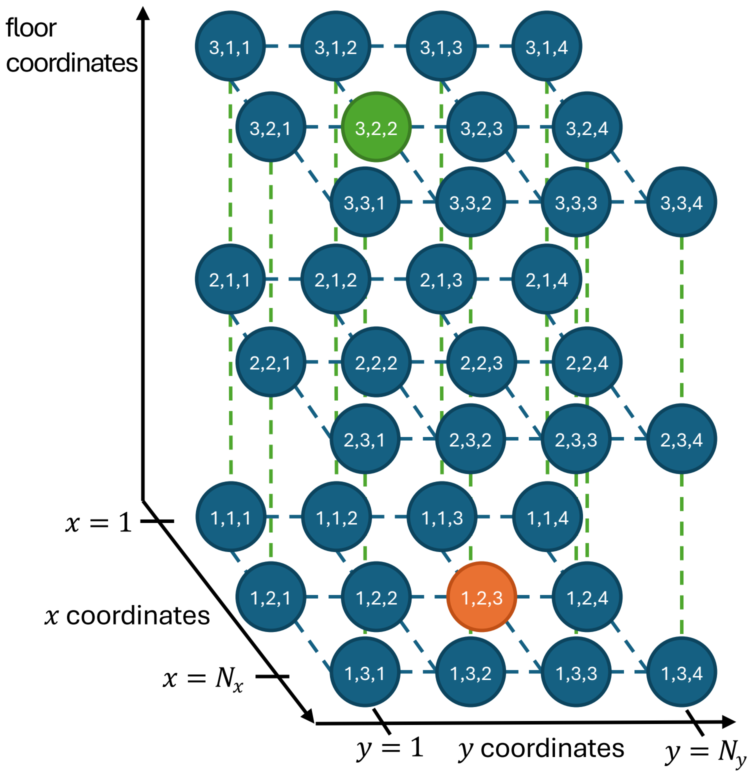


Figure 3.4 The smart factory multi-floor model for transportation.

A multi-floor smart factory transportation environment is characterized by an undirected graph, denoted as G_{floor}. It is defined by the tuple (N_f, N_x, N_y), where each element signifies the maximum value of the floor, x, and y coordinates, respectively. This configuration inherently spans an Euclidean space.

Specifically:

- N_f represents the number of floors, with values ranging from 1 to N_f.
- Each individual floor is structured as an $N_x \times N_y$ lattice graph (also known as a square grid graph). nodes within this graph correspond to either a production robot or a production cell. The x coordinates span from 1 to N_x, while the y coordinates range from 1 to N_y.

The links of G_{floor} depict feasible navigation steps between adjacent nodes, whether through the floor, x, or y coordinates. Given the nature of adjacency, these links are inherently undirected. When navigating within the same floor, nodes adjacent either in the x or y direction are interconnected, consistent with the properties of a lattice graph. For transitions between floors, typically facilitated by UAVs or elevators which come with their own constraints [66, 54], only nodes adjacent in floor coordinates and situated at the links of the floor are connected. This is further elaborated in (3.3).

It's important to note that G_{floor} is a cyclic graph, characterized by the presence of multiple graph cycles. Moreover, all links within this graph bear weights, indicative of the energy consumption associated with traversal. The energy required is contingent upon the link's position within the coordinate space, denoted as $\varepsilon_f, \varepsilon_x, \varepsilon_y$.

The formal definition of G_{floor} is presented in (3.3), where nodes are expressed in terms of $v_{i,j,k}$ coordinates, sequenced by the floor, x, and y dimensions. Additionally, a node positioned at the link, expressed as $v_{i,j_{\text{side}},k_{\text{side}}}$, is located at the side of the ith floor, as elaborated in (3.3c) and (3.3e). Figure 3.4-(a) provides a visualization of an example G_{floor} characterized by $(N_f = 3, N_x = 3, N_y = 4)$. If an AMR is required to navigate from node

$v_{1,2,3}$ to $v_{3,2,2}$, it would necessarily traverse the nodes situated on the links of both the first and second floors to facilitate cross-floor navigation.

$$G_{floor} = (\mathbf{V}_{floor}, \mathbf{E}_{floor}) \tag{3.3a}$$

$$\mathbf{V}_{floor} = \{v_{i,j,k}\}, i = 1, \dots, N_f, j = 1, \dots, N_v, k = 1, \dots, N_h \tag{3.3b}$$

$$\mathbf{E}_{floor} = \{(v_{i,j,k}, v_{i,j',k'})\} \cup \{(v_{i,j_{\text{side}},k_{\text{side}}}, v_{i',j_{\text{side}},k_{\text{side}}})\} \tag{3.3c}$$

$$j' = j \pm 1, k' = k \pm 1, \tag{3.3d}$$

$$j_{\text{side}} = 1, N_v, k_{\text{side}} = 1, N_h, i' = i \pm 1 \tag{3.3e}$$

A transportation-MRS comprises K AMRs, hereafter referred to as K-AMRs, operating with synchronous discrete navigation in each time slot t. Within a given slot, an AMR may either move from one node to another or stay put, an action termed as a "navigation step" throughout this paper. Every navigation step consumes one time slot, independent of energy consumption. This synchronized approach promotes real-time adjustments ensuring collision-free operations, as corroborated by prior studies [19, 76, 75]. Given that industrial AMRs may have diverse navigation step durations, the time slot is designed to be flexible, accounting for longer tasks like cross-floor navigation. The multi-floor model acts both as the environment for the AMR system and as a knowledge base for intelligent decision-making, given that both time and energy consumption metrics for navigation are predefined.

3.2 Single Cyber-Physical AI Agent

In this chapter, a single cyber-physical AI agent is defined as an entity in both cyber and physical domains that is capable of autonomously solving tasks in an uncertain environment. This is achieved through the application of domain knowledge, information gathered during the problem-solving process, and AI cognition. The agent perceives its cyber-physical environment and makes decisions to act upon it. While uncertainties in the task environment are modeled probabilistically, it is noteworthy that the agent is unaware of these probabilistic models.

The notion of a *cyber-physical AI agent* serves as an intersection between the concepts of *cybernetic agent* and *rational agent*. According to [89], rationality at any given moment is determined by four key factors:

- The performance measure that outlines the criteria for success.
- The agent's prior knowledge of the environment.
- The range of actions that the agent can execute.
- The sequence of percepts that the agent has experienced to date.

Therefore, a rational agent is defined as one that, for every possible sequence of percepts, selects an action expected to maximize its performance measure. This selection is based on the evidence gathered through the percept sequence and any built-in knowledge the agent possesses.

In addition to perception and acting, [83] highlights that a cybernetic agent places a strong emphasis on feedback loops for system control. Agents utilize feedback to compare their current state to a predefined goal or desired state. This comparative analysis informs the decision-making process, allowing the agent to adapt its actions in response to environmental changes.

3.2.1 Artificial Intelligence, Cybernetic and Reinforcement Learning

AI is a broad field that incorporates a variety of techniques and methodologies, one of which is Reinforcement Learning (RL). As a subset of machine learning, RL equips agents with the capability to learn tasks autonomously, without explicit programming. As noted in the preface of Chapter 3, RL algorithms leverage data to improve an agent's performance in given tasks. In RL, agents learn to optimize their actions to gain maximum cumulative rewards over time. They do this through a process of trial and error, iteratively updating their behaviors in response to the feedback they receive from the environment. The ultimate aim is to find a policy that consistently leads to desirable outcomes. AI and RL are closely related since RL gives the fundamental rationality - the agent acts to gain more rewards. While RL serves as a mechanism for realizing specific AI objectives—such as mastering games or governing robotic movements—other AI techniques, like deep learning, can be employed to enhance the effectiveness of RL algorithms. This interplay between AI and RL techniques allows for the development of increasingly sophisticated and adaptive systems.

Cybernetics and RL are both disciplines focused on the study of adaptive and learning systems, although with different scopes and emphases. Cybernetics, made known by Norbert Wiener in the 1940s, offers a broader perspective on systems that can self-regulate. Often referred to as "cybernetic systems", these entities employ feedback loops to receive information about their current states and adjust their behaviors accordingly. In contrast, RL is a specialized field within machine learning that concentrates on how agents can learn to optimize their actions in specific environments through trial and error. The agent's primary goal is to maximize cumulative rewards based on its interactions with the environment. The interplay between cybernetics and RL becomes evident when RL algorithms are employed to train agents operating within cybernetic systems. Therefore, cybernetics provides a comprehensive framework for understanding the principles underlying adaptive systems, RL offers a targeted methodology for training agents to behave optimally within those systems.

In this study, RL and stochastic optimization techniques—including tabular methods, Q-learning, and actor-critic algorithms—are extensively employed to enable agent adaptation in environments characterized by unknown probabilistic uncertainties. The rationale for this approach is numerous numerous.

Firstly, RL leverages machine learning models, such as tabular methods and Neural Networks (NN), to approximate optimal policies, state-value functions and state-action-value functions (Q functions). These models are not only supported by a robust body of literature but are also highly versatile, making them suitable for a range of applications.

Secondly, Stochastic Gradient Descent (SGD) serves as a key stochastic optimization (Section 2.3) technique within machine learning. SGD is essentially a stochastic approximation of gradient descent optimization, designed to minimize an objective function formulated as a sum of differentiable functions. Its stochastic nature is particularly well-suited for handling non-stationary, non-deterministic, and unknown probabilistic uncertainties.

Thirdly, RL's concept of "reward"—a real-numbered value reflecting the feedback in cybernetic terms—complements SGD optimization effectively. For SGD optimization applied to neural networks, certain assumptions are generally made: the loss function must be differentiable, approximately convex, and stationary. When examining the convergence rates of gradient-based methods like SGD, researchers often assume that the gradients (or the differences between parameter updates) are bounded in their L2 norm. Real-numbered rewards and well-chosen loss functions—such as Mean Squared Error (MSE) and smooth L1 loss—facilitate these assumptions, thereby aiding in convergence and stability.

Consequently, the cyber-physical AI agent agent model employed in this study adopts an RL framework. Here, the agent's perception of the task environment is represented as a real-numbered state vector, its decision-making is exhibited through a real-numbered action vector, and feedback is quantified as a real-numbered reward. The agent aims to maximize its cumulative reward over time through a process of trial and error, thereby integrating a data-driven aspect into its operational paradigm.

3.2.2 Problem-solving with Domain Knowledge and AI Cognition

Figure 3.5 outlines the framework of the cyber-physical AI agent approach. Within this framework, both engineering challenges and domain-specific knowledge contribute to the development of the domain model. To enhance this model, general computational methods like linear models, Q tables and neural networks are integrated into the *models* module.

As the agent operates within its environment—undertaking perception, learning, problem-solving, and decision-making, certain internal changes or *state transitions* occur. These internal shifts are not always completely visible to the agent. In the context of this study, these invisible or partially visible elements are termed *uncertainties*. This is in line with some academic literature, where the phenomenon is often referred to as *partial observability*.

Additionally, during the trial-and-error learning process, the agent has the capability to collect data from its interactions with the environment. This accumulated data can be employed to update and refine the agent's model, thereby incrementally improving its performance and adaptability over time.

As introduced in Section 3.1, graph models are used to represent domain knowledge, serving as the domain models that guide agents. This domain knowledge forms the foundation upon which an agent's rational decision-making is built during problem-solving. It allows the agent to take reasonably good, near-optimal or even optimal actions without the need for extensive learning. The interplay between domain knowledge and AI cognition in the decision-making process is further detailed in the subsequent sections.

As initially outlined in the introductory section of Section 3.2, cyber-physical AI agents receive real-numbered rewards as a form of feedback. Stochastic optimization techniques are employed to adjust the weights of the general neural network model. The objective is to minimize the loss between the received reward and the reward predicted from the previous action. This process, commonly known as *training*, enables the agent to adapt to environmental uncertainties.

However, this framework presents an engineering challenge related to goal formulation. Specifically, the reward function must be both coherent with the task environment and aligned with the agent's ultimate objectives. Further discussions on the details of constructing a suitable reward function will be elaborated upon in Section 3.2.3.

This section explores the application of graph models, as initially discussed in Section 3.1. These models function as domain models and allow agents to perform autonomous reasoning

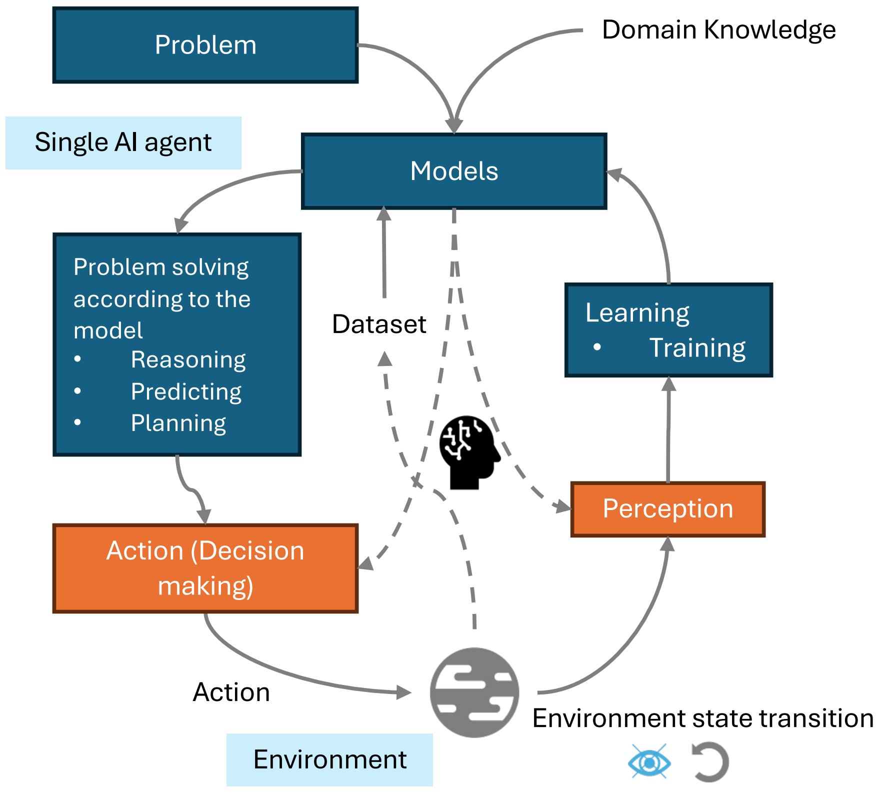


Figure 3.5 A schematic depiction of a cyber-physical AI agent.

and planning tasks within the smart factory context. It is crucial to note that the tasks mentioned here do not concern typical production or transportation tasks commonly found in smart factories. Instead, they relate to the cognitive tasks of reasoning and planning that the agents are tasked with. As for prediction, the primary focus is on making predictions under uncertainties using general models. This topic will be elaborated upon in the subsequent section.

3.2.2.1 Reasoning with Bayesian Network

Reasoning within the framework of a domain model involves applying logical inference and decision-making based on a specific set of rules, relationships, and entities defining the domain. A domain model sets the boundaries and guidelines for reasoning, ensuring that the outcomes are more consistent and accurate. Utilizing a domain model allows an AI agent to achieve a more nuanced understanding of the problem, enhancing the contextual relevance of its reasoning. Additionally, a domain model aids in efficiently navigating the solution space while adhering to predefined rules and relationships. It serves to refine complicated real-world scenarios into a manageable set of entities and relationships, thereby streamlining the reasoning process. When multiple AI agents operate within the same domain, a shared domain model facilitates interoperability by standardizing the rules and constraints underpinning their reasoning and decision-making.

Bayesian networks serve as a robust tool for representing probabilistic uncertainties and capturing interdependencies through conditional probabilities. A Bayesian network, also known as a Bayes net or belief network, is a directed graph where each node is annotated with quantitative probability information. As noted in the preface of Section 3.1, Bayesian networks are graphical models widely used in ML. While graph models are powerful domain models, as briefly mentioned in Section 5, they often fall short in terms of scalability—a requirement for smart factories. As elaborated in Section 3.1.1, each production robot in an MRS is connected to neighboring nodes in two distinct domains: the physical domain, defined by the production flow, and the cyber domain, characterized by a stochastic partially connected wireless network.

In this reasoning process, Bayesian networks integrate information from upstream production robots in each production flow, denoted as G_{phy}^{t}, along with domain knowledge obtained from task execution for reasoning. This forms a Bayesian network aimed at identifying the actuator errors that contribute to defects in completed products, as illustrated in Figure 3.6. Here the actuator error is a real number that characterizes the difference between required task execution and real execution. Such errors are challenging to detect in flow-based manufacturing without immediate, NP-hard feedback from every downstream robot. It's denoted by error for the rest of this section. By doing so, the Bayesian network provides a scalable solution for reasoning about which robots are contributing to defects, adapting to the number of production robots within the MRS.

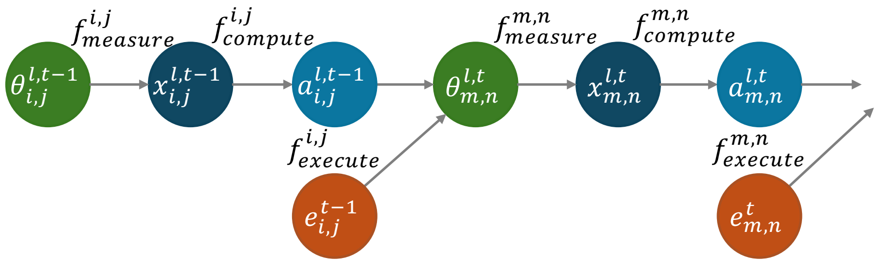


Figure 3.6 Bayesian network that describes a segment of cyber-physical interaction.

Initiated by [76, 75], the MRS operates with tight time synchronization, wherein each task execution occurs within a single time slot, denoted by t. This synchronization is achieved through multiplexing considerations in MRTA [76]. The task execution process for a production robot is modeled as follows: During each time slot t, the nth type-m production robot in the MRS, represented by $R_{m,n}$, receives a raw material or an unfinished product-l with a state quantified by $p_{m,n}^{l,t} \in \Theta_p$, where Θ_p signifies the product state space. Subsequently, $R_{m,n}$ employs its onboard sensors to obtain a noisy measurement $x_{m,n}^{l,t}$, denoted by $f_{\text{measure}}^{m,n}(\cdot)$ in (3.4a), in which the measurement noise is characterized by additive Gaussian noise $\zeta_{m,n}^{l,t}$.

Next, given that error is unobservable, each production robot estimations the error belief of the upstream robot, using social information, which is exchanged via wireless communications within a partially connected network, constituting collaborative social learning. This will be detailed in Section 3.3.2.1. $R_{m,n}$ calculates the parameter in task execution, denoted as action $a_{m,n}^t \in \mathcal{A}$, based on the measurement $x_{m,n}^{l,t}$ and the MRS error belief, $\mathbf{b}_{m,n:sys}^t$. This calculation is represented by $f_{\text{compute}}^{m,n}(\cdot,\cdot)$ in (3.4b).

However, due to imperfections in wireless communications and the network's partial connectivity, social information may be incomplete. Therefore, the derivation of $f_{\text{compute}}^{m,n}(\cdot,\cdot)$ necessitates the application of AI computing. This ensures the resulting actions are not only coordinated but also capable of adapting to real-time changes in production flows, which will be detailed in Section 6.1.

Finally, when $R_{m,n}$ executes $a_{m,n}^t$ (employing actuators to process product-l and alter $p_{m,n}^{l,t}$), denoted by $f_{\text{execute}}^{m,n}(\cdot,\cdot,\cdot)$ in (3.4c), an error with respect to the hypothetical global reference from the required task arises due to unobservable error. This error is represented by $e^t_{m,n} \in [-\gamma_R, \gamma_R]$ as in (3.4c), with γ_R defining the error boundary resulting from error. When $e^t_{m,n} \notin [-\gamma_R, \gamma_R]$, $R_{m,n}$ produces defective products, leading to point failures and necessitating maintenance. Without loss of generality, task execution is characterized as a linear combination of measurement, action, and errors, as expressed in (3.4c). Here, the action $a^t_{m,n}$ specifically indicates the coordinated action for better productivity performance. The quantitative action for task assignment is related to product design and robotics, which is beyond the scope of this study. The above model characterizes the unobservable uncertainties. Although the error resulting from error, $e^t_{m,n}$, affects task execution, the computation of action cannot take it into consideration.

$$x^{l,t}_{m,n} = f^{m,n}_{\text{measure}}(p^{l,t}_{m,n}) = p^{l,t}_{m,n} + \zeta^{l,t}_{m,n} \sim \mathcal{N}(0, \sigma_{\text{measure}}) \tag{3.4a}$$

$$a^t_{m,n} = f^{m,n}_{\text{compute}}(x^{l,t}_{m,n}, \mathbf{b}^t_{m,n:sys}) \tag{3.4b}$$

$$\begin{aligned} p^{l,t+1}_{m',n'} &= f^{m,n}_{\text{execute}}(p^{l,t}_{m,n}, a^t_{m,n}, e^t_{m,n}) \\ &= f^{m,n}_{\text{execute}}(p^{l,t}_{m,n}, f^{m,n}_{\text{compute}}(f^{m,n}_{\text{measure}}(p^{l,t}_{m,n}), \mathbf{b}^t_{m,n:sys}), e^t_{m,n}) \\ &= p^{l,t}_{m,n} + \zeta^{l,t}_{m,n} + a^t_{m,n} + e^t_{m,n} \end{aligned} \tag{3.4c}$$

Utilizing the Naive Bayes approach [8, 17] and under the assumption that the three processes—measurement, computation, and execution—are independent, Here we can define the joint probability of $p^{l,t}_{m,n}$, $x^{l,t}_{m,n}$, $a^t_{m,n}$, and $e^t_{m,n}$ as detailed in Equation (3.5), where α is a normalization factor. Equation (3.5a) provides the full joint distribution. Given that the measurement and task execution processes are independent, it is possible to estimate the errors $e^t_{m,n}$ that lead to the observed outcomes, also known as effects or events. These are represented in boldface vector form as $\mathbf{x}^{l,t}_{m,n}$ and $\mathbf{p}^{l,t+1}_{m',n'}$, and can be estimated using Equations (3.5b) and (3.5c). It is important to note during these estimations that the conditional probabilities are explicitly defined in the task execution process in Equation (3.4). Specifically, $Pr(x_j|\zeta^{l,t}_{m,n})$ is defined by $f^{m,n}_{\text{measure}}(\cdot)$ in Equation (3.4a), and $Pr(p_j|e^t_{m,n}, x^{l,t}_{m,n})$ is defined by $f^{m,n}_{\text{execute}}(\cdot,\cdot,\cdot)$ in Equation (3.4c). These estimations enable each robot to discern whether a measured error originates from an upstream robot's process or from a process further upstream, thereby facilitating a crucial reasoning process within the multi-robot system.

$$Pr(\zeta^{l,t}_{m,n}, e^t_{m,n}, x^{l,t}_{m,n}, p^{l,t+1}_{m',n'}) = Pr(\zeta^{l,t}_{m,n})Pr(e^t_{m,n})Pr(x^{l,t}_{m,n}|\zeta^{l,t}_{m,n})Pr(p^{l,t+1}_{m',n'}|e^t_{m,n}) \tag{3.5a}$$

$$Pr(\zeta^{l,t}_{m,n}|\mathbf{x}^{l,t}_{m,n}) = \alpha Pr(\zeta^{l,t}_{m,n}) \prod_{x_j \in \mathbf{x}^{l,t}_{m,n}} Pr(x_j|\zeta^{l,t}_{m,n}) \tag{3.5b}$$

$$Pr(e^t_{m,n}|x^{l,t}_{m,n}, \mathbf{p}^{l,t+1}_{m',n'}) = \alpha Pr(e^t_{m,n}) \prod_{p_j \in \mathbf{p}^{l,t+1}_{m',n'}} Pr(p_j|e^t_{m,n}, x^{l,t}_{m,n}) \tag{3.5c}$$

3.2.2.2 Planning with Graph Search

Planning in AI involves determining a sequence of actions to attain a specific goal within a discrete, deterministic, static, and fully observable environment [89]. This general problem-solving technique finds applications across various domains, including robotics, scheduling, and game-playing.

In most planning algorithms, a world representation is essential. This representation includes the potential states the world can assume, the actions available and the rewards to be gained. In the context of cyber-physical AI agents, this representation is encapsulated by the domain model and perception mechanisms.

RL introduces a specialized form of planning tailored for agents operating in Markov Decision Processes (MDPs) [100]. An MDP provides a stochastic model of an environment where an agent performs actions to transition between states and receives rewards. RL aims to discover a policy that optimizes the expected reward over time.

Planning within RL is commonly approached in two ways: model-based planning and model-free planning. Model-based planning entails constructing an explicit model of the MDP and utilizing it to plan action sequences. Although generally more efficient, this approach can be challenging to implement. On the other hand, model-free planning does not create a model of the MDP; instead, it learns a policy directly from interactions with the environment. While this method is typically less efficient, it offers greater generality and applicability to complex problems.

This study focuses on planning techniques grounded in the field of AI, as opposed to planning approaches that are rooted in RL. In the field of AI, the use of a domain model brings structure to the planning process by characterizing the applicable rules, actions, states, and objectives specific to a given problem space. The inclusion of a domain model serves to significantly boost the efficiency and effectiveness of AI planning algorithms.

One of the key benefits of using a domain model is that it ensures all generated plans are consistent and coherent, as they adhere to the same set of rules and constraints. This uniformity is invaluable for generating reliable solutions. Additionally, the reusability of a domain model across different instances of problems within the same space offers the advantage of saving both time and computational resources.

Domain models also aid in the abstraction of complex actions and states, transforming them into simpler, more manageable forms. This level of abstraction enables the AI system to generalize the model, making it applicable across a variety of scenarios within the same domain. A well-constructed domain model further optimizes the planning process by swiftly filtering out solutions that either have diminished utility or fail to meet specific constraints, thereby enhancing the efficacy of AI agents.

This section aims to explore these various facets of planning in AI, particularly emphasizing the integral role that domain models play in enhancing the planning process.

This section delves into the planning strategies employed by an autonomous mobile robot (AMR) for executing transportation tasks, using a domain model for guidance. Specifically, the domain model used here is represented by the multi-floor transportation model discussed in Section 3.1.3. Within this framework, the domain model $\mathsf{G}_{\text{floor}}$ and a pre-established number of plans, denoted as κ_{N}, are used to improve computational efficiency. When a task, denoted as $\mathtt{task}^m$, is assigned to AMR-k, the robot utilizes a multi-path planning method to devise its course of action. This planning method is an enhancement of Yen's algorithm, as elaborated by [4], and is referred to as `Planning` in Algorithm 3.2.

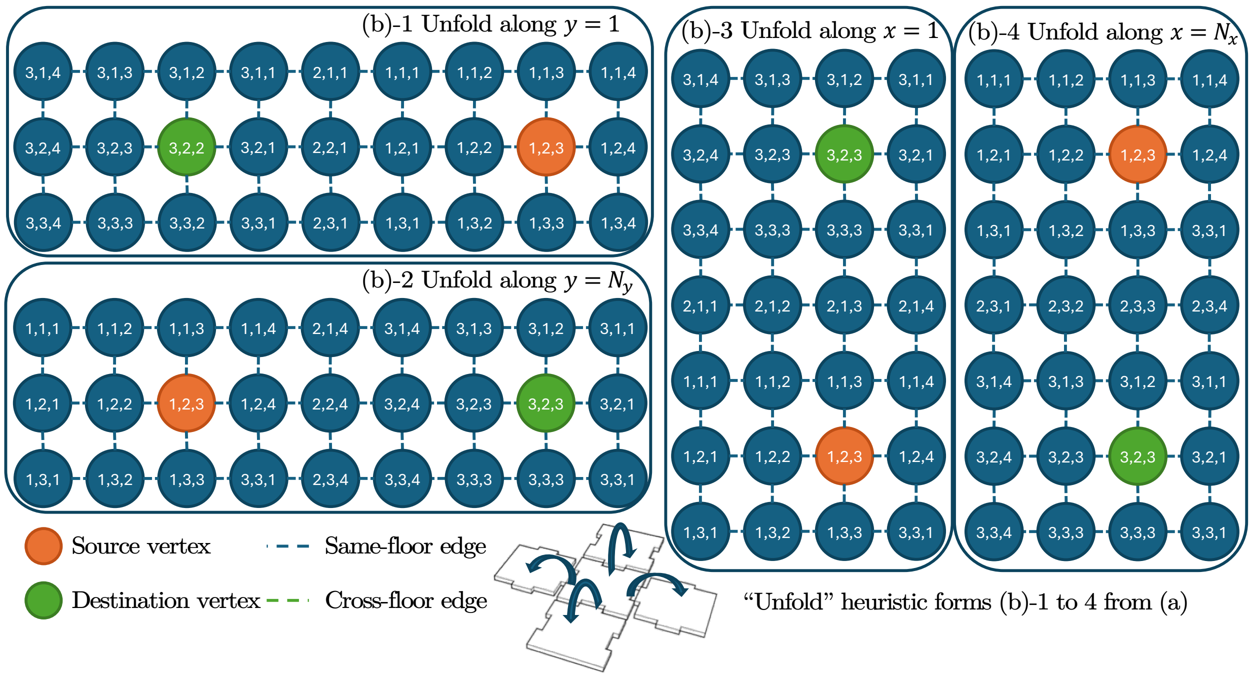

(b)-1 Unfold along $y = 1$
(b)-2 Unfold along $y = N_y$
(b)-3 Unfold along $x = 1$
(b)-4 Unfold along $x = N_x$
Source vertex
Destination vertex
Same-floor edge
Cross-floor edge
"Unfold" heuristic forms (b)-1 to 4 from (a)

Figure 3.7 "Unfold" heuristic in planning with multi-floor domain model.

The heuristic known as "unfold" is graphically illustrated in Figure 3.7. This heuristic plays a critical role in generating four acyclic sub-graphs from the cyclic graph G_{floor}. Such a transformation is vital because Yen's algorithm is specifically tailored for acyclic graphs. To provide more details, if both the current location of the AMR, denoted as v_{f_k,x_k,y_k}, and its target location, v_{f_t,x_t,y_t} (which could either be a pickup node with coordinates $f_t = f_s, x_t = x_s, y_t = y_s$ or a delivery node $f_t = f_d, x_t = x_d, y_t = y_d$), are on the same floor, then the actor algorithm uses only the sub-graph corresponding to that specific floor for multi-path planning. This is clarified in lines 3-5 of Algorithm 3.2.

$$\mathbf{V}_{\text{sub}} = \{(f_i, x_i, y_i) \in \mathbf{V}_{\text{floor}} | f_i = f_k \text{ or } f_i = f_t\}, \tag{3.6a}$$

$$\mathbf{V}_{\text{sub1}} = \mathbf{V}_{\text{sub}} \cup \{(f_i, x_i, y_i) \in \mathbf{V}_{\text{floor}} | x_i = 1\}, \tag{3.6b}$$

$$\begin{aligned}\mathbf{E}_{sub1} =& \{((f_i, x_i, y_i), (f_j, x_j, y_j)) \in \mathbf{E}_{floor} \\ & |(f_i, x_i, y_i), (f_j, x_j, y_j) \in \mathbf{V}_{\text{sub1}}\}\end{aligned} \tag{3.6c}$$

$$G_{sub1} = (\mathbf{V}_{sub1}, \mathbf{E}_{sub1}) \tag{3.6d}$$

$$\mathbf{V}_{\text{sub2}} = \mathbf{V}_{\text{sub}} \cup \{(f_i, x_i, y_i) \in \mathbf{V}_{\text{floor}} | x_i = N_x\}, \tag{3.6e}$$

$$\ldots \tag{3.6f}$$

When the AMR's current and target locations are on two different floors, the actor algorithm accommodates this by incorporating both floors and the nodes that connect them at positions $x = 1$, $x = N_x$, $y = 1$, and $y = N_y$. By doing so, the algorithm produces four acyclic sub-graphs, which are then utilized for multi-path planning. This process is demonstrated in the `Unfold` method, as outlined in Algorithm 3.2 and formally described by (3.6).

Moreover, integrating a domain model into the planning process lends a greedy heuristic to the actor module. This is because Yen's algorithm inherently generates the shortest plans first, ensuring that AMRs execute tasks by following either optimal or near-optimal paths. These shortest paths are represented as walks in the multi-floor models and are denoted by $\boldsymbol{\rho}_k^m$, as defined in Section 3.1.

3.2.3 Learn to Adapt to Uncertainties

As previously introduced in Section 3.1.1, the data generated from MRS operations for subsequent AI computations take the form of time series, also referred to as trajectories. These time series are particularly challenging to analyze due to their non-stationary and non-

Algorithm 3.2: Planning with Domain Model

```
def Plan($v_{f_k,x_k,y_k}$, $G_{floor}$, $\mathsf{task}^m$, $\kappa_N$, $\kappa_k$):
    Result: $\{\rho_k^m, \ldots\}$
1   $v_{f_t,x_t,y_t} \leftarrow \mathsf{task}^m$
2   if $f_k == f_t$:
3       $\mathbf{V}_{sub} = \{(f_i, x_i, y_i) \in \mathbf{V}_{floor} | fi = f_k\}$
4       $\mathbf{E}_{sub} = \{((f_i, x_i, y_i), (f_j, x_j, y_j)) \in \mathbf{E}_{floor} | f_i = f_j = f_k\}$
5       $G_{sub1} = (\mathbf{V}_{sub}, \mathbf{E}_{sub})$
6   else:
7       $\{G_{sub1}, \ldots, G_{sub4}\} \leftarrow$ Unfold $(G_{floor}, f_k, f_t)$
8   $\{\rho_k^m, \ldots\} \leftarrow$ Yen's_algorithm $(\{G_{sub}, \ldots\}, v_{f_k,x_k,y_k}, v_{f_t,x_t,y_t}, \kappa_N)$
```

deterministic nature, which arise from time-varying production demands and burst errors in actuators and wireless communications.

In the field of time series analysis [73], the terms “non-stationary” and “non-deterministic” are sometimes used interchangeably, but they have distinct meanings. A stationary time series has statistical properties such as mean, variance, and autocorrelation that remain constant over time. Conversely, a non-stationary time series has statistical properties that change over time. In a non-deterministic time series, even full knowledge of past values cannot perfectly predict future values due to random shocks or disturbances affecting the series.

In the case of MRS, time-varying production demands cause the reconfiguration of the system, leading to changes in statistical properties and hence, non-stationarity. Meanwhile, burst errors from actuators and wireless communications introduce random disturbances, contributing to the time series’ non-deterministic nature.

In technical terms, a non-stationary time series differs from a strictly stationary process. The latter is characterized by statistical properties that remain consistent across all time periods. On the other hand, a non-deterministic time series is distinct from a Markov process, which is a specific type of stochastic process where the future state is solely dependent on the current state and is independent of past states.

Stochastic Gradient Descent (SGD) stands out as one of the most widely used optimization algorithms for training neural networks. Its main advantage is its capability to navigate the complex, high-dimensional landscapes typical of neural network loss functions. When applied to systems that are both non-stationary and non-deterministic, SGD offers unique benefits. This section delves into the technical facets that make SGD particularly well-suited for such challenging contexts.

The first aspect is *stochasticity.* In its most basic form, standard Gradient Descent (GD) computes the gradient of the loss function with respect to all the samples in the training set and then updates the model parameters based on this gradient. In contrast, SGD approximates the true gradient by considering just a single (or a small batch of) example(s) at each iteration. This stochasticity introduces noise into the optimization process, which can be both a curse and a blessing. On one hand, this noise can prevent the algorithm from settling into sub-optimal local minima in non-convex loss landscapes common to neural networks. On the other hand, this same noise can also prevent the algorithm from reaching the exact minimum, settling instead in a region around the minimum.

The second aspect is *non-stationarity.* In non-stationary environments, where the data distribution can change over time, the stochastic nature of SGD is particularly beneficial. It allows the model to adapt more quickly to new data, making it more robust to shifts in the data distribution. This is crucial in scenarios like online learning or when the network has to adapt to new, unseen data quickly. Because each gradient update is calculated on-the-fly with the latest available data, the model remains adaptive to non-stationary patterns in the data.

The third aspect is *non-determinism.* In systems with inherent randomness or variability, capturing all the potential nuances with a deterministic model may be impractical. In such cases, the stochastic nature of SGD provides a form of regularization, preventing the model from overfitting to the training data. This means that the model can generalize better to new, unseen data.

The fourth aspect is *learning rate scheduling.* In environments that are both non-stationary and non-deterministic, the selection of an appropriate learning rate becomes a critical factor for the performance of SGD. Adaptive learning rate methods such as AdaGrad, RMSprop, and Adam can be integrated into SGD to dynamically adjust the step size. Additionally, within the context of a smart factory, domain knowledge regarding statistical property changes induced by MRS reconfigurations can be leveraged to schedule the learning rate adaptively. These enhancements not only make SGD more resilient to shifts in data distribution but also facilitate quicker convergence of the optimization algorithm.

The fifth aspect is *replay buffer reset.* Experience replay is a technique commonly employed in reinforcement learning to enhance both the stability and sample efficiency of the learning process. It involves storing the agent's historical experiences in a designated replay buffer. Instead of relying solely on real-time interactions with the environment, the agent draws samples from this replay buffer to update its policy or value functions. This strategy helps break the temporal correlations often present in sequential data, enabling the agent to learn from a more diverse set of experiences. Periodically resetting the replay buffer ensures that the agent's learning process remains aligned with the evolving state of the environment, thus maintaining a representative distribution of transitions for effective learning.

In reinforcement learning, various algorithms target different optimization objectives. For example, value-based methods like Q-learning or DQN aim to minimize the difference between the predicted value function $Q(s, a; \theta)$ and the expected return or target, often represented as $Q'(s, a)$. The Mean Squared Error (MSE) loss function, given by Equation (3.7a), is commonly used for this purpose. Here, θ are the parameters of the neural network approximating Q. Stochastic Gradient Descent (SGD) is employed to update these weights as shown in Equation (3.7d). Similarly, when optimizing the state-value function $V(s)$, the goal is to minimize the difference between the estimated value $V(s; \theta)$ and the expected return $V'(s)$, also typically using MSE as indicated by Equation (3.7b). In contrast, policy-based methods like Policy Gradients target the policy $\pi(a|s)$ directly. The aim is to maximize the expected return J by tweaking the parameters θ of the policy, as detailed in Equation (3.7c). These policy gradient methods are often combined with value function approximations to reduce variance in gradient estimates. Actor-critic methods integrate both value-based and policy-based paradigms: the critic updates the value function parameters while the actor updates the policy parameters, commonly using SGD or its variants like Adam for both. Moreover, Gradient Ascent can also be used for policy optimization in some algorithms, requiring a simple sign change in the weight update equations (3.7d) and (3.7e).

$$L(\theta) = \mathbb{E}\left[(Q'(s,a) - Q(s,a;\theta))^2\right] \tag{3.7a}$$

$$L(\theta) = \mathbb{E}\left[(V'(s) - V(s;\theta))^2\right] \tag{3.7b}$$

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)] \tag{3.7c}$$

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^{t} - lr\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{t}) \tag{3.7d}$$

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^{t} - lr\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{t}) \tag{3.7e}$$

SGD's intrinsic stochasticity makes it a robust and versatile algorithm, particularly suited for training neural networks on non-stationary and non-deterministic data. By leveraging SGD's stochastic nature and combining it with advanced learning rate scheduling and specialized algorithmic modifications, one can effectively tackle the challenges posed by such complex environments.

3.3 Cyber-Physical AI Agent Systems

The concept of a cyber-physical AI agent system is anchored in the principles of smart factories and the cyber-physical AI agent approach, as discussed in Section 3.2. In a smart factory environment, the system is designed to be decentralized, self-organized, and self-optimized. These features are implemented using edge computing and distributed computing techniques within a MRS.

Edge computing serves as a shared computational and memory resource for the MRS, effectively dealing with shared data processing needs. Additionally, the distributed computing capabilities of individual robots within the MRS are leveraged to address the challenge of system scalability. Furthermore, robot-to-robot communication protocols are utilized to facilitate collaboration among agents in the MRS.

This approach is particularly well-suited for decentralized, MRS-driven smart factories. Unlike edge computing or virtual AI agents, which do not interact directly with the physical domain, the MRS has the ability to directly learn from and optimize its interactions with the environment through a trial-and-error process. However, it's worth noting that individual robots within the MRS typically have limited computational power. While they can optimize their respective tasks, they often lack the capability for global system coordination and optimization.

One significant advantage of this approach is that it facilitates more effective system-wide coordination than could be achieved through the mere duplication of individual agents. This is further elaborated in Section 3.3.1. Subsequent to this, Section 3.3.2 provides detailed discussions on two key aspects: collaborative prediction among production robots and collaborative coordination among transportation robots, each covered in separate subsections.

3.3.1 The Whole Is Greater than the Sum of Its Parts

In multi-agent environments, the dynamics often involve a blend of both collaboration and competition, mirroring the complexities of real-world systems such as biological ecosystems, social networks, or economic markets. A cyber-physical AI agent system is particularly beneficial in navigating these complex collaborative-competitive landscapes, transcending the capabilities of isolated, single intelligent agents.

When resources are limited, a natural competition arises among agents. However, the effective utilization of these scarce resources often necessitates collaboration. This approach facilitates this by enabling more coordinated resource allocation compared to systems that merely duplicate single intelligent agents.

Agents in the system are typically oriented toward both shared and individual goals. While the pursuit of individual objectives might lead to competition, working towards common goals naturally promotes collaboration. The cyber-physical AI agent system framework enhances this balance by promoting intelligent collaboration where needed, without sacrificing beneficial competition.

Partial observability of the system often complicates the agents' decision-making processes. In such cases, the collaborative mechanisms within the system prove advantageous, helping to mitigate the uncertainties arising from the limited information of peer agents.

Communication serves as a foundation for the efficacy of this approach, especially in mixed collaborative-competitive settings. Agents can use explicit communication to coordinate actions, thereby optimizing the attainment of shared objectives. They can also share critical information about environmental changes, new constraints, or shifts in goals, allowing each agent to make more informed decisions.

Resource conflicts are inevitable in environments where multiple agents vie for the same limited assets. In a cyber-physical AI agent system, communication can streamline negotiation processes to rationally resolve these conflicts, thus preserving the system's collaborative goals.

Moreover, sustained communication within the system can help build reputations for reliability and trustworthiness among agents. This facilitates a convention of effective collaboration for future interactions. Over time, this regular communication also helps establish communal norms or "unwritten rules" [97], which serve as guiding principles in scenarios where formal rules may be lacking or ambiguous.

By excelling in these aspects, a cyber-physical AI agent system offers a significant advancement over systems that simply duplicate single agents, presenting a more nuanced and effective approach for managing the intricate dynamics of complex collaborative-competitive environments.

3.3.2 Collaboration with Communications

This section is divided into two subsections that focus on the collaborative aspects of cyber-physical AI agent systems facilitated by communication mechanisms. The collaboration is primarily manifested in the form of "agreements" among agents concerning specific variable values.

Section 3.3.2.1 explores how production robots collaborate to predict actuator errors through social learning. This approach leverages scalable social communications within an MRS and particularly thrives under conditions where the available information is incomplete and derived from a partially connected social wireless topology.

On the other hand, Section 3.3.2.2 discusses the collaborative coordination among transportation robots. This coordination is facilitated by scalable collision-instance-based communications and is executed within the framework of an innovative Multi-Agent Reinforcement Learning (MARL) architecture. This subsection focuses on enabling effective collaboration in a mixed collaborative-competitive environment.

Both of these approaches demonstrate the capabilities of the cyber-physical AI agent system approach in facilitating efficient collaborations among agents, through scalable communications, even in scenarios fraught with challenges such as partial information and resource constraints.

3.3.2.1 Collaborative Prediction with Social Learning

On top of social communications defined by CPMRS in Section 3.1.1.2, as well as the Bayesian network-based reasoning in Section 3.2.2.1, social learning forms production robot errors $\mathbf{e}^t_{sys}$ as the local belief $\mathbf{b}^t_{m,n:sys}$ from incomplete and time-inefficient information in a best-effort manner. Social learning takes a segment of production flow in G^t_{phy} as domain knowledge and performs estimation under measurement noises, incomplete and time ineffective information. Therefore, the local belief $\mathbf{b}^t_{m,n:sys}$ can also be incomplete.

In contrast to social learning methods that share beliefs instead of evidence (measurements), $R_{m,n}$ formulates the local belief of upstream robot in production flow $R_{i,j}$ as $b^t_{m,n:i,j}$ following a Combine-then-Adapt (CTA) diffusion strategy [94]. This is achieved by (3.8a) that aggregates local evidence $\mathbf{x}_{m,n}$ in form of $f^{m,n}_{\text{measure}}(\mathbf{p}_{m,n})$ and social evidence of $R_{i,j}$, $\mathbf{y}_{i,j}$ as $\mu^t_{m,n:i,j}(\varphi^t_{i,j})$, $\varphi^t_{i,j} \in [-\gamma_R, \gamma_R]$. $R_{m,n}$ then computes the local belief of $R_{i,j}$'s misalignment $b^t_{m,n:i,j}$ through maximum likelihood estimation given by (3.8b). For simplicity, both (3.8a) and (3.8b) use boldface vectors instead of time-specific values (t in superscripts) or product-specific values (l in superscripts) since social learning only concerns time index of relative errors $e^t_{i,j}$ and error is not product-sensitive.

$$\mu^t_{m,n:i,j}(\varphi^t_{i,j}) = \frac{\varphi^t_{i,j} f^{m,n}_{\text{measure}}(\mathbf{p}_{m,n}) f^{i,j}_{\text{execute}}(\mathbf{p}_{i,j}, \mathbf{a}_{i,j}, \varphi^t_{i,j}) f^{m,n}_{\text{compute}}(\mathbf{x}_{i,j}, \mathbf{b}_{i,j:sys})}{\sum_{\varphi^t_{i,j}{}' \in [-\gamma_R, \gamma_R]} \varphi^t_{i,j}{}' f^{m,n}_{\text{measure}}(\mathbf{p}_{m,n}) f^{i,j}_{\text{execute}}(\mathbf{p}_{i,j}, \mathbf{a}_{i,j}, \varphi^t_{i,j}{}') f^{m,n}_{\text{compute}}(\mathbf{x}_{i,j}, \mathbf{b}_{i,j:sys})} \tag{3.8a}$$

$$b^t_{m,n:i,j} = \underset{\varphi^t_{i,j}}{\text{argmax}} \log(\mu^t_{m,n:i,j}(\varphi^t_{i,j}) | \mathbf{x}_{m,n}, \mathbf{y}_{i,j}, \mathbf{a}_{i,j}) \tag{3.8b}$$

$$\mathbf{b}^t_{m,n:sys} = (\mathbf{b}^t_{m,n:1,1}, \dots, \mathbf{b}^t_{m,n:m,n}, \dots, \mathbf{b}^t_{m,n:M,N_M}) \tag{3.8c}$$

As defined in (3.4), in social learning, $f^{m,n}_{\text{measure}}$, $f^{m,n}_{\text{compute}}$ and $f^{m,n}_{\text{execute}}$ are transferred into probability distributions. To solve (3.8a), a prior distribution $f^{m,n}_{\text{compute}}(\mathbf{x}_{i,j}, \mathbf{b}_{i,j:sys})$ is needed. This study assumes that when MRS performs social communications, MRS also takes "straightforward" actions, which means $f^{m,n}_{\text{compute}}(\cdot, \cdot)$ does not use error beliefs to compute actions. As $f^{m,n}_{\text{measure}}(\cdot)$ is modeled Gaussian, both $f^{m,n}_{\text{compute}}(\cdot, \cdot)$ and $f^{m,n}_{\text{execute}}(\cdot, \cdot, \cdot)$ take $f^{m,n}_{\text{measure}}(\cdot)$'s outputs, all three distributions are Gaussian. In addition, social learning is performed periodically instead of every time slot so that AI computing can take effect.

After establishing the topology of the cyber domain network, we can infer that production robot $R_{m,n}$ may potentially receive measurements from all other robots, but the quality and frequency of these measurements may vary due to imperfect wireless communications. Consequently, $R_{m,n}$ attempts to perform social learning for all robots in the MRS, utilizing incomplete and time-ineffective information to build the local belief $\mathbf{b}^{t}_{m,n:sys}$ that becomes increasingly comprehensive over time as given by (3.8c). Furthermore, changes in the physical domain topology, such as alterations to the production flow by MRTA, introduce further diversity into social communications, potentially accelerating the completeness of social measurements for the MRS.

3.3.2.2 Predictive Coordination in Complex Collaborative-Competitive Environment

The architecture of the proposed solution—a decentralized, off-policy Multi-Agent Reinforcement Learning (MARL) algorithm referred to as "Multi-Agent Actor-Mixed Critics". MARL, as outlined in the work of Li et al. [61], is a specialized subfield of RL that aims to equip multiple agents with the capabilities to learn and interact within either collaborative or competitive environments. This domain emerges at the confluence of reinforcement learning and game theory and is specifically designed to address the intricate dynamics and potential conflicts that arise when multiple autonomous agents co-exist and make decisions in a shared setting.

In a typical MARL framework, each agent operates as an autonomous unit capable of perceiving its environment, executing actions, and receiving consequential feedback in the form of rewards or penalties. These agents are designed to refine their individual policies over time, all while taking into account the dynamically evolving behaviors and policies of other agents within the same environment. The central goal of MARL is to strike an optimal balance between exploration—investigating new strategies—and exploitation—utilizing known effective strategies. This ensures that agents not only optimize their individual performances but also adapt responsively to the shifting tactics employed by their peer agents.

In the cyber-physical AI agent system approach designed for transportation robots, agents operate in a complex, dynamic environment characterized by non-stationarity and non-determinism, as depicted by Figure 3.8. Each agent possesses an actor module for planning upon receiving a transportation task assignment, utilizing predictions from both the shared global critic and its own local critic to decide a collision-avoidant paths execution plan. Given that this planning doesn't consider the plans of other agents, potential collisions, when they arise, are resolved through token-passing among the colliding agents. These agents perceive their environment with the aid of the multi-floor model, generating global states, global rewards, local states, and local rewards. Here, agents do not possess the ability to directly control or predict the multi-floor model, the transportation tasks, or any task reconfigurations. This adds layers of complexity as, from each agent's perspective, not only are the behaviors of peer agents unpredictable, but the nature of the transportation tasks themselves is also inherently unpredictable.

When an agent-k receives a task assignment, denoted as task^m, its actor module engages in planning activities. The planning process takes into consideration the multi-floor model G_{floor}, its own local critic Q_k, and the global critic V_g. The resulting plans are designed to either be collision-free or to minimize the risk of collisions. Furthermore, these plans aim for optimal or near-optimal delays in task execution.

To enhance the readability and consistency of this section, the narrative uses g and l as subscripts for variables associated with the global and local critics, respectively. It should be noted that the critic variables do not differentiate between different time slots.

The global critic, denoted as V_g, is a shared value function among all agents and is hosted via edge computing for accessibility. This critic is responsible for predicting the global reward, which quantifies an agent's contribution to the system's collective objective or the global utility. The state vector for the global critic, denoted as $s_{g,k}$, is tailored to each agent and encapsulates the current status of that agent's task execution relative to the system's collective objective.

Unlike traditional centralized critic methods frequently encountered in existing literature, this approach features an agent-specific global reward $r^t_{g,k}$. This reward offers a nuanced

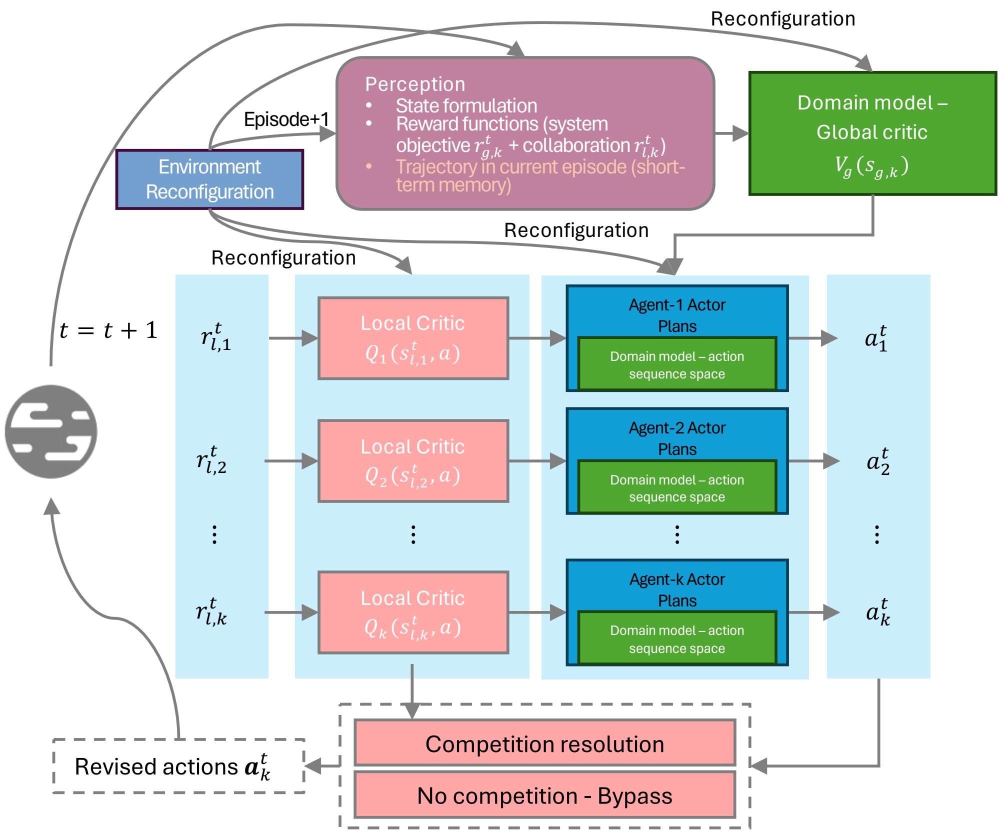


Figure 3.8 Architecture of the multi-agent actor-mixed critics with the knowledge-based domain model.

understanding of how each agent's actions individually contribute to the attainment of the global utility.

Each agent also hosts its own local critic, Q_k, and engages in distributed training for this critic. This local critic is geared toward guiding agent-k in avoiding potential collisions by predicting the associated penalties, referred to as local rewards, for all conceivable actions within its action space $\mathcal{A}$.

In a departure from the global state, the local state $s_{l,k}$ is designed to be agent-specific, representing the current status of an individual agent's task execution in relation to its localized objectives. Similar to the global rewards, these local rewards are also agent-specific, providing a measure of how each agent's actions contribute to fulfilling its own localized goals.

In the collaborative-competitive setting, the roles of the global and local critics are distinct yet complementary. The global critic V_g and its corresponding global rewards focus on the collaborative aspects, driving agents toward maximizing the collective global utility. This serves as the system's collaborative component.

Conversely, the local critic Q_k and the associated local rewards act as the competitive component, capturing the competitive dynamics among agents. During instances of competition, the involved agents engage in a distributed process to resolve the competition and reach a competitive equilibrium. Finding the global utility optima is closely related to the individual utilities of agents and their competitive equilibrium, which is proved by Theorem 2.3.5 in [97]. This competition resolution may result in a modified action vector, one that aligns the individual actions of all agents with the environmental constraints bringing the competition in the first place.

Each agent is also equipped with an actor module that is responsible for both planning and executing actions in line with the guidance from both the shared global critic and the agent's own local critic. This dual-guidance system operates effectively thanks to a short-term memory mechanism for storing plans and an advantage function [95]. The advantage function provides a relative valuation of possible actions within the agent's action space, based on assessments from both the global and local critics.

Collaborative training of shared global critic contributes to sample efficiency. Edge computing, which hosts the global critic, lacks the capability for task execution. As a result, transitions for training the global critic are collaboratively collected by all agents. Considering the execution of task^m as an episode, each agent compiles two global critic transitions—one from the pickup path and the other from the delivery path. By the episode's completion, two paths $\boldsymbol{\rho}_k^m$ and two cumulative returns R_k^m are collected, thus constituting two global state vectors and their corresponding global rewards. It's important to note that, being a state-value function, the global critic doesn't necessitate actions within its transitions. For the global critic, it is implemented by a neural network as the function approximator, characterized by its weights $\boldsymbol{\theta}_g^t$.

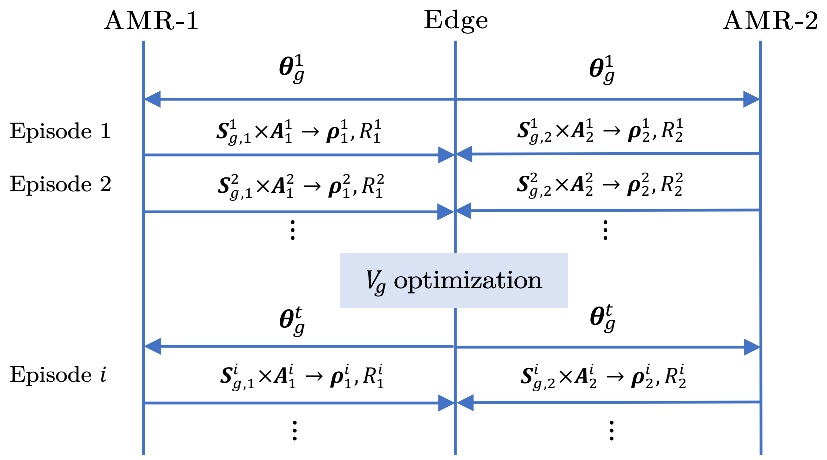

AMR-1
Edge
AMR-2
$\boldsymbol{\theta}_g^1$
$\boldsymbol{\theta}_g^1$
Episode 1
$\boldsymbol{S}_{g,1}^1 \times \boldsymbol{A}_1^1 \to \boldsymbol{\rho}_1^1, R_1^1$
$\boldsymbol{S}_{g,2}^1 \times \boldsymbol{A}_2^1 \to \boldsymbol{\rho}_2^1, R_2^1$
Episode 2
$\boldsymbol{S}_{g,1}^2 \times \boldsymbol{A}_1^2 \to \boldsymbol{\rho}_1^2, R_1^2$
$\boldsymbol{S}_{g,2}^2 \times \boldsymbol{A}_2^2 \to \boldsymbol{\rho}_2^2, R_2^2$
V_g optimization
$\boldsymbol{\theta}_g^t$
$\boldsymbol{\theta}_g^t$
Episode i
$\boldsymbol{S}_{g,1}^i \times \boldsymbol{A}_1^i \to \boldsymbol{\rho}_1^i, R_1^i$
$\boldsymbol{S}_{g,2}^i \times \boldsymbol{A}_2^i \to \boldsymbol{\rho}_2^i, R_2^i$

Figure 3.9 An example of decentralized sample collection with 2 agents.

As illustrated in Figure 3.9, the edge computing system, via robot-to-infrastructure (R2I) communications, consistently updates $\boldsymbol{\theta}_{global}$ for all agents. In turn, at the completion of each episode, every AMR transmits two transitions to the edge server. Consequently, the incorporation of additional AMR directly augments the number of transitions available for training the global critic, rendering the training both scalable and efficient. The optimization of the global critic employs gradient ascent, utilizing mean squared loss [64] and experience replay with unified sampling [31]. The trajectory (or path) $\boldsymbol{\rho}_k^m$ accounts for the sequential global rewards $r_{g,k}^t$ realized at each step.

Chapter 4: Hypergraph Search for Productive and Energy Effective Real-time Multi-Robot Task Allocation

This is my initial trial of factored representation ([89] Chapter 2.4.7) of environment or knowledge, to address the challenges in Multi-Robot Task Allocation (MRTA) in a smart factory. Traditional graphical methods often rely on exhaustive search algorithms to handle the complex interrelated constraints and optimize the objective functions. Although effective and guaranteed optimality in most cases, these methods are computationally intensive and do not scale well with the total number of production robots, especially when dealing with a large number of production robots with various multiplexing capabilities.

To overcome these limitations, the hypergraph search algorithm was specifically designed for the hypergraph model. This algorithm efficiently identifies task assignments in a simple path form and offers a well-scaled, constant time complexity, making it suitable for real-time decision-making without compromising optimality.

To evaluate the performance of MRTA, two numerical metrics are defined: throughput, denoted as $\mathcal{T}$, and energy efficiency, denoted as $\mathcal{E}$.

Throughput serves as a measure of the total products manufactured by an MRS within a unit of time, thereby indicating the system's productivity. For a given task assignment $\{\boldsymbol{\rho}^l\}$, where $l = 1, 2, \dots, L$, the number of time slots, τ^l, required to manufacture product-l is given by ν as per equation (3.1a), under the assumption of guaranteed-delivery transportation. For consistency with the temporal-spatial MRS model introduced in Section 1.3, here define the time slot as the unit time, as shown in (4.1a). This definition also aids in the immediate evaluation of throughput upon deriving a task assignment from MRTA, consequently reducing the time complexity.

The energy efficiencies are derived from the edges as depicted in (3.1a). Let the function $w(\cdot)$ represent the weight of an inputted edge. Such an edge's weight can be one among ε_h, ε_v, and ε_d. The overall energy consumption, denoted as ε, aligns with the throughput definition, and is defined as the average product of the additive edge weights, as described in (3.1a). The conversion from energy consumption to energy efficiency is achieved simply by taking the reciprocal, as presented in (4.1b).

$$\mathcal{T} = \frac{1}{\tau^1} + \frac{1}{\tau^2} + \cdots + \frac{1}{\tau^L} \tag{4.1a}$$

$$\mathcal{E} = \frac{1}{\varepsilon} = \frac{L}{\sum_{l=1}^{L}(\sum_{i,j\in\rho^l} w(e_{i,j;i+1,j+1}))} \tag{4.1b}$$

The proposed algorithm integrates a collection of heuristics, each addressing distinct facets of the MRTA problem as characterized by the hypergraph model:

- *Constraint Satisfaction Heuristic:* Adopted from [89], this heuristic handles production demands and ensures adherence to sequential processing constraints.
- *Maximum Flow Heuristic:* Based on [27], it is employed to optimize the system's throughput.
- *Greedy Heuristic:* This approach is utilized to amplify energy efficiency.

Algorithm 4.1 outlines the hypergraph search process. The inputs for this algorithm are twofold. The MRS hypergraph model $H(\{R_{m,n}\}, E)$ as defined in Section 3.1.2. Production demands for L types of products, denoted as $\{\boldsymbol{\lambda}^l\}$ where $l = 1, 2, \ldots, L$. These demands are elaborated upon in Section 1.3.1 and must satisfy the condition given in (4.2).

$$\sum_{l=1}^{L} \lambda_m^l \leq N_m \omega_m \quad \forall m \in \{1, 2, \ldots, M\} \tag{4.2}$$

The algorithm presented in Algorithm 4.1 works as follows:

Algorithm 4.1: Hypergraph Search

```
   Data: $M, \mathbf{N}, \boldsymbol{\omega}, \varepsilon_h, \varepsilon_v, \varepsilon_d, \{\boldsymbol{\lambda}^l\}, L$
1  residual $\leftarrow M, \mathbf{N}, \boldsymbol{\omega}, \varepsilon_h, \varepsilon_v, \varepsilon_d$
2  $p = 0$
3  while True :
4      if $p >= L$ :
5          $p = 0$
6      else:
7          $p \leftarrow p + 1$
8      task_flow $\leftarrow \boldsymbol{\lambda}^p$
9      $R_{m,n} \leftarrow$ MAKE_CHOICE($\boldsymbol{\lambda}^p$, residual)
10     while task_flow != NULL :
11         current_type $\leftarrow m$
12         task $\leftarrow$ task_flow.pop(first task)
13         if task.type > current_type :
14             candidates $\leftarrow$ all type-($m+1$) hypernodes
15             $R_{m+1,n'}$, edge = MakeChoice(candidates, residual)
16             if $R_{m+1,n'}$ != NULL :
17                 $\boldsymbol{\rho}^p \leftarrow R_{m+1,n'}$, edge
18                 $R_{m,n} = R_{m+1,n'}$
19         else:
20             candidates $\leftarrow$ all type-$m$ hypernodes
21             $R_{m,n'}$, edge = MakeChoice(candidates, residual)
22             if $R_{m,n'}$ != NULL :
23                 $\boldsymbol{\rho}^p \leftarrow R_{m,n'}$, edge
24                 $R_{m,n} = R_{m,n'}$
25     if residual == NULL or $\boldsymbol{\rho}^p$ == NULL :
26         return Solution
27     else:
28         Solution, residual $\leftarrow \boldsymbol{\rho}^p$
```

- *Task Assignments:* Denoted as Solution, these assignments are returned in lines 26 and 28, taking the form of simple paths $\{\boldsymbol{\rho}^l\}$ where $l = 1, 2, \ldots, L$.

- *Residual Hypergraph:* A residual hypergraph, residual $H(\{r(R_{m,n})\}, E)$, is constructed over $H(\{R_{m,n}\}, E)$. In this graph, each hypernode only comprises unassigned blocks (where $\beta^b_{m,n} = 0$), accumulated using the function $r(R_{m,n})$. This hypergraph is initialized in line 1 and updated in line 28 whenever a task assignment $\boldsymbol{\rho}^p$ is determined.

- *Throughput Maximization:* The throughput is optimized using a maximum flow heuristic, as detailed in [27]. This optimization is realized through the while loop beginning at line 3 and the termination conditions from lines 25 to 28.

- *Iterative Search:* The loop at line 3 perpetually seeks task assignments for the products specified in $\{\boldsymbol{\lambda}^l\}$.

- *Task Choice:* The `MakeChoice` function, invoked in lines 9, 15, and 21 with $\mathsf{residual}$ as its input, alongside the termination conditions in lines 25 to 28, adheres to constraint (4.3a). The function $c(R_{m,n}, l)$ provides the count of single task execution capabilities allocated to manufacture product-l in $R_{m,n}$. This ensures robots operate at full multiplexing or stop when their residual multiplexing capabilities can no longer meet additional production demands, preventing task assignments that surpass a robot's multiplexing capacity.

$$\sum_{n=1}^{N_m} r(R_{m,n}) \geq \sum_{n=1}^{N_m} c(R_{m,n}, l) \tag{4.3a}$$

$$\lambda_m^l = \sum_{n=1}^{N_m} c(R_{m,n}, l), \text{ for all the } R_{m,n} \in \boldsymbol{\rho}^l \tag{4.3b}$$

- *Task Flattening:* In lines 1 to 8, the set $\{\boldsymbol{\lambda}^l\}$ is flattened to produce a sequence of single tasks.

- *Initial Robot Choice:* In line 9, the function `MakeChoice` takes $\boldsymbol{\lambda}^l$ as input and selects a robot with positive residual multiplexing capabilities at random. This robot is then designated as the first robot in the solution.

- *Constraint Handling:* The while loop from lines 10 to 24 is influenced by the principles of constraint satisfaction problems [89]. Its role is to address interrelated constraints, particularly the production demands constraint represented by (4.3b). This constraint mandates that task assignments $\boldsymbol{\rho}^l$ align with the quantity of required tasks.

- *Meeting Production and Order Constraints:* To ensure adherence to both the production demands constraint and the ascending order constraint, lines 8 and 12 define task_flow as a sequence. This, in turn, guides lines 14 and 20 to expand the search space exclusively with hypernodes of a consistent type.

In Algorithm 4.1, as the search domain expands based on the variable residual, the function `MakeChoice` invoked in lines 15 and 21, employs a greedy heuristic. This heuristic is designed to select a hypernode with the minimal edge weight, thereby aiming to reduce energy consumption during transportation, exemplifying a one-step greedy decision-making process.

If there are multiple hypernodes with identical edge weight values, the function chooses one at random. This combination of one-step greedy logic and random selection ensures that the time complexity of Algorithm 4.1 remains constant. However, a trade-off exists: while this method offers computational efficiency, it might not always find the globally optimal task assignment, potentially resulting in sub-optimal solutions.

Furthermore, the `MakeChoice` function seamlessly integrates both the constraint satisfaction heuristic and the greedy heuristic. It's architected to operate efficiently, relying on logical flow control and, when necessary, straightforward sorting processes rather than looping.

4.1 Complexity of the Algorithm

To fulfill the real-time requirements of smart factories, the hypergraph search exhibits constant time complexity, taking M, $\mathbf{N}$, $\boldsymbol{\omega}, \varepsilon_h, \varepsilon_v, \varepsilon_d, \{\boldsymbol{\lambda}^l\}, L$ as inputs. This ensures both effectiveness and dual-objective optimization.

Consider the while loop at line 10 in Algorithm 4.1, which iterates over the tasks in task_flow. In the worst-case scenario, a product p requires all types of tasks for its manufacture. This implies that all λ_m^p values in $\boldsymbol{\lambda}^p$ are positive. Consequently, the length of task_flow is given by $\sum_{l=1}^{L} \lambda_m^l$ for all L product demands. Let T_1 represent the time taken for one iteration of the while loop at line 10. Since the execution time is linear with respect to its input, it is $\sum_{l=1}^{L} \lambda_m^l T_1$. All other lines within this loop execute in a constant time C_1. Therefore, the total execution time of this loop is $\sum_{l=1}^{L} L\lambda_m^l T_1 + C_1$.

The while loop at line 3 iterates over the product sequence until its termination condition is satisfied. In the worst-case scenario, every product among the L demanded requires at least one type-1 task. If T_2 denotes the time for one iteration of the while loop at line 3, its execution time, being linear to the input, is $\omega_1 N_1 T_2$. All other lines outside this loop take a constant time C_2 to execute. Thus, the execution time for the while loop at line 3 is $\mathsf{T} = \omega_1 N_1 T_2 + C_2$.

In summary, the execution time of the while loop at line 3 can be expressed as $\mathsf{T} = \omega_1 N_1(\sum_{l=1}^{L} \lambda_m^l T_1 + C_1) + C_2$. Its time complexity is $\mathcal{O}(\omega_1 N_1(\sum_{l=1}^{L} \lambda_m^l))$, indicating constant time with respect to the given inputs.

4.2 Computational Experiments

Computational experiments are conducted on an MRS equipped with 18 production robots. As previously discussed, Algorithm 4.1 does not guarantee a globally optimal solution. To account for this and to analyze the deterministic behavior of the hypergraph search approach, here employed a Monte Carlo simulation, executed 10,000 times. This allows the demonstration of the sub-optimality of the approach through statistical means. The parameters utilized in the experiments are provided in Table 4.1. For all experimental runs, $\varepsilon_h = 1$, $\varepsilon_v = 2$, and $\varepsilon_d = \sqrt{5}$ are set. This choice is grounded in the consideration of Euclidean geometry, along with the aim of achieving practical transportation efficiency and ensuring flexibility within the manufacturing systems.

This section delves into a statistical analysis that highlights the sub-optimality of the proposed hypergraph approach. To achieve this:

Table 4.1 Parameters of real-time MRTA experiments.

Parameters	Small-scale Case	Large-scale Case
M	4	5
$\mathbf{N}$	(3, 3, 4, 3)	(4, 4, 4, 3, 3)
$\boldsymbol{\omega}$	(2, 1, 4, 3)	(2, 1, 4, 3, 2)
L	2	4
$\boldsymbol{\lambda}^a$	(2, 1, 4, 2)	(3, 1, 5, 0, 1)
$\boldsymbol{\lambda}^b$	(1, 0, 6, 3)	(2, 2, 0, 3, 2)
$\boldsymbol{\lambda}^c$		(1, 1, 6, 2, 1)
$\boldsymbol{\lambda}^d$		(1, 0, 4, 3, 2)

- Monte Carlo simulation is performed on the hypergraph approach, focusing on a large-scale case. The parameters for this simulation are detailed in the corresponding column of Table 4.1. This was carried out on the same MRS to maintain consistency.

- Despite using consistent product demands $\{\boldsymbol{\lambda}^a, \boldsymbol{\lambda}^b, \boldsymbol{\lambda}^c, \boldsymbol{\lambda}^d\}$, the demands at each execution is permutated. This approach was adopted to minimize sequence-induced deviations within the production demands. As a result, we ended up with a substantial 240,000 task assignments in total.

- The results of these simulations are pictorially represented in two figures:

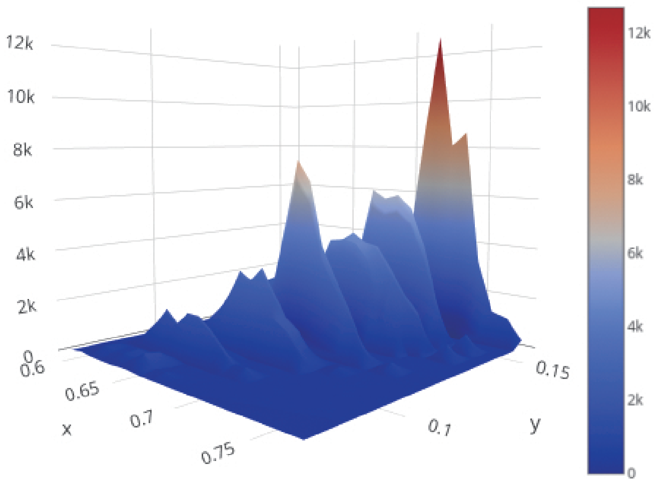
    1. Figure 4.1 showcases the cumulative distributions of solutions from two distinct viewpoints.

    2. Figure 4.1a presents the joint cumulative distribution for throughput-energy efficiency. Here, the x-axis represents throughput, the y-axis denotes energy efficiency, and the z-axis, characterized by a cold-warm color gradient, displays the cumulative number of identical solutions returned by the hypergraph search.

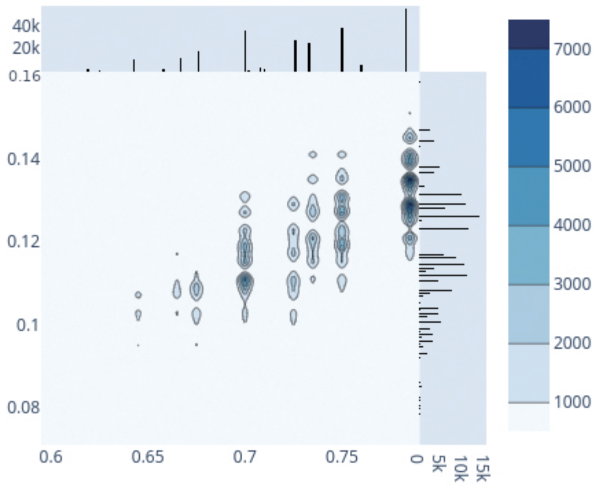
    3. Conversely, Figure 4.1b visualizes the same joint cumulative distribution, but the color depth symbolizes the cumulative number of identical solutions. The throughput is still on the x-axis and energy efficiency is on the y-axis. Furthermore, the

marginal cumulative distributions for both throughput and energy efficiency are displayed at the top and the right, respectively.

The intention behind this structured analysis is to provide a comprehensive perspective on the sub-optimality of the hypergraph approach, based on robust statistical methods.



(a) Joint cumulative distribution



(b) Marginal cumulative distribution

Figure 4.1 Statistical plot of Monte Carlo simulation.

In Figure 4.1a, the joint cumulative distribution displays a notable ridge formation along the z-axis, which can be characterized as the vector connecting adjacent maximum values. This ridge is oriented towards enhanced values of both throughput and energy efficiency within the Euclidean vector space delineated by the x, y, and z axes. Specifically, the ridge peaks at a throughput value of $\mathcal{T} = 0.78$ and an energy efficiency value of $\mathcal{E} = 0.16$.

On the other hand, Figure 4.1b, while visually representing the same observation through its color depth on the joint cumulative distributions, provides marginal cumulative distributions for throughput and energy efficiency. A closer examination of these marginal distributions reveals that the medians (0.733 for throughput and $1186.2e^{-4}$ for energy efficiency) exceed their respective means (0.727 for throughput and $1186.1e^{-4}$ for energy efficiency).

The observations outlined previously highlight the dual-objective optimization capability intrinsic to Algorithm 4.1. While it might appear trivial at first glance, the potential solution space for a system with 18 production robots is formidable, being expressed as $2^{\mathsf{N}\cdot\omega} = 2^{43}$. The addition of production demands amplifies this space further to $18!$. It's worth noting that even though the valid solution space is marginally smaller than this calculated expanse, not all valid solutions present equal likelihood during the MRTA solution process using Algorithm 4.1. Drawing from the law of large numbers, among the myriad task assignments Algorithm 4.1 yields for consistent input parameters, those solutions with elevated throughput and energy efficiency tend to manifest more frequently. This behavior underscores the algorithm's proclivity to optimize task assignments across both the throughput and energy efficiency dimensions, aligning well with the requisites of a contemporary smart factory.

THe prior sections concentrated on providing quantitative optimization objectives with a focus on product-averaged throughput and energy efficiency. However, when contemplating decision-making, several other metrics merit consideration. This includes, but isn't limited to, factors like production prioritization informed by comprehensive big data market analysis and forecasting, manufacturing costs derived from nuanced supply chain analytics, and forecasts pertaining to MRS maintenance grounded in data-centric insights, as elucidated by [85].

4.3 Toward Robust Real-time MRTA

Beyond the sub-optimality analysis for a specific MRS, smart factories, when grappling with dynamic production demands, encounter a nuanced interplay between achieving task assignment optimality during each system reconfiguration and ensuring aggregate task assignment optimality over time. This section posits that in the context of a smart factory, MRTA transcends mere optimization, emerging more aptly as a decision-making challenge. Guided by this perspective, the concept of "permissible MRS task assignments" is introduced. Opting for permissible solutions strikes a balance: it prioritizes computational efficiency and system resilience, even if it means a slight departure from absolute optimality. This approach ensures results that are in close proximity to the optimal solution. Moreover, the ensuing parts of this section delve into the robustness intrinsic to the hypergraph approach. This robustness is pivotal, enabling the system to nimbly adapt to fluctuating production demands and offering resilience against practical cyber anomalies, such as packet errors encountered in wireless communications.

4.3.1 Permissible Task Assignment Approximation

While the experiment hints that Monte Carlo simulations of Algorithm 4.1 might be adept at estimating local optimal task assignments, it's imperative to introduce scientific termination conditions to scale these simulations. Bootstrapping serves as a potent tool in this context, offering a statistical approximation of the empirical distribution derived from a limited set of solutions. Such an approach aids in achieving real-time computation. Consequently, this section harnesses bootstrapping to approximate the marginal cumulative distributions of both throughput and energy efficiency, ensuring these approximations closely adhere to the Gaussian distribution [59]. Additionally, using importance sampling, 99% confidence intervals are constructed for these two marginal cumulative distributions. These intervals delineate permissible task assignment boundaries and shed light on the delicate balance between Monte Carlo simulation iterations and the pursuit of task assignment optimality.

4.3.2 Parameter-Based Throughput and Energy Efficiency Estimation

Before deploying Algorithm 4.1, it's feasible to analytically determine the "best-case" and "worst-case" task assignments based on the provided inputs: $M, \mathbf{N}, \boldsymbol{\omega}, \varepsilon_h, \varepsilon_v, \varepsilon_d, \{\boldsymbol{\lambda}^l\}, L$. Specifically, Equation (4.4) identifies a "bottleneck" robot type, contingent upon a given production demand $\{\boldsymbol{\lambda}^l\}$. In this equation, the type-m robots exhibit the least ratio—this is computed by dividing the aggregate multiplexing capability of all type-m robots by the total number of type-m tasks present in $\{\boldsymbol{\lambda}^l\}$. Within the framework of the hypergraph search, these "bottleneck" type robots are prone to nearing their multiplexing capability limits. This saturation can either be attributed to their inherent limited multiplexing capabilities or to an excessive requirement of type-m tasks in $\{\boldsymbol{\lambda}^l\}$.

$$\hat{m} = \operatorname*{argmin}_m \left(\frac{N_1\omega_1}{\frac{1}{L}\sum_{l=1}^{L}\lambda_1^l}, \frac{N_2\omega_2}{\frac{1}{L}\sum_{l=1}^{L}\lambda_2^l}, \dots, \qquad \frac{N_m\omega_m}{\frac{1}{L}\sum_{l=1}^{L}\lambda_m^l}, \dots, \frac{N_M\omega_M}{\frac{1}{L}\sum_{l=1}^{L}\lambda_M^l} \right) \tag{4.4}$$

Consequently, the throughputs for the best-case, $\mathcal{U}_{\mathrm{T}}$, and the worst-case, $\mathcal{L}_{\mathrm{T}}$, are estimated as delineated in Equation (4.5). The identified “bottleneck” robot type, denoted as $\hat{m}$, signifies the best-case scenario. Meanwhile, the condition presented in Equation (4.2) illustrates the worst-case scenario, where only one unit of each demanded product is completed within a single time slot.

$$\mathcal{U}_{\mathrm{T}} = \frac{\frac{N_{\hat{m}}\omega_{\hat{m}}}{\frac{1}{L}\sum_{l=1}^{L}\lambda_{\hat{m}}^l}}{\frac{1}{L}\sum_{l=1}^{L}\sum_{m=1}^{M}\lceil\frac{\lambda_m^l}{\omega_m}\rceil} \tag{4.5a}$$

$$\mathcal{L}_{\mathrm{T}} = \frac{L_t}{\frac{1}{L_t}\sum_{l-1}^{L_t}\sum_{m=1}^{M} min(N_m, \lambda_m^l)} \tag{4.5b}$$

With respect to energy efficiency, it’s reasonable to assume that the energy consumption for transportation adheres to the relationship: $\varepsilon_h < \varepsilon_v < \varepsilon_d$. The best-case $\mathcal{U}_{\mathrm{E}}$ and worst-case $\mathcal{L}_{\mathrm{E}}$ for energy efficiency are estimated in Equation (4.6). In optimal scenarios, transport occurs solely along vertical and horizontal edges for both intra-type and inter-type movements. Conversely, in suboptimal scenarios, all inter-type transport actions transpire along diagonal edges.

$$\mathcal{U}_{\mathrm{E}} = \frac{1}{\frac{1}{L}\sum_{l=1}^{L}\{(M-1)\varepsilon_h + \varepsilon_v\sum_{m=1}^{M}\lceil\frac{\lambda_m^l}{\omega_m} - 1\rceil\}} \tag{4.6a}$$

$$\mathcal{L}_{\mathrm{E}} = \frac{1}{\frac{1}{L}\sum_{l=1}^{L}[(M-1)\varepsilon_d + \varepsilon_v\sum_{m=1}^{M}(\lambda_m^l - 1)]} \tag{4.6b}$$

It's worth noting that both Equations (4.5) and (4.6) should be treated as approximations since their derivations are predicated on product averages and do not differentiate among the L distinct products.

4.3.3 Permissible Task Assignment Boundaries

In the quest to derive scientifically informed permissible boundaries, this section begins by applying bootstrapping, using an initial subset of algorithm executions, to obtain a preliminary set of task assignments, θ. The empirical marginal distributions for throughput and energy efficiency are denoted in terms of their means, μ_t and μ_e, as well as their variances, σ_t^2 and σ_e^2, respectively. To counteract potential bias stemming from the limited initial executions, the analytical estimations $\mathcal{U}_\mathrm{T}$, $\mathcal{L}_\mathrm{T}$, $\mathcal{U}_\mathrm{E}$, and $\mathcal{L}_\mathrm{E}$ are integrated into the bootstrapping pool. From this, the values of μ_t, μ_e, σ_t^2, and σ_e^2 are derived, leveraging the sample mean and variance across B bootstrap iterations.

To derive distributions of particular interest, importance sampling is performed on both the marginal cumulative throughput and energy efficiency distributions. The means for these samplings are set to the best-case estimations for throughput, $\mathcal{U}_\mathrm{T}$, and energy efficiency, $\mathcal{U}_\mathrm{E}$, respectively. Let w_t and w_e represent the likelihood or weight of these two importance samplings. Given a permissible task assignment characterized by throughput $\mathcal{T}$ and energy efficiency $\mathcal{E}$, boundaries for this assignment can be defined as per (4.7). Here, $z_{1-0.99/2}$ represents the critical value for the standard Gaussian distribution.

$$\mathcal{T} \geq \mathcal{T}_\mathrm{permissible} = \mu_t - z_{1-0.99/2}\frac{w_t\sigma_t}{\sqrt{\theta+2}} \tag{4.7a}$$

$$\mathcal{E} \geq \mathcal{E}_\mathrm{permissible} = \mu_e - z_{1-0.99/2}\frac{w_e\sigma_e}{\sqrt{\theta+2}} \tag{4.7b}$$

Consequently, the trade-off mechanism can be distilled into three pivotal parameters: the initial execution time θ, the desired quantity of permissible task assignments N, and

Table 4.2 Parameters for 10000 random cases.

Parameter	Values or Vectors
M	$\mathcal{U}(3, 10)$
N_m	$(\mathcal{U}(3, 10)), \quad m = 1, 2, \dots, M$
ω_m	$(\mathcal{U}(1, 5)), \quad m = 1, 2, \dots, M$
L	$\mathcal{U}(1, 5)$
λ_m^l	$\mathcal{U}(0, 5), \quad l = 1, 2, \dots, L$

the upper limit for Monte Carlo simulation time, MC_{max}. Augmenting the values of these parameters prompts further runs of Algorithm 4.1, steering the solution closer to the sought-after optimality.

An experiment was conducted involving 10,000 randomly generated MRSs using parameters defined in Table 4.2. The tuple was defined in the following, and a uniform distribution, $\mathcal{U}(a, b)$, was used. Remarkably, out of these 10,000 cases, 6,476 achieved 10 permissible solutions and concluded within just 10 executions. This highlights the superiority of permissible task assignments compared to relying solely on a large fixed number of Monte Carlo simulations. Conversely, 782 cases culminated at 100 iterations, with an average parameter L value of 1.99, which is notably lower than the overall mean of 3 observed across all 10,000 cases. These findings underscore that the efficacy of the trade-off mechanism is influenced not just by production demand dynamics, but also by the specific parameters of the MRS, laying the groundwork for intriguing future research avenues.

$$(\theta, B, w_t, w_e, \mathsf{N}, \mathsf{MC}_{max}) = (10, 100, 0.1, 5.0, 10, 100)$$

4.3.4 Raise of Non-stationarity and Resilience Challenges

Smart factories, equipped with real-time MRTA-coordinated MRS, offer advantages in flexibility, productivity, and energy efficiency. However, they also raise concerns about robustness in the face of dynamic production demands and resilience against issues related to cyber reliability and security [21, 114].

The experiments reveal that the permissible task assignment method strikes a good balance between computation time and task assignment optimality. Furthermore, the robustness of Algorithm 4.1 has been validated under the challenging scenario of $10,000$ random MRSs.

This section delves deeper by evaluating the hypergraph approach against 20 randomly generated production demands, simulating 20 reconfigurations determined by the edge computing parameters $(M, \mathbf{N}, \boldsymbol{\omega}, \varepsilon_h, \varepsilon_v, \varepsilon_d)$ as listed in Table 4.1, and $\{\boldsymbol{\lambda}^l\}$, L from Table 4.2.

Figure 4.2 displays the reconfigurations for $t = 1, 2, \ldots, 20$. At each reconfiguration time instant t, real-time MRTA is executed, yielding 1,000 solutions as its termination conditions. This helps in observing the deterministic nature of the approach. The upper and lower sections of Figure 4.2 depict the throughput and energy efficiency over the course of t respectively. The blue shades represent the marginal cumulative distribution of the 1,000 solutions, while the dashed line indicates the lower bounds of the permissible solutions.

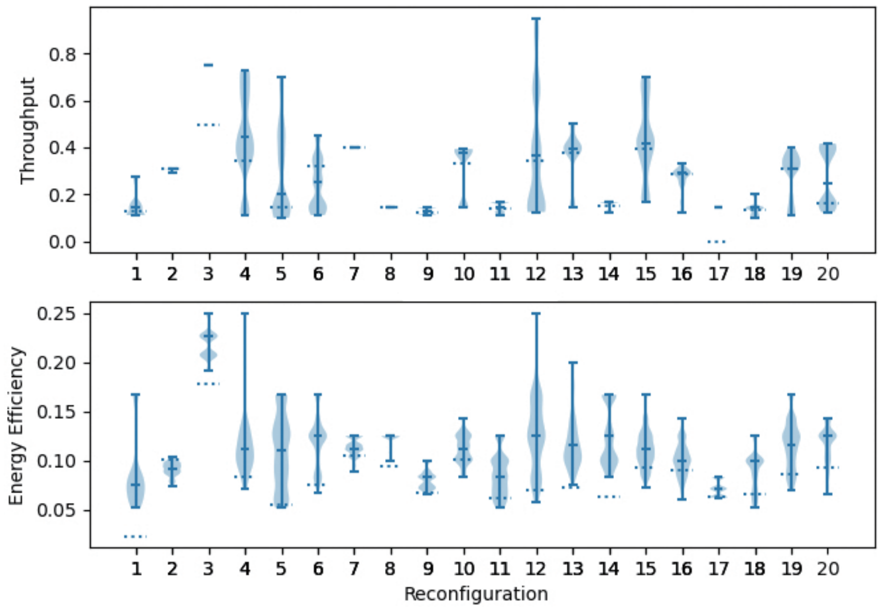


Figure 4.2 Time dynamic experiment uses randomly generated production demands to request real-time MRTA and the MRS is reconfigured following output task assignments to simulate the dynamics of smart factories.

The results underscore the prowess of the hypergraph methodology when confronted with randomly generated production demands. It consistently delivers effective, near-optimal solutions. The marginal cumulative distributions vividly illustrate this, showcasing encouraging outcomes even under challenging conditions where feasible solutions are limited. Such findings highlight the robustness of the proposed approach, which can assure effective task assignments without the need for explicit prediction of production demands.

Moreover, the influence of MRS reconfigurations and their intrinsic non-stationarity becomes evident. As depicted in Figure 4.2, the statistical characteristics of each reconfiguration shift in response to the changing dynamics of production demands. Considering that task assignments dictate the operation of the MRS, the oscillating nature of these assignments—and by extension, task execution—presents a formidable resilience challenge for smart factories.

Resilience in smart factories extends beyond just adaptability; it also necessitates short latency and swift responsiveness [62] especially when faced with dynamic production demands compounded by potential cyber reliability issues and potential cyber-attacks.

Among the pivotal technologies introduced in this study, the MRS—which encompasses production and transportation robots, wireless networks, wireless communications, and edge computing—stands out. While these technologies can significantly enhance manufacturing processes, they also introduce new risks.

This landscape opens doors for research opportunities, particularly in the realm of cyber-physical systems. Exploring the nexus between the cyber domain—comprising AI and wireless networking—and the physical domain, characterized by production flow-based networked robots, becomes crucial [43].

Ensuring resilience in the operations of an edge computing-coordinated MRS demands a reliable network design [43]. It also requires the adoption of ultra-reliable, low-latency communications [21, 22, 62] and the implementation of distributed computing on robots [19].

Using wireless communications in smart factories as a lens to examine cyber reliability issues, an experiment based on Table 4.2 is conducted. Five identical MRSs—each with the same production demands and the same reconfigurations for $t = 1, 2, \ldots, 20$ are examined. However, each MRS had unique permissible task assignments, as depicted in Figure 4.3.

Instead of the analytical throughput, in this context, throughput is assessed at each time slot. This is based on the straightforward MRS resilient consensus principle [135]: a production flow will only execute the assigned task if all robots are free from random wireless packet errors. These wireless error rates span from 0.01 to 0.1 [34].

Given that the MRSs are identical and face the same production demands, their analytical throughput remains consistent, represented as the scenario "Error rate = 0" in Figure 4.3. Yet, when accounting for the variations in MRS task assignments and random wireless errors, the simulated throughput exhibits divergent drops across the 5 identical MRSs for all three error rate scenarios and all 20 reconfigurations. This starkly illustrates the influence of cyber domain errors on the physical domain within an MRS-driven smart factory.

From the above, it's evident that while MRS task assignments optimized for throughput and energy efficiency at all times can be appealing, they may not align perfectly with the overarching goals of smart factories. Such assignments might not always translate to optimized productivity and resource efficiency in the long run. Moreover, they could introduce additional computation time and elevate the risks associated with robustness and resilience. This further underscores the sub-optimality of the hypergraph search approach.

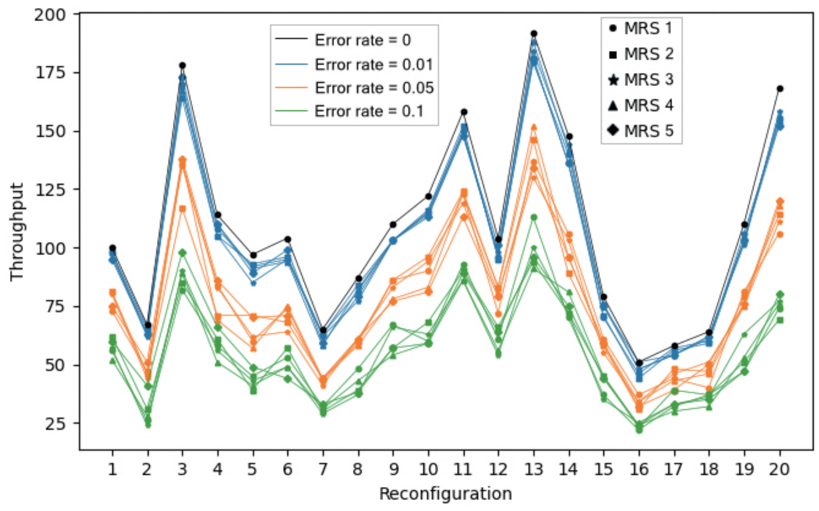


Figure 4.3 Throughput of the same MRS under 5 wireless packet error rates.

Chapter 5: Predictive Path Coordination of Collaborative Transportation MRS for Productive Transportation

AMRs in the transportation-MRS execute transportation tasks in an end-to-end sequence, leading to asynchronous task execution; a new task is assigned upon completion of the prior one. Dynamic production flows necessitate reconfigurations in two MRSs, specifically, changes in production and transportation tasks. Nonetheless, between these reconfigurations, the transportation-MRS operates with a static set of tasks that reflect production tasks and flows. This dynamic nature means transportation task assignments for AMRs are inherently unpredictable, given they might be reconfigured a different set of tasks to serve altered production flows, reconfigured to support an ongoing flow, or assigned a completely new task due to reconfiguration.

Let M^t denote the total number of transportation tasks adjustable at reconfiguration intervals represented by $t = t_1, t_2, ...$. At each reconfiguration, edge computing formulates a new set of M^t tasks, TS^t, consisting of pairs of a pickup vertex $v^m_{f_p,x_p,y_p}$ and a delivery vertex $v^m_{f_d,x_d,y_d}$, as defined in (5.1a). The task generation process is mathematically described by (5.1b), where h_{DU} denotes a discrete uniform distribution used to select two vertices from G_{floor} for pickup and delivery.

$$\begin{aligned} TS^t = \{&(v^1_{f_p,x_p,y_p}, v^2_{f_d,x_d,y_d}), \ldots, (v^m_{f_p,x_p,y_p}, v^m_{f_d,x_d,y_d}), \\ &\ldots, (v^{M^t}_{f_p,x_p,y_p}, v^{M^t}_{f_d,x_d,y_d})\} \end{aligned} \tag{5.1a}$$

$$(v^m_{f_p,x_p,y_p}, v^m_{f_d,x_d,y_d}) = h_{\mathrm{DU}}(2; 0, N_f \cdot N_x \cdot N_y) \tag{5.1b}$$

$$m = 1, 2, \ldots, M^t \tag{5.1c}$$

$$t = t_1, t_2, \ldots \tag{5.1d}$$

A transportation task is denoted by task^m for $m = 1, 2, \dots, M^t$. When this task is assigned to AMR-k, it is defined by the pickup vertex $v^m_{f_p,x_p,y_p}$, the delivery vertex $v^m_{f_d,x_d,y_d}$, and the associated pickup and delivery due times, $\tau^m_{\text{due,pickup}}$ and $\tau^m_{\text{due,deliver}}$, respectively, as described in (5.2). Assuming AMR-k is at location v_{f_k,x_k,y_k} at the time of assignment, it first navigates to $v^m_{f_p,x_p,y_p}$ for pickup and then proceeds to $v^m_{f_d,x_d,y_d}$ for delivery. The A^* algorithm, denoted by h_{A^*} [103], calculates the shortest feasible delay for both routes. However, since AMRs might not always choose the shortest path, a margin parameter β is introduced to adjust the due times accordingly.

$$\mathsf{task}^m = \left(v^m_{f_p,x_p,y_p}, v^m_{f_d,x_d,y_d}, \tau^m_{\text{due,pickup}}, \tau^m_{\text{due,deliver}}\right) \tag{5.2a}$$

$$\tau^m_{\text{due,pickup}} = (1+\beta) \cdot h_{A^*}\left(v_{f_k,x_k,y_k}, v^m_{f_p,x_p,y_p}\right) \tag{5.2b}$$

$$\tau^m_{\text{due,deliver}} = (1+\beta) \cdot h_{A^*}\left(v_{f_p,x_p,y_p}, v^m_{f_d,x_d,y_d}\right) \tag{5.2c}$$

In summary, the set of transportation tasks is refreshed during the reconfiguration time slot, which occurs every several hundred to thousand time slots [120], as indicated by (5.1).

A single navigation step taken by AMR-k at time slot t is denoted as a^t_k (termed an action). This action transitions AMR-k from vertex $v_{f,x,y}$ to an adjacent vertex $v_{f',x',y'}$ in G_{floor}, as concisely represented by (5.3a). Equations (4b) to (4h) explicitly define the empirical actions "west", "south", "east", "north", "stay", "up-floor", and down-floor and their corresponding numerical representations. Thus, the discrete action space for AMRs is given by $\mathcal{A} = \{0, 1, 2, 3, 4, 5, 6\}$.

Integrating the models from the previous two subsections, when AMR-k at location v_{f_0,x_0,y_0} is assigned a task, task^m, as defined by (5.2) at time slot t, it first navigates to v_{f_p,x_p,y_p} in $\tau^m_{k,\text{pickup}}$ time slots for pickup. Subsequently, it moves to v_{f_d,x_d,y_d} in $\tau^m_{k,\text{deliver}}$ time slots for delivery, marking the completion of the task execution.

This assignment results in two walks on G_{floor}: $\boldsymbol{\rho}^m_{k,\text{pickup}}$ and $\boldsymbol{\rho}^m_{k,\text{deliver}}$, as defined by (5.4a) and (5.4b), respectively. The delay and energy consumption associated with AMR-k completing task^m are described by (5.4c) and (5.4d), respectively.

$$v_{f,x,y} \xrightarrow{a_k^t} v_{f',x',y'} \tag{5.3a}$$

$$a_k^t = 0 := f' = f, x' = x, y' = y - 1 \tag{5.3b}$$

$$a_k^t = 1 := f' = f, x' = x + 1, y' = y \tag{5.3c}$$

$$a_k^t = 2 := f' = f, x' = x, y' = y + 1 \tag{5.3d}$$

$$a_k^t = 3 := f' = f, x' = x - 1, y' = y \tag{5.3e}$$

$$a_k^t = 4 := f' = f, x' = x, y' = y \tag{5.3f}$$

$$a_k^t = 5 := f' = f + 1, x' = x - 1, y' = y \tag{5.3g}$$

$$a_k^t = 6 := f' = f - 1, x' = x - 1, y' = y \tag{5.3h}$$

As established in Section 3.1.3, each navigation step occupies one time slot. Therefore, the delay is equivalent to the number of vertices in the walks $\boldsymbol{\rho}^m_{k,\text{pickup}}$ and $\boldsymbol{\rho}^m_{k,\text{deliver}}$ minus one. This count also corresponds to the number of actions forming each walk.

$$\begin{aligned}\boldsymbol{\rho}^m_{k,\text{pickup}} = \big(v_{f_0,x_0,y_0} \xrightarrow{a_k^t} v_{f_1,x_1,y_1} \xrightarrow{a_k^{t+1}} v_{f_2,x_2,y_2} \dots \\ \dots \xrightarrow{a_k^{t+\tau_{\text{pickup}}}} v^m_{f_p,x_p,y_p}\big)\end{aligned} \tag{5.4a}$$

$$\begin{aligned}\boldsymbol{\rho}^m_{k,\text{deliver}} = \big(v^m_{f_p,x_p,y_p} \xrightarrow{a_k^{t+\tau_{\text{pickup}}+1}} v_{f_3,x_3,y_3} \dots \\ \dots \xrightarrow{a_k^{t+\tau_{\text{pickup}}+\tau_{\text{deliver}}}} v^m_{f_d,x_d,y_d}\big)\end{aligned} \tag{5.4b}$$

$$\tau^m_k = \tau^m_{k,\text{pickup}} + \tau^m_{k\text{deliver}} \tag{5.4c}$$

$$\mathcal{E}^m_k = h_{\text{weight}}(\boldsymbol{\rho}^m_{k,\text{pickup}}) + h_{\text{weight}}(\boldsymbol{\rho}^m_{k,\text{deliver}}) \tag{5.4d}$$

A potential collision during task execution can arise when the set of AMRs, $K^t_{f,x,y}$, is located at $v_{f,x,y}$ at time t. Given all planned actions a^t_k for k in $K^t_{f,x,y}$, the action set is invalid, as indicated by (5.5), due to the presence of duplicate elements. Fortunately, with time synchronization, potential collisions can be mitigated using token-passing. This resolution might prompt AMRs to opt for the "stay" action, which could prolong the task execution delay.

$$\nexists\{a^t_k|\forall a^t_k, k \in K^t_{f,x,y}\} \tag{5.5}$$

Delay (also referred to as completion time or makespan) and energy consumption stand as the primary performance indicators for smart factory transportation [39, 28]. Thus, these indicators are used to assess the performance of the transportation-MRS, as described by (5.6a). This evaluation takes into account the proposed multi-floor model, task model, and the formulated discrete navigation problem. Given that both delay and energy consumption are more favorable when minimized, a negative Cobb-Douglas utility function [56] is used. In this function, α signifies the preference balance between delay and energy. The function given by (5.6a) is termed the global utility, representing the average delay and energy consumption across all completed tasks and all AMRs. Further, (5.6c) indicates that both delay and energy originate from the task execution paths, $\boldsymbol{\rho}^m_{k,\text{pickup}}$ and $\boldsymbol{\rho}^m_{k,\text{deliver}}$. These paths must be consistent with task^m.

$$\text{minimize} \quad \frac{\sum_{k=1}^{K}\sum_{t=1}^{T}\sum_{m=1}^{M^t}(\tau^m_k)^{\alpha}(\mathcal{E}^m_k)^{1-\alpha}}{K\sum_{t=1}^{T}M^t} \tag{5.6a}$$

$$s.t. \tag{5.6b}$$

$$\exists \quad \tau^m_k, \mathcal{E}^m_k \leftarrow (\mathsf{task}^m, \boldsymbol{\rho}^m_{k,\text{pickup}}, \boldsymbol{\rho}^m_{k,\text{deliver}}) \tag{5.6c}$$

$$m = 1, 2, \ldots, M^t \tag{5.6d}$$

The primary objective of coordination within the transportation-MRS is the optimization of both delay and energy consumption. This is assessed using the global utility as defined by (5.6a). It's important to note that the delay and energy consumption during task executions result from the discrete navigation actions of all AMRs within the transportation-MRS. This scenario presents a multiagent optimization problem.

5.1 Collborative Multi-Intelligent Robot System with Domain Model for Path Coordination

This section practices the cyber-physical AI agent system approach introduced in Section 3.3 and integrates the multi-floor model from Section 3.1.3 as its domain model. The overall architecture is illustrated in Figure 5.1.

Within this framework, agents perceive their environment primarily through transportation tasks, with the multi-floor domain model serving as an assisting tool. Notably, any reconfigurations leading to non-stationary transportation tasks are known by the agents. These AMRs work in a collaborative and predictive manner, executing transportation tasks along paths that are planned to avoid collisions. In instances where potential collisions arise, they are addressed using a token-passing-based collision resolution.

Collectively, this methodology targets the optimization of the collective objective, as delineated by Equation (5.6).

The global critic, represented as V_g, is a shared entity amongst all AMRs and is hence maintained via edge computing. Its primary role is to forecast the global reward, which is a reflection of a pickup or delivery path's contribution to timely task completion and the global utility. This prediction is made based on the information from the assigned task, task^m, and its associated path, $\boldsymbol{\rho}_k^m$.

Deviant from the prevailing centralized critic methodologies documented in scholarly literature, the global reward mechanism in this design is intrinsically agent-centric. Given the decentralized nature of the framework, accurately predicting the exact time slot earmarked

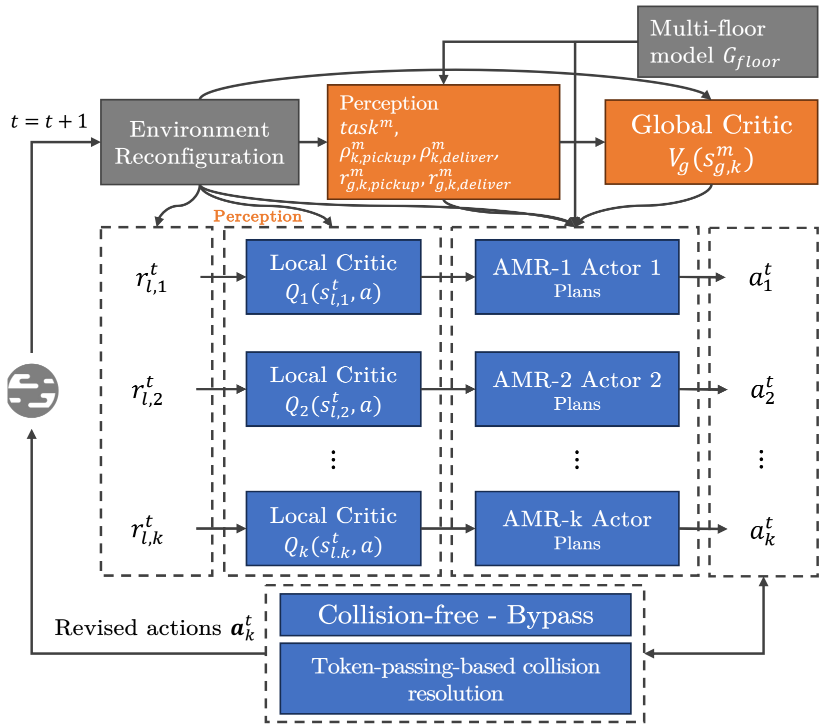

Multi-floor model G_{floor}
$t = t + 1$
Environment Reconfiguration
Perception $task^m$, $\rho^m_{k,pickup}, \rho^m_{k,deliver}$, $r^m_{g,k,pickup}, r^m_{g,k,deliver}$
Global Critic $V_g(s^m_{g,k})$
Perception
$r^t_{l,1}$
$r^t_{l,2}$
$r^t_{l,k}$
Local Critic $Q_1(s^t_{l,1}, a)$
Local Critic $Q_2(s^t_{l,2}, a)$
Local Critic $Q_k(s^t_{l,k}, a)$
AMR-1 Actor 1 Plans
AMR-2 Actor 2 Plans
AMR-k Actor Plans
a^t_1
a^t_2
a^t_k
Revised actions $\boldsymbol{a}^t_k$
Collision-free - Bypass
Token-passing-based collision resolution

Figure 5.1 Architecture of multi-agent "Actor-Mixed Critics" approach under MARL framework.

for task culmination across all AMRs becomes an intricate challenge. To address this, the global reward is disseminated in each distinct time slot. As elucidated in (5.7), when AMR-k achieves either a pickup or delivery within the prescribed timeframe, it garners an amplified reward of 10. Conversely, any task that overshoots its due time is penalized, receiving a muted reward of 5. Throughout the intervening period, until the pickup or delivery milestone is realized, the ledger for AMR-k consistently reflects a global reward tally of 0.

$$r^t_{g,k} = \begin{cases} 10 & \text{picked up or delivered and } \tau^m_k \leq \tau^m_{\text{due}} \\ 5 & \text{picked up or delivered and } \tau^m_k > \tau^m_{\text{due}} \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

Each AMR operates and refines its local critic, denoted Q_k, in a distributed fashion. The primary purpose of the local critic Q_k is to guide AMR-k in evading potential collisions. This is achieved by predicting the expected penalty, which is referred to as the local reward, for every conceivable action in its defined action space, $\mathcal{A}$. These predictions stem from historical encounters with potential collision scenarios.

The local state $s_{l,k}^t$ is adeptly designed as an 8-element vector, which resonates with AMR-k's spatial position at the time instant t, denoted as v_{f_k,x_k,y_k}^t. By having such a formulation, the relationship between the local state and the ensuing actions is better generalized, ensuring a more efficient navigation system.

Local rewards, as articulated by (5.8), are derived considering the situations where K_{f_k,x_k,y_k}^t AMRs are concurrently positioned at v_{f_k,x_k,y_k} during time t. An AMR that takes an action that directly conflicts with the maneuver of another AMR, or situations where more than three AMRs are congregated at a single vertex, invokes a penalty via the local rewards mechanism. It is imperative to note that when three AMRs coexist at a singular position, the susceptibility to collisions escalates dramatically.

$$r_{l,k}^t = \begin{cases} -1 & \text{action cause potential collision as (5.5),} \\ & \text{or } K_{f_k,x_k,y_k}^t > 3 \\ 0 & \text{otherwise} \end{cases} \tag{5.8}$$

5.1.1 Predictive Collision Avoidance

Building upon the planning methodology elucidated in Section 3.2.2.2, the actor module strategically chooses κ_k plans out of the available κ_N plans. This selection process is driven by the most promising advantage values, as delineated by A_k in (5.9). This particular advantage function amalgamates insights from both the global and local critics' assessments for AMR-k.

Drawing inspiration from the generalized advantage function presented by [95], which presents a comparative evaluation of actions within an action space, (5.9) ventures a step further. It encapsulates the overarching global utility metric (5.6a), emphasizing the relative

Algorithm 5.1: Actor module

```
def Evalaute_Path($\boldsymbol{\rho}_k^m$):
    Result: plans
    values_global ← $V_g(s_{g,k}^m \leftarrow$ PathEmbd($\{\boldsymbol{\rho}_k^m, \dots\}$))
    values_local_mean ← Mean ($Q_k(\{s_l^k, a_k, \dots\} \leftarrow \{\boldsymbol{\rho}_k^m, \dots\})$)
    $A_k(\{\boldsymbol{\rho}_k^m, \dots\}) \leftarrow \alpha_g \cdot$ values_global + values_local_mean
    plans ← argsort ($\kappa_k, A_k(\{\boldsymbol{\rho}_k^m, \dots\})$)

def Calculate_Value($a_k^t, a_k^{t-1}, s_{l,k}^t, Q_k$):
    Result: $\mathbf{V_k^t}$
    values_local ← $Q_k(s_{l,k}^t, a), a \in \mathcal{A}$
    values_local ← argsort (values_local)
    for $a$ in $\mathcal{A}$:
        if $a$ == Reverse ($a_k^{t-1}$):
            values_local($a$) = 0
        elif $a == a_k^t$:
            values_local($a$) = 6
        elif $a$ meet cross_floor_condition:
            values_local($a$) = 0
    $\mathbf{V_k^t}$ ← Map (values_local, [0, 1, 2, 3, 4, 5, 6])

def Act(plan, $v_{f_k,x_k,y_k}, G_{floor}$, task$^m$, $V_g, Q_k, \kappa_N, \kappa_k$):
    Result: $a_k^t$
    if deadlock detected:
        switch_flag ←Switch_plan ($v_{f_k,x_k,y_k}$)
        if ¬switch_flag:
            if Random > 0.5:
                return $a_k^t = 4$
    if $v_{f_k,x_k,y_k} \notin$ plan:
        switch_flag ←Switch_plan ($v_{f_k,x_k,y_k}$)
        if ¬switch_flag:
            Plan ($v_{f_k,x_k,y_k}, G_{floor}$, task$^m$, $V_g, Q_k, \kappa_N, \kappa_k$)
    plan ← Get_plan ($v_{f_k,x_k,y_k}$)
    $a_k^t$ ← plan
```

impact of the plans on pivotal aspects such as prompt task execution, delay minimization, energy conservation, and effective collision circumvention.

Incorporating concepts from TD(λ) [100], for any designated path plan $\boldsymbol{\rho}_k^m$, the expression in (5.9) integrates the global critic V_g, offering a projection of the cumulative expected returns arising from $\boldsymbol{\rho}_k^m$. Concurrently, it melds insights from the local critic Q_k, estimating the prospective values of all individual actions forming the backbone of $\boldsymbol{\rho}_k^m$. Yet, given the varying action count within each plan $\boldsymbol{\rho}_k^m$, there's an innate bias. To counteract this, the predictions from Q_k are uniformly averaged across all constituent actions, indexed as i.

In this multifaceted planning landscape, the determination of an ideal coefficient, represented as α_g, becomes pivotal. This coefficient aims to balance the evaluations from the global critic with those of the local critic, and its ideal value should be empirically derived, weighing the nuances of both global and local reward formulations.

$$A_k(\boldsymbol{\rho}_k^m) = \alpha_g V_g(s_{g,k}^m \leftarrow \texttt{PathEmbd}(\boldsymbol{\rho}_k^m)) + \frac{1}{i} \sum_{s_l^k, a_k \in \boldsymbol{\rho}_k^m} Q_k(s_l^k, a_k \leftarrow \boldsymbol{\rho}_k^m) \tag{5.9}$$

Upon evaluating the advantage values, the actor finalizes its decision by adopting the plan with the paramount advantage value as the chief policy for action. The remainder of the plans, that is $\kappa_k - 1$, are retained as contingency or backup strategies. This decision-making process is compactly detailed within the `Evaluate_Path` function, specifically from lines 1 to 4, as presented in Algorithm 5.1.

5.1.2 Collision Resolution with Token-passing

Under the proposed collaborative cyber-physical AI agent system approach, a token-passing-based collision resolution is employed to address situations where potential collisions have not been preemptively avoided. This method adeptly manages the equilibrium between collaboration and the risk of collisions through communication among intelligent agents.

In the transportation-MRS introduced in Section 3.1.3, even though task execution progresses asynchronously, discrete navigation steps among AMRs are synchronous. When potential collisions occur, the involved AMRs collaboratively resolve the collision in a decentralized fashion to maintain collision-free task executions.

The token-passing-based collision resolution draws inspiration from the Dutch auction method [92]. In this configuration, each AMR has a local numerical value, denoted as token_k^t, serving as a "currency" to streamline the action-claiming process. Every AMR commences with a primary allocation of tokens, symbolized by token_k^0.

Referring to Figure 5.2, imagine a situation where a potential collision is forecasted among a group of AMRs, such as AMR-$1, 2, 3, 4$, at a specific time t. Initially, each AMR employs its actor module to determine a value vector V_k^t relative to the action space. The derivation of this vector is elaborated in Algorithm 5.1 under the function `Calculate_Value`, particularly from lines 5 to 14. It ensures actions that circumvent potential collisions have elevated values, with lines 14 to 19 integrating domain-specific knowledge. A planned action, highlighted in lines 16 and 17, is assigned a value of 6. Conversely, unplanned cross-floor actions, discerned by `cross_floor_condition`, and the action reverting the last one, are assigned a value of 0. The `Map` function ensures that all V_k^t integer elements are within the $[0, 6]$ interval, in agreement with action space $\mathcal{A}$.

Subsequently, AMRs disseminate their value vectors and token counts, leading to a unified value vector, $\mathbf{V}_{k,a}^t$, and token vector $\mathbf{token}^t$ for the decentralized execution of Algorithm 5.2. Lines 2 and 4 accentuate that AMRs possessing more tokens receive preference, enabling them to commit to their prime actions ahead of others. Conversely, token-deficient AMRs are left with the residual choices in `Action_set`. In cases of token scarcity, as shown in lines 7 to 13, certain AMRs might not secure any action. Line 5 emphasizes that AMRs only select actions with values greater than 0, viewing 0-valued actions as unfavorable. Unsuccessful AMRs default to the "stay" action, which is free of token charges, as outlined in lines 14 to 16. Conclusively, the `Token_exchange` process guarantees the tokens expended by AMRs are evenly redistributed among others, ensuring the system's token count remains invariant, reflecting a zero-sum game. Post-resolution, all AMRs exchange the obtained outcomes, $\mathbf{X}^t$, and their refreshed token numbers, represented by $\mathbf{token}^t$.

The resolution of potential collisions is decentralized among the colliding AMRs. This decentralized approach is enabled through a proximity-based ad hoc wireless network. Such a network offers significant advantages, including enhanced scalability, optimized communi-

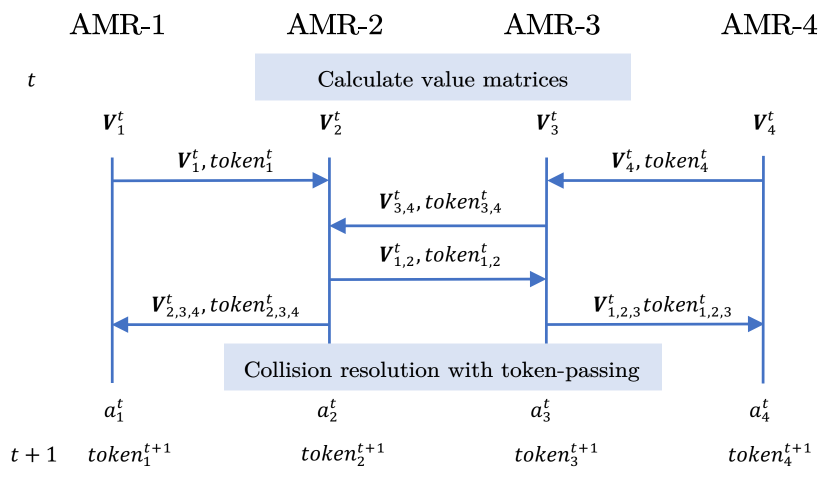

AMR-1
AMR-2
AMR-3
AMR-4
t
Calculate value matrices
$\boldsymbol{V}_1^t$
$\boldsymbol{V}_2^t$
$\boldsymbol{V}_3^t$
$\boldsymbol{V}_4^t$
$\boldsymbol{V}_1^t, token_1^t$
$\boldsymbol{V}_4^t, token_4^t$
$\boldsymbol{V}_{3,4}^t, token_{3,4}^t$
$\boldsymbol{V}_{1,2}^t, token_{1,2}^t$
$\boldsymbol{V}_{2,3,4}^t, token_{2,3,4}^t$
$\boldsymbol{V}_{1,2,3}^t token_{1,2,3}^t$
Collision resolution with token-passing
a_1^t
a_2^t
a_3^t
a_4^t
$t+1$
$token_1^{t+1}$
$token_2^{t+1}$
$token_3^{t+1}$
$token_4^{t+1}$

Figure 5.2 An example of distributed collision resolution among 3 AMRs.

cation efficiency, and reduced energy consumption. In scenarios where multiple AMRs show an inclination towards the same actions and possess identical token counts, a synchronized random seed is utilized. The synchronization ensures that the resulting decisions, denoted as $\mathbf{X}^t$, remain uniform across all participating AMRs.

5.2 Computational Experiments

To showcase the efficacy, near-optimality, and scalability of the proposed methodology, computational experiments are performed using varying numbers of AMRs and multi-floor models. Specifically, we consider:

$$K \in \{13, 17, 19, 23, 29, 31\}$$

and

$$(N_f, N_x, N_y) \in \{(2,4,5), (2,5,5), (2,5,6), (3,3,4), (3,4,4), (3,4,5)\}.$$

To emphasize potential collisions, the scenario with 31 AMRs on a multi-floor model is defined by the dimensions $(3,3,4)$, resulting in 36 vertices. This experiment runs for a total of $10,000$ timesteps to underscore the method's effectiveness and near-optimality. This

Algorithm 5.2: Collision resolution with token-passing

```
   Data: $\mathcal{A}, \mathbf{a}_n^t = (a_k^t, \dots), \mathbf{V}_{k,a}^t, \mathbf{token}^t = (\mathsf{token}_k^t, \dots)$
 1 $\mathbf{X}^t$, prices $\leftarrow \emptyset$
 2 AMR_set $\leftarrow$ argsort ($\mathbf{token}$)
 3 Action_set $\leftarrow \mathcal{A}$
 4 for $k$ in AMR_set:
 5     desired_actions_k $\leftarrow$ argsort ($\mathbf{V}_{k,a}^t$)
 6     for $a_k$ in desired_actions_k:
 7         if $a_k$ in Action_set:
 8             $\mathbf{X}^t \leftarrow k, a_k$
 9             price $\leftarrow \mathbf{V}_{k,a}^t$
10             if price $\geq \mathsf{token}_k$:
11                 Action_set.remove($a_k$)
12                 prices $\leftarrow$ price
13             break
14     if $k$ not in $\mathbf{X}^t$:
15         $\mathbf{X}^t \leftarrow k, a_k = 4$
16         price $\leftarrow 0$
17 $\mathbf{token}^{t+1} \leftarrow$ Token_exchange ($\mathbf{X}^{n,t}$, prices, $\mathbf{token}$)
```

configuration, having the highest AMR count and the fewest vertices across all scenarios, inherently poses the maximum risk of potential collisions given the path constraints.

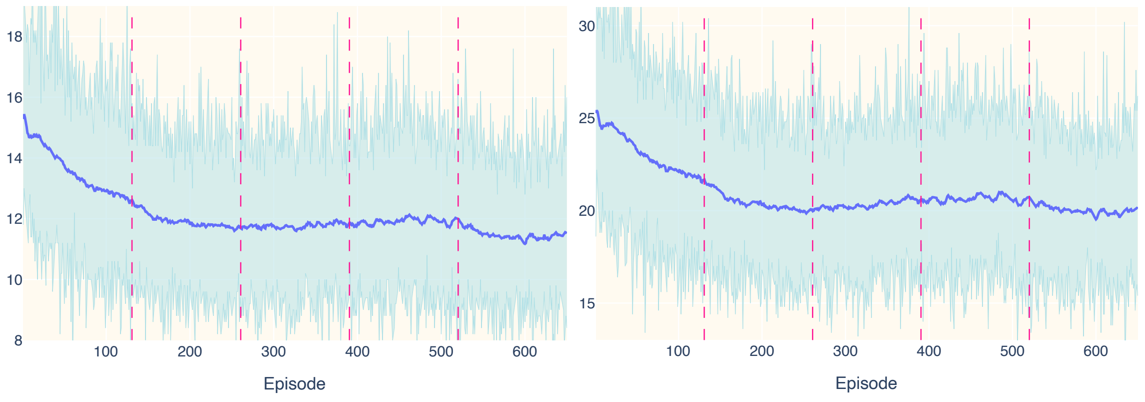
Throughout the $10,000$ timesteps, task reconfigurations occur at intervals: 2000, 4000, 6000, and 8000 timesteps. This simulates dynamic reconfigurations typically observed in smart manufacturing environments, a situation unforeseen by all AMRs. To ensure result comparability across different task sets, a consistent task set is maintained where tasks possess an optimal delivery path delay of 6. This consistency ensures that every task set presents a similar level of challenge for the AMRs.

All experiments are executed five times, each with unique seed sets for the environment and individual AMRs. Leveraging the parameters enumerated in Table 5.1, the results illustrate average curves (considering both AMR numbers and five iterations) with accompanying shaded regions indicating the range observed over the five repetitions.

The experimental outcomes for a scenario with 31 AMRs on a multi-floor model of dimensions $(3, 3, 4)$ are presented in Figures 5.3a through 5.6b.

Table 5.1 Parameters of predictive transportation path coordination experiments.

Parameter	*Value*	*Parameter*	*Value*
N_f, N_x, N_y, K	3, 3, 4, 31, 7	p_{walks}, q_{walks}	1, 1
$\varepsilon_f, \varepsilon_x, \varepsilon_y$	3, 1, 2	κ_k	5
β	0.2	ϵ	1
α	0.7	token_k^0	10
κ_N	240	Training period	10 time slots
lr_g, lr_l^k	0.001, 0.0005	V_g hidden size	[6, 6]
γ_l, γ_{decay}	0.98, 0.90	Q_k hidden size	[8]
α_{global}	0.1	V_g buffer size	5000
$n_{\text{walks}}, n_{\text{steps}}$	3, 3	Q_k buffer size	1000

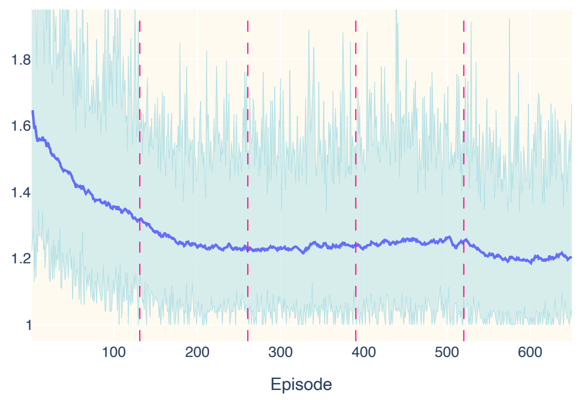


(a) Delay. (b) Energy consumption.

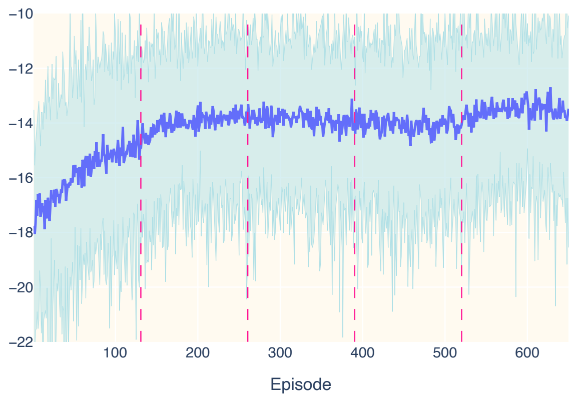
Figure 5.3 The delay and energy consumption results.

Among these, Figures 5.3a to 5.4b, as well as Figures 5.5a and 5.5b, employ the x-axis to denote episodes, i.e., completed tasks. Conversely, Fig. 5.6a uses the x-axis to represent timesteps, or time slots. This distinction arises because performance metrics like delay, energy consumption, and global utility are determined post-task completion, whereas both global and local rewards are designed to enhance these very metrics.

Vertical dashed lines within these figures indicate the timesteps at which task reconfigurations, or changes in the task set, occur. Given that the number of timesteps within each episode can vary, the reconfiguration timesteps in figures with episodes as the x-axis are approximations.



(a) Shortest time ratio.



(b) Global utility.

Figure 5.4 The shortest time ratio and global utility results.

The Shortest Time Ratio (STR) is an essential metric that showcases near-optimal performance. It is defined as (5.10) and its results are depicted in Figure 5.4a.

The STR computes the mean ratio of the delay for each finalized task (τ_k^m) to the shortest achievable delay (τ_{shortest}^m). Notably, the term τ_{shortest}^m shares similarities with τ_{due}^m from (5.2), but with the distinction of having a margin parameter set as $\beta = 0$.

Furthermore, drawing parallels to (5.2), v_{f_k,x_k,y_k} represents the location of AMR-k when the task task^m is allocated to it.

An STR value approximating 1 implies that a majority of tasks are finalized utilizing optimal paths, and a deviation from this value indicates otherwise.

$$\text{STR}^T = \frac{1}{K \sum_{t=1}^{T} M^t} \sum_{t=1}^{T} \frac{\tau_k^m}{\tau_{\text{shortest}}^m} \tag{5.10a}$$

$$\exists \quad \tau_k^m \leftarrow \left(\mathsf{task}^m, \boldsymbol{\rho}_{k,\text{pickup}}^m, \boldsymbol{\rho}_{k,\text{deliver}}^m\right) \tag{5.10b}$$

$$\tau_{\text{shortest}}^m = h_{A^*}\left(v_{f_k,x_k,y_k}, v_{f_p,x_p,y_p}^m\right) + h_{A^*}\left(v_{f_p,x_p,y_p}, v_{f_d,x_d,y_d}^m\right) \tag{5.10c}$$

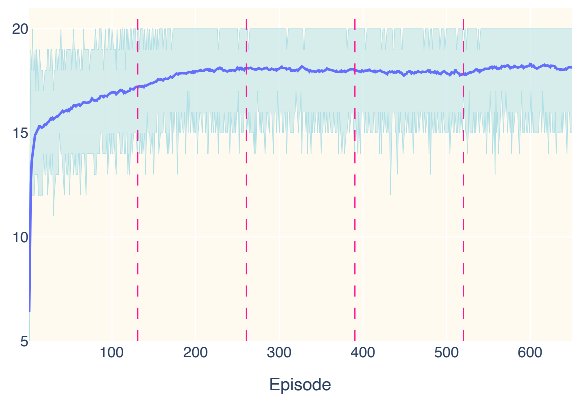
$$m = 1, 2, \dots, M^T \tag{5.10d}$$

Although leveraging the domain model for planning assures near-optimality, AMRs dynamically adapt to avoid potential collisions and coordinate their task execution paths. Figure 5.3a through 5.4b elucidates this:

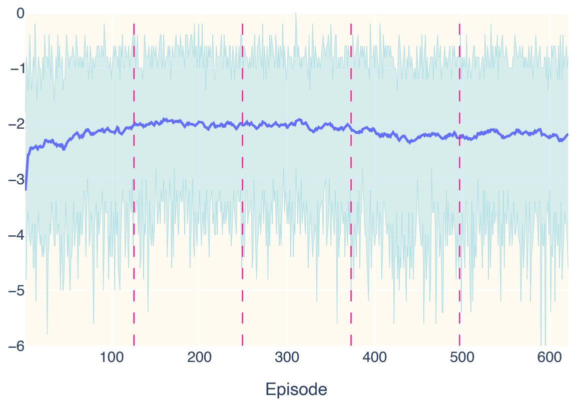
- In the initial 200 episodes, there's a substantial reduction in delay, energy consumption, and STR, implying improved task execution efficiency. Concurrently, there's a noticeable upswing in global utility.

- Beyond the 200 episode mark, even with subtle task set alterations, all four performance metrics exhibit only marginal fluctuations. This behavior indicates that, within the first 200 episodes, AMRs have not only adopted collision-averse strategies but have also adeptly adapted to task reconfigurations to uphold consistent performance levels.

- Pertaining to near-optimality, post the 200 episode threshold, the STR gravitates towards an average of 1.23. This suggests that the mean delay is approximately 23% longer than the ideal optimal delay. Remarkably, this is achieved with 31 AMRs operating in a G_{floor} having 36 vertices (represented as $3 \times 3 \times 4$). This configuration means, on average, one AMR occupies 86% of the space of another AMR.



(a) Global reward.



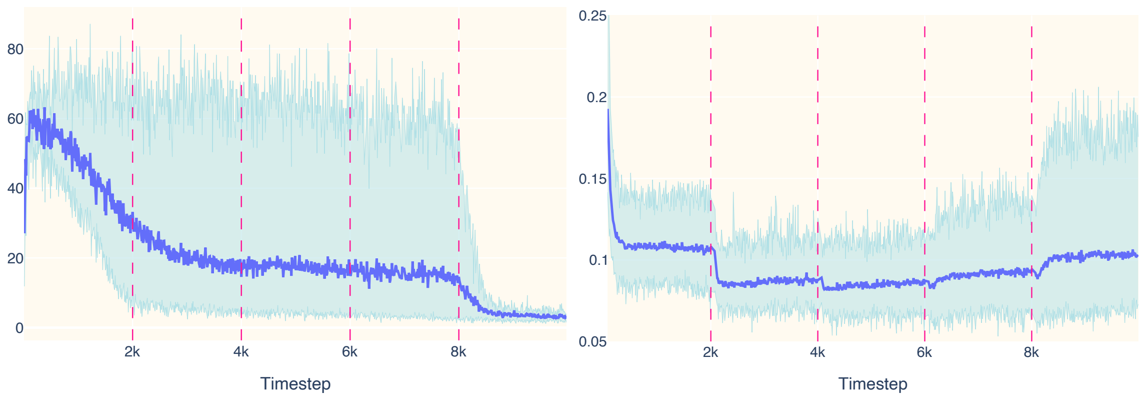
(b) Local reward.

Figure 5.5 The global and local reward results.

Figure 5.5a portrays the average global rewards. Notably, the trend seen here aligns well with the global utility, as presented in Figure 5.4b. This coherence is expected, given that the global critic and global reward are devised to optimize both delay and energy consumption during task execution.

On the other hand, Figure 5.6b showcases the average local reward, which essentially provides insights into the mean number of potential collisions an AMR encounters in each episode. Observations from this figure are as follows:

- During the initial 20 episodes, a substantial enhancement in local rewards is witnessed, with values moving from approximately -3 to -2.
- For the episodes that ensue, the local reward remains relatively stable around the -2 mark. This consistent pattern implies that, in a scenario where 31 AMRs are functioning on a G_{floor} comprised of 36 vertices, one can anticipate encountering two potential collisions in every episode.



(a) MSE loss of the global critic. (b) Smooth L1 loss of local critics.

Figure 5.6 The global and local critic loss results.

Figure 5.6a and Figure 5.6b display the MSE loss for the global critic and the average smooth L1 loss for local critics, respectively. A deeper analysis yields the following insights:

- *Global Critic Loss:* Post the mark of approximately 9000 time slots, the global critic loss exhibits a declining trend, eventually approximating a value close to 3. This decline is noteworthy considering that the global rewards span a relatively broad range ($[0, 10]$). Training on transitions from all 31 AMRs yields a signature indicative of promising convergence.

- *Local Critic Loss Dynamics:* The behavior of the average local critic loss underscores the challenges inherent to AMR path plan coordination in the wake of reconfigurations. An initial sharp decline from 0.25 to 0.11 is observed within the first 400 time slots. Subsequently, each reconfiguration event, delineated by vertical dotted lines, introduces perturbations to the average local critic curve. These perturbations can be attributed to the disruption caused to the coordination previously established by the AMRs, instigating a distributional shift. In the face of reconfigurations, the AMRs are compelled to collaboratively and adaptively forge new coordination.

- *Local Critics' Adaptation:* Pertinent to local critics, the reconfigurations lead to shifts in optimal weights. Consequently, abrupt augmentations or reductions in loss are manifest, which are subsequently optimized courtesy of the replay buffer reset fine-tuning mechanism.

Chapter 6: Operational Resilience and Integrity of MRS in a Smart Factory

This chapter addresses the operational resilience and integrity of MRS under cyber-physical errors with cyber-physical AI agents and smart contracts. Section 6.1 focuses on collaborating to mitigate the impact of production robot actuators' accuracy degradation that leads to defects in the physical domain. Section 6.2 focuses on collaborating to detect incorrect cyber operations by enforcing cyber operations as immutable blockchain transactions by smart contracts.

6.1 Social Learning Coordination of Collaborative MRS for Resilient Production

The accuracy of production robots inevitably degrades over time due to factors such as degradation of mechanical components, onboard sensor measurement noise, and environmental factors including temperature, humidity, dust, or vibration [86]. Traditional factory automation typically carries out system-wide, resource-intensive, and human-extensive calibration to hopefully maintain a consistent global accuracy reference. However, such efforts disrupt factory operations and reduce productivity [32]. On the contrary, AI computing-facilitated MRS coordination offers a more flexible and agile approach to relative accuracy, which avoids the need for frequent system-wide maintenance with a global reference. Each production robot can adjust actuator parameters within its operation range, in real-time based on AI computing outputs. As a result, coordinated task execution leads to the finished products meeting effective production requirements in a more cost-effective and efficient manner.

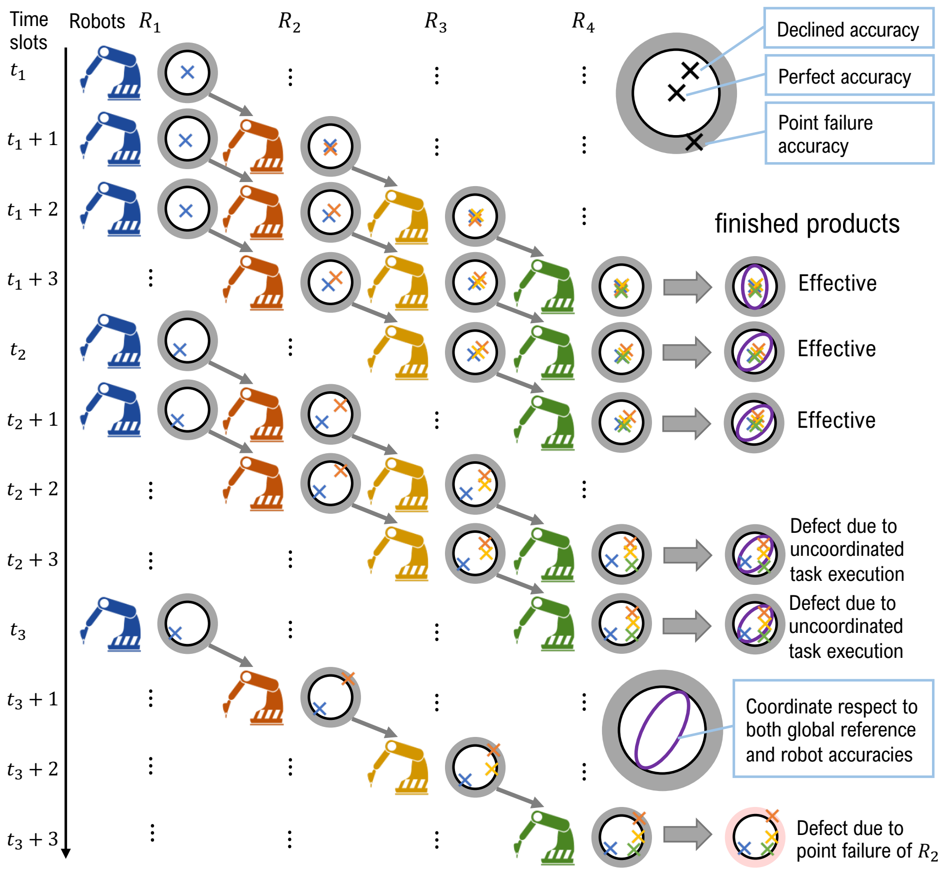


Figure 6.1 Time-slotted MRS for flow-based manufacturing suffers from accuracy degradation without frequent maintenance that impacts the resilient production.

Multi-robot coordination can effectively mitigate the accuracy degradation with the relative accuracy approach. When the completion of a product involves multiple task executions by several robots (also referred to as processes or stages), each task execution is performed by heterogeneous robots with distinct reference systems. As a result, maintaining a globally precise reference among such heterogeneous MRS becomes exceedingly challenging [18]. Consequently, the inspection of finished products relies on the relative alignment of all processes involved in the production, rather than an absolute global reference [102]. By collaboratively coordinating task executions, MRS can minimize such relativity so that the finished products are effective even in the presence of robot accuracy degradation [99]. From t_1 to $t_1 + 3$ in Figure 6.1, although accuracy degradation occurs, the finished products are effective since MRS are performing coordinated task executions. However, from t_2 to $t_2 + 3$, as accuracy continues to degrade, the relativity in multiple task execution increases, which leads to defects. Such defects can be mitigated by MRS within the operational range of the actuator, by reducing the group's relativity through coordination.

The error resulting from accuracy degradation, $e^t_{m,n}$, is characterized by the last maintenance time t_M, Time to Repair ($TTR_{m,n}$), and degradation rate $\mathbf{w}_{m,n}$ [107], assuming that maintenance resets $e^0_{m,n}$ to 0. The degradation is modeled as linear until $e^t_{m,n} = -\gamma_R$ or γ_R, with $TTR_{m,n} \sim \mathcal{U}(t_S, t_L)$. Here, $\mathcal{U}$ represents a discrete uniform distribution defined by the shortest possible maintenance time t_S and the longest possible maintenance time t_L. The degradation and degradation rate, $\mathbf{w}_{m,n}$, are characterized by (6.1a). Consequently, given the errors of all robots at time $t = 0$ as per (6.1b) and their respective degradation rates according to (6.1c), the error of the MRS at time t is given by (6.1d).

$$\mathbf{w}_{m,n} \begin{pmatrix} TTR_{m,n} + t_\mathrm{M} \\ 0 \end{pmatrix} = -\gamma_R \quad \text{or} \quad \gamma_R \tag{6.1a}$$

$$\mathbf{e}^0_{sys} = (e^0_{1,1}, \dots, e^0_{m,n}, \dots) \tag{6.1b}$$

$$\mathbf{w}_{sys} = (w_{1,1}, \dots, w_{m,n}, \dots) \tag{6.1c}$$

$$\mathbf{e}^t_{sys} = \mathbf{w}_{sys} \cdot \begin{pmatrix} \mathbf{t} \\ \mathbf{e}^0_{sys} \end{pmatrix} \tag{6.1d}$$

Furthermore, a production flow is composed of a series of tasks executed by production robots as delineated by MRTA [76]. Given the focus on formulating an AI computing-oriented coordination problem, we employ $a^t_{m,n}$ in (3.4) as a concise representation of task execution. Formally, a production flow $\boldsymbol{\rho}^l$ for product-l is a vector composed of i robots, as specified in (3.1a). This vector is further refined to a simple path in CPMRS in Section 3.1.1. All the i actions $\mathbf{a}^l$ and errors $\mathbf{e}^\mathbf{l}$ contributing to the processing of product-l are provided in (6.2a) and (6.2b) respectively. As depicted in (6.2c), the state of the finished product-l, $p^{l,t+i+1}_{m',n'}$, progresses through all the actions $\mathbf{a}^l$ and errors $\mathbf{e}^l$ temporally, with the robot and number (i) in the defined production flow (3.1a) varying over time. Although accuracy degradation is linearly modeled by (6.1), finished products are processed by a linear combination of robots, each with varying and dynamic accuracy degradations. This results in complex interactions within the MRS and introduces significant non-linearity. The outcome state of a product is influenced by the additive Gaussian noises in measurements and errors in task executions of all production robots in the corresponding production flow.

$$\mathbf{a}^l = (a^{l,t}_{m_1,n_1}, a^{l,t+1}_{m_2,n_2}, \dots, a^{l,t+i-2}_{m_{i-1},n_{i-1}}, a^{l,t+i-1}_{m_i,n_i}) \tag{6.2a}$$

$$\mathbf{e}^{\mathbf{l}} = (e^{t}_{m_1,n_1}, e^{t+1}_{m_2,n_2}, \dots, e^{t+i-2}_{m_{i-1},n_{i-1}}, e^{t+i-1}_{m_i,n_i}) \tag{6.2b}$$

$$\begin{aligned} p^{l,t+i+1}_{m',n'} &= f^{m_i,n_i}_{\text{execute}}(f^{m_{i-1},n-1}_{\text{execute}}(\dots(f^{m_2,n_1}_{\text{execute}} \\ &\quad (f^{m_1,n_1}_{\text{execute}}(p^{l,t}_{m_1,n_1}, a^{t}_{m_1,n_1}, e^{t}_{m_1,n_1}), a^{t+1}_{m_2,n_2}, e^{t+1}_{m_2,n_2}), \\ &\quad \dots), a^{t+i-2}_{m_{i-1},n_{i-1}}, e^{t+i-2}_{m_{i-1},n_{i-1}}), a^{t+i-1}_{m_i,n_i}, e^{t+i}_{m_i,n_i}) \\ &\quad = p^{l,t}_{m_1,n-1} + \varepsilon^{l,t}_{m_1,n_1} + a^{t}_{m_1,n-1} + e^{t}_{m_1,n_1} + \varepsilon^{l,t}_{m_2,n_2} \\ &\quad + a^{t}_{m_2,n-2} + e^{t}_{m_2,n_2} + \cdots + \varepsilon^{l,t}_{m_{i-1},n_{i-1}} + a^{t}_{m_{i-1},n_{i-1}} \\ &\quad + e^{t}_{m_{i-1},n_{i-1}} + \varepsilon^{l,t}_{m_i,n_i} + a^{t}_{m_i,n_i} + e^{t}_{m_i,n_i} \end{aligned} \tag{6.2c}$$

AI computing-enabled MRS coordination can maintain production against the impact of accuracy degradation without necessitating maintenance or calibration, which is referred to as resilient production.

As a practical metric for smart factory production, the effective rate (also referred to as the survival rate) of finished products is defined as the percentage of satisfactory products subject to constraint Δ among all finished products. This definition is inspired by the "yield" definition in [19]. As previously introduced, the satisfactory constraint Δ is determined based on the relativity of task execution considered for finished products. Building upon the modeled task execution (3.4), accuracy degradation (6.1), and subsequent production flow definitions, the effective rate Φ^t for all L finished products can be expressed by equation (6.3).

The effective rate Φ^t is a function of MRS actions $\mathbf{a}^l$ and degradation error $\mathbf{e}^l$, subject to dynamic production flows $\{\boldsymbol{\rho}^1, \boldsymbol{\rho}^2, \ldots, \boldsymbol{\rho}^l, \ldots, \boldsymbol{\rho}^L\}$, while both $\mathbf{a}^l$ and $\mathbf{e}^l$ are vectors of numerical sequences with respect to time slots t. Regardless of the raw material's state, when the range between the maximum and minimum task execution satisfies the predetermined constraint Δ, the finished product is considered effective, and vice versa, which is coherent with the relative accuracy approach proposed in this study. The vector $\mathbf{a}^l$ is determined adaptively and collaboratively by the MRS, rendering (6.3a) a multiagent optimization problem to be addressed by the decentralized AI computing, while the time dynamic production flows $\boldsymbol{\rho}^l$, $l = 1, \ldots, L$, and errors from accuracy $\mathbf{e}^l$ remain unknown. The functions $max(\cdot)$ and $min(\cdot)$ return the maximum and minimum elements of the input vector, respectively, and $\mathbf{1}_\Delta$ is an indicator function. The constraints in (6.3b) pertain to coordination, indicating that the action must be accomplished within the action space and that coordination is only valid in the absence of point failures.

$$\Phi^t(\mathbf{a}^l, \mathbf{e}^l) = \frac{\sum_{l \in L} \mathbf{1}_\Delta : max(\mathbf{a}^l + \mathbf{e}^l) - min(\mathbf{a}^l + \mathbf{e}^l) \ \rightarrow \{0, 1\}}{L} \tag{6.3a}$$

$$\begin{aligned} s.t. \quad & \exists \boldsymbol{\rho}^l, \mathbf{a}^l, \mathbf{e}^l \\ & \forall a^t_{m,n} \in \mathbf{a}^l, a^t_{m,n} \in \mathcal{A} \\ & \forall e^t_{m,n} \in \mathbf{e}^l, e^t_{m,n} \in [-\gamma_R, \gamma_R] \\ & l = 1, \ldots, L \end{aligned} \tag{6.3b}$$

Furthermore, resilience in an engineering context refers to the capacity of a system, component, or infrastructure to withstand, adapt, and recover from disturbances, shocks, or stressors while maintaining its core functionality [118]. This concept is essential for designing and managing systems that can endure uncertain or adverse conditions, such as natural disasters, equipment failures, or human errors, and ensure their continued operation

[65]. Although resilience has no common definition, the definition of resilient production could be derived from the productivity metric, effective rate Φ^t. This study adopts Mean Time To Fail (MTTF) [107] based on the effective rate Φ^t to evaluate resilient production. As resilience is concerned with maintaining core functionality, which, for MRS-driven smart factories, is production, MTTF measures the average time MRS operates before failure, i.e., effective rate decrease to 0. A longer MTTF indicates more resilient production facilitated by MRS facing accuracy degradation. The effective rate Φ^t can be traced in every time slot, as once production tasks are assigned, multiple production flows in MRS continuously generate finished products; however, it is not a function of time slots t. Thus, MTTF is defined by (6.4), which is a composition of Φ^t, $\mathbf{a}^l$, and $\mathbf{e}^l$, with $\mathbf{a}^l$ and $\mathbf{e}^l$ being vectors of time sequences. By maintaining the effective rate Φ^t through MRS coordination in the presence of accuracy degradation, the time until MRS reaches $\Phi^t = 0$ will be longer, indicating more resilient production.

$$MTTF(\mathbf{a}^l, \mathbf{e}^l) = \int_0^\infty \Phi^t(\mathbf{a}^l, \mathbf{e}^l)dt \tag{6.4}$$

After establishing the topology of the cyber domain network, we can infer that production robot $R_{m,n}$ may potentially receive measurements from all other robots, but the quality and frequency of these measurements may vary due to imperfect wireless communications. Consequently, $R_{m,n}$ attempts to perform social learning for all robots in the MRS, utilizing incomplete and time-ineffective information to build the local belief $\mathbf{b}^t_{m,n:sys}$ that becomes increasingly comprehensive over time as given by (3.8c). Furthermore, changes in the physical domain topology, such as alterations to the production flow by MRTA, introduce further diversity into social communications, potentially accelerating the completeness of social measurements for the MRS.

To improve the accuracy and completeness of error belief estimation, the MRS employs the SGD to estimate the degradation weight $\mathbf{w}^t_{m,n:sys}$, which is a local belief of (6.5d) used to update the local belief $\mathbf{b}^t_{m,n:sys}$. The degradation rate $\mathbf{w}^t_{m,n:i,j}$ can be calculated by (6.5d) using the most updated local beliefs, as given by (6.5a). Estimating $\mathbf{w}^t_{m,n:sys}$ is an online stochastic optimization process, defined by (6.5c), which is incrementally updated by the most updated $b^t_{m,n:i,j}$ using the Huber loss function, as given by (6.5b). The Huber loss function is preferred over squared error losses as it is less sensitive to outliers in the data, with δ distinguishing the quadratic interval, and α_{SGD} is the learning rate. The estimated belief of the degradation rate $\mathbf{w}^t_{m,n:sys}$ is given by (6.5d) and the missing beliefs (or missing time indices) can thus be calculated by (6.5a) with $\mathbf{w}_{m,n:i,j}$ and substituting $b^t_{m,n:i,j}$ with $\hat{b}^t_{m,n:i,j}$.

$$b^t_{m,n:i,j} = \hat{\mathbf{w}}_{m,n:i,j} \begin{pmatrix} t \\ b^0_{m,n:i,j} \end{pmatrix} \tag{6.5a}$$

$$\mathbf{L}^t = \begin{cases} \frac{1}{2}(\mathbf{b}^t_{m,n:sys} - \hat{\mathbf{b}}^t_{m,n:sys}) & \text{for } |\mathbf{b}^t_{m,n:sys} - \hat{\mathbf{b}}^t_{m,n:sys}| \leq \delta \\ \delta(|\mathbf{b}^t_{m,n:sys} - \hat{\mathbf{b}}^t_{m,n:sys}| - \frac{1}{2}\delta) & \text{otherwise} \end{cases} \tag{6.5b}$$

$$\begin{pmatrix} \mathbf{w}^t_{1,1:sys} \\ \vdots \\ \mathbf{w}^t_{m,n:sys} \\ \vdots \\ \mathbf{w}^t_{M,N_M:sys} \end{pmatrix} = \begin{pmatrix} \mathbf{w}^{t-1}_{1,1:sys} \\ \vdots \\ \mathbf{w}^{t-1}_{m,n:sys} \\ \vdots \\ \mathbf{w}^{t-1}_{M,N_M:sys} \end{pmatrix} - \begin{pmatrix} \alpha_{SGD} \bigtriangledown_{w^T} \hat{\mathbf{L}}^t_{1,1:sys} \\ \vdots \\ \alpha_{SGD} \bigtriangledown_{w^T} \hat{\mathbf{L}}^t_{m,n:sys} \\ \vdots \\ \alpha_{SGD} \bigtriangledown_{w^T} \hat{\mathbf{L}}^t_{M,N_M:sys} \end{pmatrix} \tag{6.5c}$$

$$\mathbf{w}^t_{m,n:sys} = (\mathbf{w}^t_{m,n:1,1}, \dots, \mathbf{w}^t_{m,n:m,n}, \dots, \mathbf{w}^t_{m,n:M,N_M}) \tag{6.5d}$$

6.1.1 Adaptive Coordination for Resilient Productivity by RL

The local beliefs $\mathbf{b}^t_{m,n:sys}$ and $\mathbf{w}^t_{m,n:sys}$ may not be complete or precise, but they suffice to coordinate actions within collaboration groups via RL, denoted by $f^{m,n}_{\text{computed}}$ in (6.2c).

As illustrated in Figure 6.2, the implemented RL algorithm comprises a main policy $\pi_{\text{main}}(\mathbf{b}^t_{m,n;sys})$, two sub-policies π_{sub1} and π_{sub2}, and a state-action value function $Q^t_{m,n}$. The main policy π_{main} takes $\mathbf{b}^t_{m,n;sys}$ as the state and returns π_{sub1} when $R_{m,n}$ has no local belief about itself and π_{sub2} otherwise. Both π_{sub1} and π_{sub2} take local evidence $x^{l,t}_{m,n}$(from (5.3a))

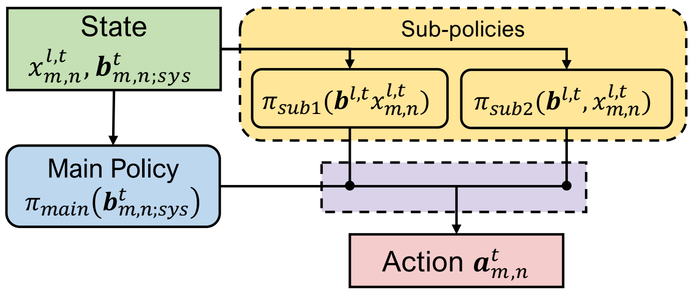

State
$x^{l,t}_{m,n}, \boldsymbol{b}^t_{m,n;sys}$
Sub-policies
$\pi_{sub1}(\boldsymbol{b}^{l,t} x^{l,t}_{m,n})$
$\pi_{sub2}(\boldsymbol{b}^{l,t}, x^{l,t}_{m,n})$
Main Policy
$\pi_{main}(\boldsymbol{b}^t_{m,n;sys})$
Action $\boldsymbol{a}^t_{m,n}$

Figure 6.2 Reinforcement Learning structure for MRS coordination.

and $\mathbf{b}^t_{m,n;sys}$ as states to maximize the future reward $r^{l,t}$ shared among collaboration group. π_{sub1} takes coordinated actions based on the error beliefs of other robots, while π_{sub2} takes actions based on the error beliefs of itself. Both policies share the same state-action value function Q, which is updated with a learning rate α_q and discount factor η, given by (6.6).

$$Q^t_{m,n}(\mathbf{b}^t, a^t) = Q^{t-1}_{m,n}(\mathbf{b}^t, a^t) + \alpha_q[r^{l,t} + \eta \max_a Q^{t-1}_{m,n}(\mathbf{b}^{t+1}, a) - Q^{t-1}_{m,n}(\mathbf{b}^t, a^t)] \tag{6.6}$$

Rewards play a critical role in determining the performance of the Q-function, as they specify how the function generalizes the expected rewards with respect to both the state and action space. In this case, rewards are determined based on the inspection of finished products from collaborating robots at each time slot, where a greater value indicates a higher yield and vice versa. This concept is expressed in (6.7), which is defined following (6.2a). In this equation, $\bar{\mathbf{a}}^l$ represents the mean of $\mathbf{a}^l$, I denotes the total number of actions (i.e., the total number of robots) processing product l, and β_a and β_b are coefficients empirically determined based on the learning rate. It is important to note that a robot may take different actions when involved in multiple production flows, depending on the local evidence and rewards associated with each flow.

$$r^{l,t} = \exp\left(\left(\frac{1}{\beta_a}\sqrt{\frac{\sum_i^I (a^{l,t+i-1}_{m_i,n_i} - \bar{\mathbf{a}}^l)^2}{I}}\right) + \beta_b\right) \tag{6.7}$$

The RL-based MRS coordination enables production robots to optimize productivity even under incomplete $\mathbf{b}^t_{m,n;sys}$. Algorithm 6.1 summarizes the synthesis of state formulation and RL-based MRS coordination on $R_{m,n}$, which includes functions $\mathsf{SocialLearning}$ (equation (3.8)), SGD (equation (6.5)), and $\mathsf{UpdateBelief}$, and functions RL and $\mathsf{UpdateQ}$ (equation (6.6)).

The implementation is executed with respect to a maximum number of social steps during which MRS computes "straightforward" actions based solely on local measurements. At the beginning of each reconfiguration, MRS performs social communications and "straightforward" actions. Once the maximum social steps are reached, MRS performs social learning and SGD to form or update beliefs for RL, enabling adaptive and collaborative AI decision-making, as indicated by lines 4, 5, 12, 13, 21 and 22. Once social communication terminates, MRS employs the SDG to update the belief (line 19) because the prior in social learning is no longer accurate when MRS takes actions with RL.

6.1.2 Experiments and Numerical Results

Computational experiments are conducted to evaluate productivity by the effective rate and resilience of the MTTF. To simulate time-varying production demands and MRTA defining production flows, $\boldsymbol{\omega}$ and λ_m^l from [76] are used, which are unknown for MRS. The noisy measurement $f_{\text{measure}}^{m,n}(p_{m,n}^{l,t})$ in (3.4a) is implemented using a Gaussian distribution with a mean of $p_{m,n}^{l,t}$ and a variance of 0.2. The unobservable $\mathbf{e}_{sys}^t$ is initially set to all zeros at $t = 0$, and $\overline{TTR}$ is the mean of $TTR_{m,n}$ that characterizes the accuracy degradation in (6.1a). The wireless channels are modeled with a 10% error rate, while social communications are not re-transmitted or back-forwarded. The experiments are conducted with the parameters in Table 6.1, and the results are presented as the mean curves of 5 repetitions over 5 distinct sets of random seeds, with the shades representing the range of the results due to the randomness introduced by the stochastic, partially connected cyber domain network and MRTA.

The first experiment aims to investigate the impact of accuracy degradation on MRS coordination problems and to evaluate the effectiveness of the proposed CPMRS model. To achieve this goal, we compare the implemented RL-based MRS coordination with a baseline

Algorithm 6.1: RL algorithm on $R_{m,n}$

```
Data: $G^t_{phy}$, $G_{cyb}$, $MRTA$, $\overline{TTR}$, $\mathbf{p}^t_{m,n}$, $\mathcal{A}$, $\alpha_{SGD}$, $\alpha_q$, $\delta$, $\eta$,
      $\pi_{\text{main}}$, $\pi_{sub1}$, $\pi_{sub2}$, $Q^t_{m,n}$, max_social_steps
Reconfiguration effect until $t = T$
physical_neighbors $\leftarrow G^t_{phy}$
social_neighbors $\leftarrow G_{cyb}$
social_steps $= 0$
learning_flag $=$ False
for $t$ in $T$ :
    $x^t_{m,n} \leftarrow f^{m,n}_{\text{measure}}(p^t_{m,n})$
    $\mathbf{y}^t \leftarrow$ social_neighbors
    if social_steps $<$ max_social_steps :
        $a^t_{m,n} \leftarrow f^{m,n}_{\text{compute}}(\mathbf{x}^t)$
        social_steps $+= 1$
    elif social_steps $\geq$ max_social_steps :
        if not learning_flag :
            $\mathbf{b}^t_{m,n:sys} \leftarrow$ SocialLearning($\mathbf{x}^t, \mathbf{y}^t$)
            $\mathbf{w}^t_{m,n:sys} \leftarrow$ SGD($\mathbf{b}^t_{m,n:sys}, \alpha, \delta$)
            $\mathbf{b}^t_{m,n:sys} \leftarrow$ UpdateBelief($\mathbf{b}^t_{m,n:sys}, \mathbf{w}^t_{m,n:sys}$)
            learning_flag $=$ True
        else:
            $\mathbf{b}^t_{m,n:sys} \leftarrow$ UpdateBelief($\mathbf{b}^t_{m,n:sys}, \mathbf{w}^t_{m,n:sys}$)
        $a^t_{m,n} \leftarrow$ RL($x^t_{m,n}, \mathbf{b}^t_{m,n:sys}, a^{t-1}_{m,n}, \pi_{main}, \pi_{sub1}, \pi_{sub2}, Q^t_{m,n}$)
    $p^{t+1}_{i,j}, r^{l,t} \leftarrow f^{m,n}_{\text{execute}}(p^t_{m,n}, a^t_{m,n}, e^t_{m,n})$
    UpdateQ ($\mathbf{b}^t_{m,n:sys}, \mathbf{b}^{t-1}_{m,n:sys}, a^t_{m,n}, r^{l,t}, \alpha_q, \eta$)
    if reconfigure :
        $G^t_{phy} \leftarrow MRTA$
        social_steps $= 0$
        learning_flag $=$ False
```

method, referred to as "bare MRS", which only takes "straightforward" actions without AI computing. In addition, "burst errors" are introduced in the experiments to test the resilience of MRS to out-of-distribution or un-modeled noises. Burst errors are sudden errors that occur at five randomly selected time slots, specifically $(20, 60, 100, 140, 180)$, which are not known to the MRS and are not modeled in the degradation model. At each burst error, one randomly selected robot will suddenly have an error of either 10 or -10 with a probability of 0.5 each. These burst errors violate the modeled accuracy degradation and

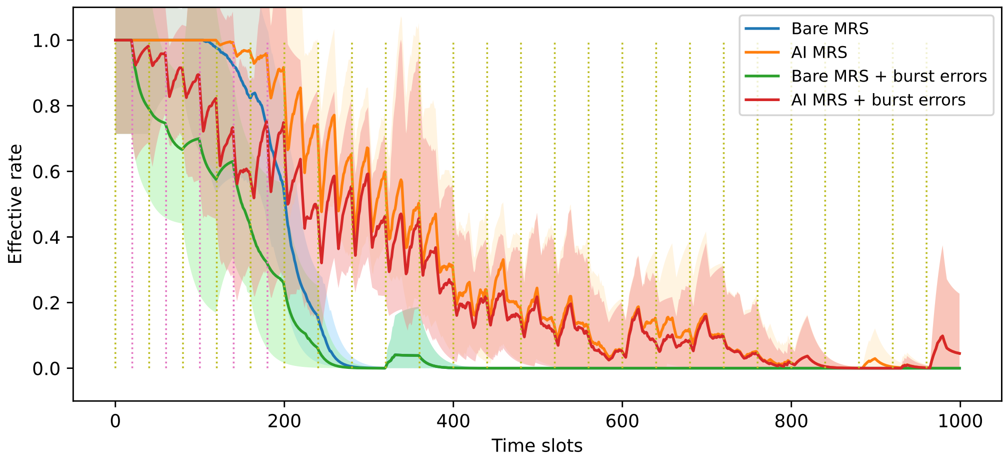
Table 6.1 Parameters of resilient production robots experiments.

Parameters	Values	Parameters	Values
M	6	α_{SGD}, α_q	$0.001, 0.1$
$\mathbf{N}$	$(5, 5, 5, 5, 5, 5)$	β_a, β_b, β_q	$5, 2, 0.1$
$\boldsymbol{\omega}$	$(6, 5, 6, 5, 7, 6)$	Δ	10
L	$\mathcal{U}(1,5)$	δ	4
λ_m^l	$\mathcal{U}(0,8)$	η	0.9
Ω_p	$[-10, 10]$	z	5
σ_{measure}	0.2	Reconfiguration	25 times
γ_R	5	Interval (in time slots)	40
$\overline{TTR}$	200	Social composite degree	$(4, 0.25, 0.75)$
$\mathcal{A}$	$[-5, 5]$	Wireless error rate	0.1

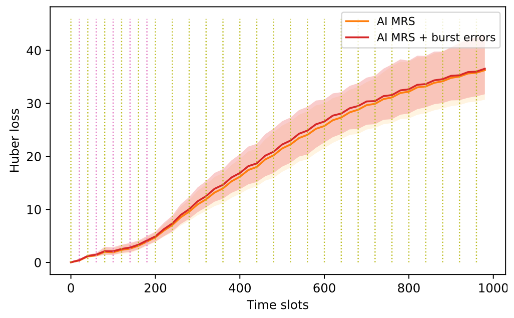
can occur in real-world smart factories due to cyber attacks that sabotage a small number of robots, causing point failures and rapidly decreasing the effect rate to 0.

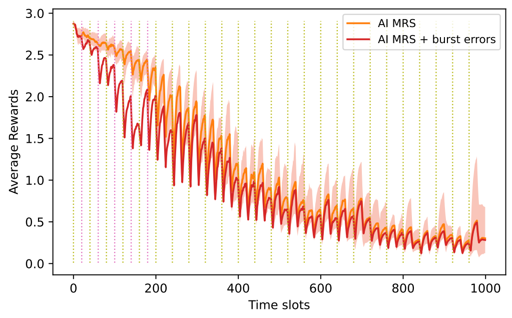
Figs. 6.3a - 6.3d present the results of the aforementioned experiment. In Figure 6.3a, four groups labeling “bare MRS”, “AI MRS”, “bare MRS + burst errors” and “AI MRS + burst error” are compared, with respective MTTF values of 190.48, 350.00, 125.50 and 282.10. Reconfigurations are indicated by yellow vertical dot lines and burst errors by pink vertical dot lines. During the first 100 time slots, “bare MRS” and “AI MRS” achieve an effective rate of 1, indicating that accuracy degradation has not caused finished products to become defective. However, after 100 time slots, both curves begin to decrease, and with burst errors, this happens after 20 time slots. Table 6.1 shows that since $\overline{TTR} = 200$, bares MRS fails before the mean of TTR, and even earlier with burst errors. The implemented RL-based coordination, on the other hand, results in 83.75% and 124.78% improvements in MTTF, respectively.

Furthermore, as shown in Figs. 6.3b and 6.3c, although the RL-based coordination greatly improves MTTF, it cannot fully compensate for the effects of accuracy degradation on MRS, as suggested by the increasing loss and decreasing rewards over time. Finally, Figure 6.3d demonstrates how MRTA-driven reconfigurations make the raised coordination problem non-stationary through Root Mean Squared Error (RMSE) of the Q function. Each



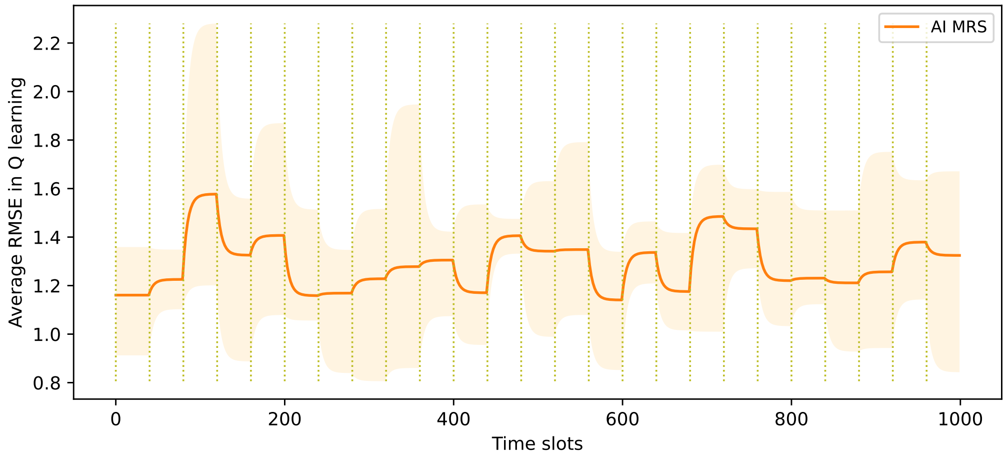
(a) MRS effective rate Φ^t.





(b) SGD Huber loss.

(c) RL average rewards.



(d) RL average RMSE of Q function

Figure 6.3 Numerical results of computational experiments showing the impact of raised accuracy degradation problem and how MRS coordination effectively mitigates the problem without frequent maintenance.

reconfiguration indicated by the vertical line leads to a shift in the optimal Q function, and the RL framework with stochastic optimization successfully adapts to such changes.

The second experiment aims to demonstrate the scalability of the proposed CPMRS and RL-based MRS coordination by evaluating the MTTF of MRSs with different numbers of production robots and burst errors. The experiments are conducted on MRSs with $(17, 23, 29, 37, 43, 47)$ production robots and burst errors of $(0, 2, 4, 6, 8, 10)$. The results, as shown in Figure 6.4, indicate that when MRSs have burst errors from 0 to 5, the MTTF decreases as the number of robots increases, which can be attributed to the fact that larger MRSs can accommodate more production flows and are thus more capable of handling accuracy degradations. However, when MRSs have burst errors from 6 to 10, the impact of burst errors becomes more pronounced for smaller MRSs, resulting in an increase in MTTF as the number of robots increases. This is because the number of burst errors compared to the total number of robots is relatively small when the MRS is large enough. These results demonstrate the scalability of the proposed CPMRS and RL-based coordination, which can effectively coordinate a large number of production robots in smart factories and ensure resilience in the face of burst errors.

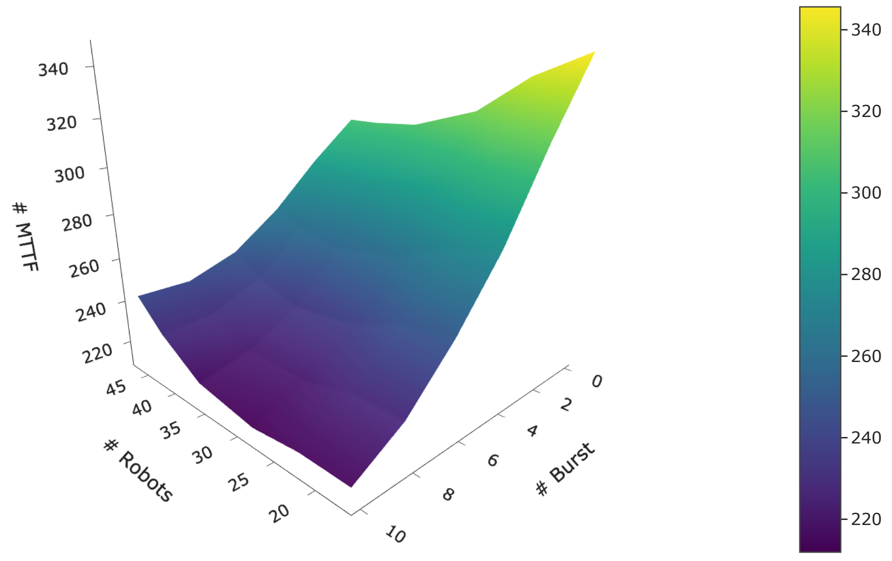


Figure 6.4 MTTR in terms of the size of MRS and a number of burst errors.

While a comparison study can provide additional insights, it is not included in this study as the implementation is based on the proposed CPMRS model, making it difficult to trace. Nonetheless, the optimality of the approach can be observed from the results, with the optimum effective rate achieving 1. This study's primary contribution is the introduction of the CPMRS model, which enables adaptive and collaborative MRS coordination to address productivity issues. The results demonstrate that social communications facilitate the estimation of unobservable errors. Furthermore, the RMSE of the Q functions indicates that CPMRS allows stochastic optimization by enabling data collection from social communications. This systematic design and analysis have not been precisely studied in the literature.

6.1.3 Toward Group Decision for Predictive Maintenance

While the proposed MRS coordination method mitigates accuracy degradation and circumvents the need for frequent maintenance and calibration by prioritizing relative accuracy, it is less effective in dealing with point failures. To address this concern, a group decision-based predictive maintenance approach [127] is proposed to further improve resilient production, which only mandates maintenance for robots identified as potential points of failure by the collective decision of the MRS.

As elaborated in Section 3.1.1.2, the maintenance procedure in MRS-driven smart factories necessitates system-wide changes including disruptions to the production flows, adjustments to the production flows, and reductions in the number of operational robots. Such comprehensive changes require a group decision-making process, facilitated by edge computing. In this regard, the proposed approach seeks to maintain only those production robots that demonstrate significant accuracy degradation or point failure, as identified by predictive group decisions to limit the cost of maintenance.

Classical consensus algorithms in distributed systems, which are typically designed for group decision-making, are based on fully connected cyber topologies and are not generally applicable to the CPMRS model. Furthermore, as Section 3.1.1.2, only collaborative MRSs can detect decreased accuracy, and edge computing cannot directly observe it. Although consensus is unattainable over a partially connected cyber network, edge computing can still gather trust from all robots and formulate group decisions regarding necessary maintenance. This approach is illustrated in Figure 6.5 with a 4-robot-system, where the edge computing gathers trust vectors from all robots (R_1 to R_4) as a trust matrix. In particular, this figure indicates that R_2 is the least trusted robot and is thus causing point failures that need to be addressed through maintenance.

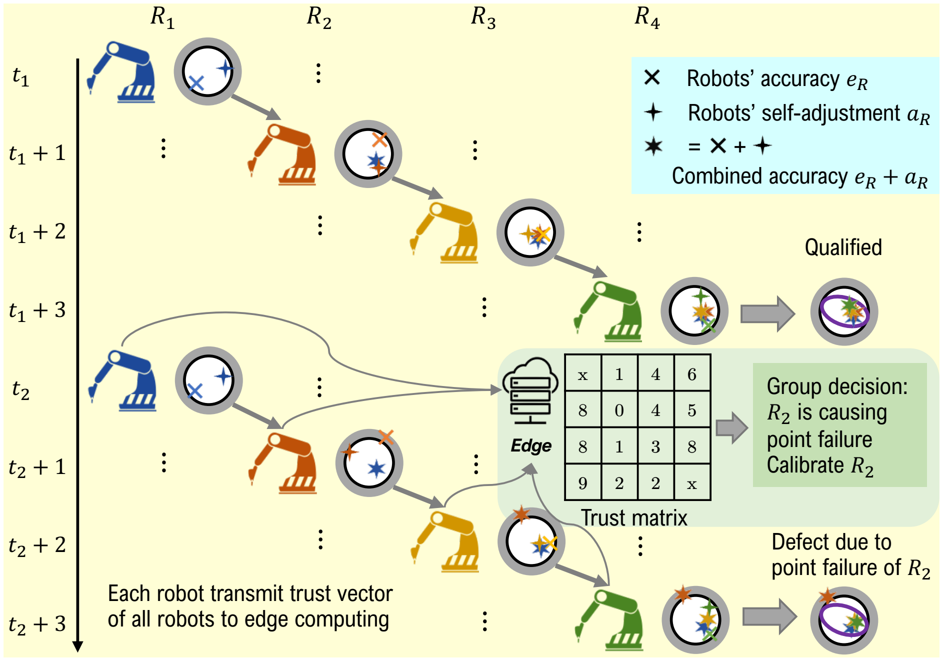

R_1
R_2
R_3
R_4
t_1
t_1+1
t_1+2
t_1+3
t_2
t_2+1
t_2+2
t_2+3
× Robots' accuracy e_R
✦ Robots' self-adjustment a_R
✶ = × + ✦
Combined accuracy $e_R + a_R$
Qualified
Edge
x 1 4 6
8 0 4 5
8 1 3 8
9 2 2 x
Trust matrix
Group decision: R_2 is causing point failure Calibrate R_2
Defect due to point failure of R_2
Each robot transmit trust vector of all robots to edge computing

Figure 6.5 The group decision-based predictive maintenance.

To make the maintenance decisions, the edge computing collects a trust vector $\mathbf{o}^t_{m,n;sys}$ from all robots, which predicts the number of time slots until significant accuracy degradation for each robot based on error beliefs $\mathbf{b}^t_{m,n:sys}$ and $\mathbf{w}^t_{m,n;sys}$. The trust value $o^t_{m,n:i,j}$ is calculated from (6.8a) and stored in the local trust vector $\mathbf{o}^t_{m,n;sys}$. The edge computing then collects all local trust vectors to form a trust matrix OM^t as in (6.8c) and uses the auto-correlation of OM^t to determine which robots need maintenance. After the trust matrix, OM^t is computed according to (6.8c), edge computing selects z robots with the least average trust as the ones causing point failure, which requires scheduled maintenance. The performance of the decision is evaluated through the auto-correlation of OM^t using (6.8d) and (6.8e). Group decision-making can be seen as an information aggregation process on edge computing, leading to near-optimal decisions. Compared to centralized methods, information aggregation only occurs when a maintenance decision is necessary, for instance when point failures are detected (but their location cannot be determined) or when a system-wide reconfiguration is triggered by new production demands. This approach results in limited, regulated wireless communications and a well-balanced computation load between edge computing and mobile computing, contributing to the resilience of the system.

$$\mathbf{w}^t_{m,n;i,j} \begin{pmatrix} t + o^t_{m,n:i,j} \\ \mathbf{b}^t_{m,n;i,j} \end{pmatrix} = -\gamma_R \text{ or } \gamma_R \tag{6.8a}$$

$$\mathbf{o}^t_{m,n;sys} = (o^t_{m,n;1,1}, \dots, o^t_{m,n;m,n}, \dots, o^t_{m,n;M,N_M}) \tag{6.8b}$$

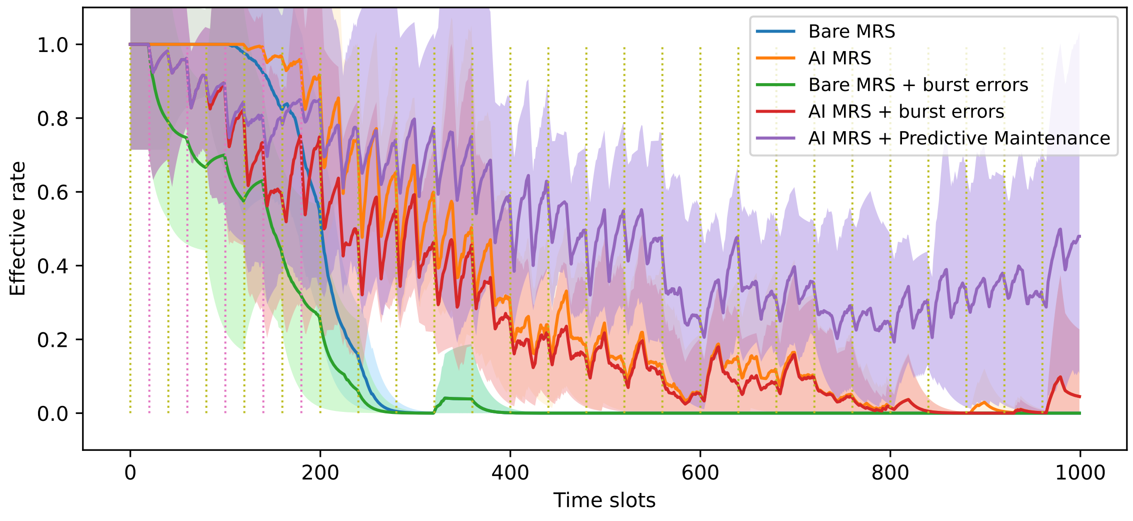
$$OM^t = (\mathbf{o}^t_{1,1;sys}, \dots, \mathbf{o}^t_{m,n;sys}, \dots, \mathbf{o}^t_{M,N_M;sys}) \tag{6.8c}$$

$$CR^t = \begin{pmatrix} cr^t_{1,1;1,1} & \cdots & cr^t_{1,1;m,n} & \cdots & cr^t_{1,1;M,N_M} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ cr^t_{m,n;1,1} & \cdots & cr^t_{m,n;m,n} & \cdots & cr^t_{m,n;M,N_M} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ cr^t_{M,N_M;1,1} & \cdots & cr^t_{M,N_M;m,n} & \cdots & cr^t_{M,N_M;N_M,M} \end{pmatrix} \tag{6.8d}$$

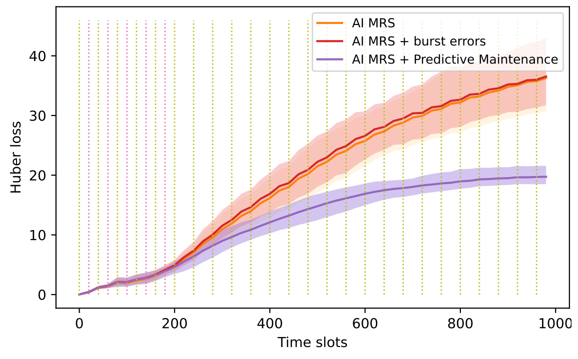
$$cr^t_{m,n;i,j} = \frac{Cov(\mathbf{o}_{m,n;sys}, \mathbf{o}_{i,j;sys})}{\sigma_{m,n;sys}\sigma_{i,j;sys}} \tag{6.8e}$$

In this section, the group decision approach is evaluated in terms of its ability to achieve predictive maintenance and resilience in MRS-driven smart factories. First, the same experiment conducted in Section 6.1.2 is repeated with the value of z suggested by Table 6.1. As depicted in Figure 6.6, the AI MRS with predictive maintenance is able to maintain Φ^t, the SGD loss, and RL rewards even under burst errors, fulfilling the definition of a resilient system capable of recovering from failures and achieving new stability.

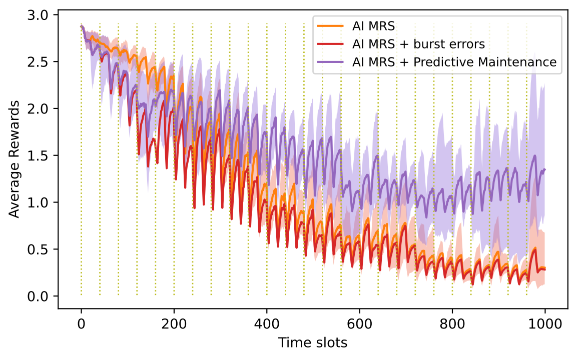
Regarding optimality, please note that the group decision approach does not guarantee global optima, which goes beyond the scope of this study. However, since the ideal optimum for Φ^t is 1, the empirical observation from Figure 6.6a demonstrates that the group decision approach achieves an MTTF of 523.68, which indicates a 317.27% improvement from the bare MRS and 85.64% improvement from the AI MRS.



(a) MRS effective rate Φ^t.



(b) SGD huber loss.



(c) RL average rewards.

Figure 6.6 Experiment results on the predictive maintenance based on the group decision-making.

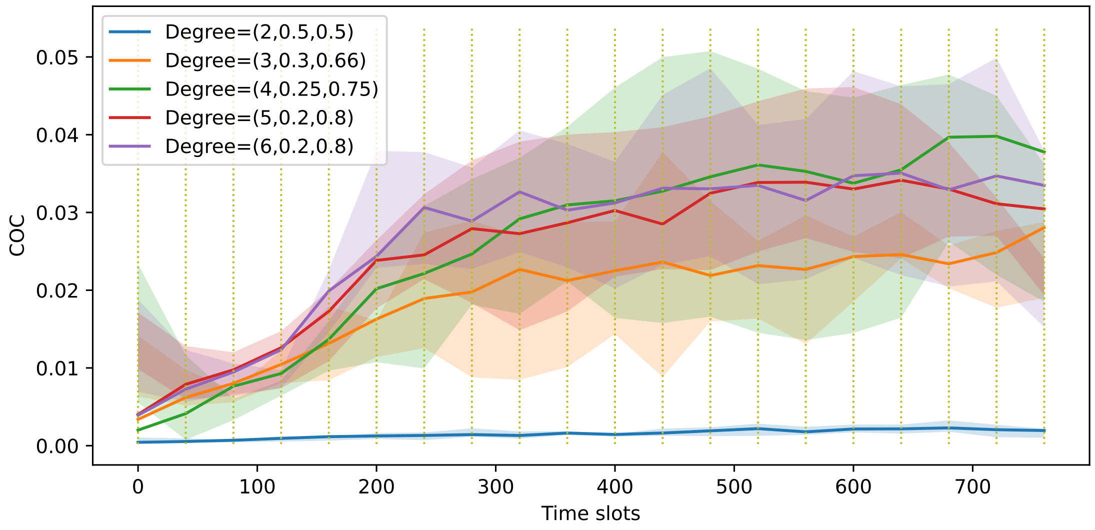
Furthermore, since the maintenance decision is made on the edge server from the trust vector of each robot, the partially connected cyber domain network leads to incomplete trust vectors and thus an incomplete opinion matrix on the edge server, which makes the precise group decision-making challenging. We analyze the trade-off between composite social degrees and group decision-making with results shown in Figure 6.7a - 6.7c on a 43-robot system with $\mathbf{N} = (6, 7, 7, 7, 7, 7)$.

The coherence of consensus (COC) is adopted to evaluate the quality of group decision-making from the mean of the product of the opinion matrix and effective trust values given by (6.9) [79]. In this equation, f_O calculates the ratio of successfully trusted robots to the total number of robots, since the trust vector is not complete due to partial observability. Please note that the coherence of consensus does not mean consensus, and it actually reflects the quality and quantity of group decision-making correlations.

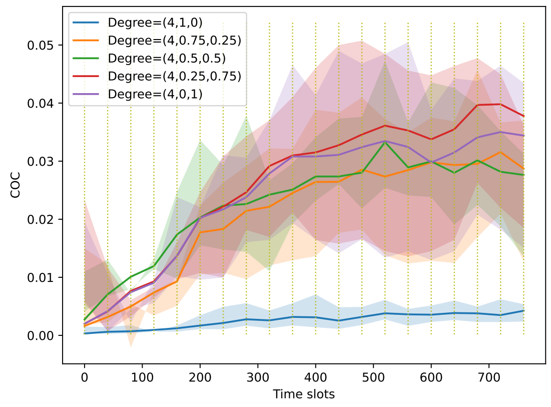
$$\mathrm{COC}^t = \frac{\sum_{(m,n),(i,j)\in G_{cyb}} cr^t_{m,n;i,j}}{\|CR^t\|} \cdot \frac{\sum_{m,n\in G_{cyb}} f_O(\mathbf{o}^t_{m,n;sys})}{\sum_{i\in M} N_i} \tag{6.9}$$

Figure 6.7a presents an analysis of the coherence of consensus with respect to the total number of links d for each robot in the composite degree of the cyber network $\mathcal{D}_{cyb} = (d, kd, (1-k)d)$, where k is the fraction of outgoing links. As the total number of links d increases, significant differences in the coherence of consensus can be observed at $d = 2$, $d = 3$, and $d = 4$ for similar k values (since the number of links is discrete, identical k values cannot be obtained). For $d \geq 4$, the coherence of consensus shows similar performance, which is consistent with the behavior of random network evolution, as the proposed stochastic unified-degree cyber network is a special case of the ER network. According to the evolution of random networks, when $d \geq 4$, the largest connected cluster has a size close to the total number of robots, which demonstrates the effectiveness of the stochastic partially connected cyber network.

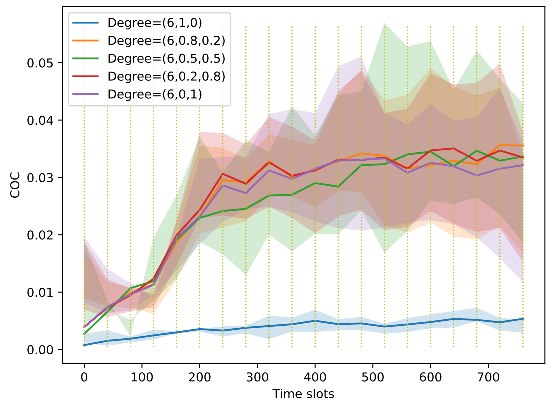
Figs. 6.7b and 6.7c investigate the coherence of consensus as a function of the ratio k between intra-type links and total links, for systems with $d = 4$ and $d = 6$, respectively. In both cases, a significant decrease in the coherence of consensus is observed when $k = 1$, corresponding to systems without any inter-type links. For $d = 4$ and $k < 1$ (Figure 6.7b), the coherence of consensus is higher when the ratio of inter-type links is higher than that of intra-type links. However, for $d = 6$ and $k < 1$ (Figure 6.7c), all groups show similar performance, possibly because the network is more connected than in the case with $d = 4$.



(a) Coherence of consensus of 43-robots MRS with total degree d of 2 to 6.



(b) Coherence of consensus of 43-robots MRS with total degree d of 4.



(c) Coherence of consensus of 43-robots MRS with total degree d of 6.

Figure 6.7 Empirical analysis of the proposed stochastic partially connected social network.

In summary, the empirical results indicate that networks with $d \geq 4$ and $k \leq 0.5$ are advantageous for maintenance decision-making, as they favor fewer intra-type robot links and more inter-type robot links. Consequently, a total of $4\sum_{m=1}^{M} N_m$ social messages are transmitted in each time slot, yielding constant complexity relative to the total number of production robots. The experiments discussed in Section 6.1.2 and this Section demonstrate the impact of accuracy degradation, which can reduce productivity to 0. Moreover, the proposed CPMRS model facilitates social learning and RL-based MRS coordination, resulting in a 124% improvement in MTTF and a 317.27% improvement in MTTF with predictive maintenance. This notable improvement in MTTF signifies resilient production, prevents frequent system-wide maintenance, reduces defects, and consequently, further enhances the productivity and resource efficiency of MRS-driven smart factories.

6.2 MRS Operational Integrity and Privacy Addressed by Smart Contracts

Federated Learning (FL) is a scheme coherent with Multi-Robot Systems (MRS), promoting collaborative machine learning across numerous decentralized robot nodes. However, with this approach, data privacy and operation integrity emerge as principal concerns.

To understand how these concerns are addressed by smart contracts, let's first introduce the concept of a blockchain network. A blockchain network is essentially a decentralized assembly of computers that collectively maintain a storage—or ledger—of transactions [53]. Specifically, within a smart factory, this blockchain network is fully connected. This means that all M edge nodes and N robot nodes possess identical copies of the blockchain.

Elevating the functionalities of blockchain, we have *smart contracts*. They are essentially self-executing contracts with the terms and conditions directly embedded into lines of code. They provide automation and programmability to the blockchain [60].

The decentralized nature of blockchain networks ensures transparency, availability, and immutability, which are crucial for MRS-driven smart factories. Meanwhile, smart contracts offer added layers of automation, trust, integrity, and interoperability. Given these attributes, both blockchain networks and smart contracts present themselves as excellent complements to FL.

The principal technologies underpinning blockchain include cryptographic hash functions [6], a peer-to-peer (P2P) network structure [115], and consensus mechanisms [57]. Cryptographic hash functions process an input message to compute a fixed-length byte string, termed a hash. This hash acts as a “digest” of the input message, remaining consistent for identical inputs while exhibiting drastic alterations upon minor modifications to the input. Moreover, the re-engineering of the original input message from its corresponding hash is computationally prohibitive, making hashes crucial for data integrity. SHA-256 is a prominent hash function employed in numerous blockchain implementations.

Blockchain adopts a P2P network structure, constituting a fully interconnected topology among participant edge nodes and robot nodes. Every node in the network concurrently performs as both a server and client, all possessing equal authority and the capability to independently add and validate transactions and blocks.

The legitimacy of transactions and blocks relies on a consensus mechanism among all participating nodes. Proof of Work (PoW) [87] is a frequently utilized consensus mechanism in which the node proposing the addition of a new block to the blockchain must solve a complex mathematical problem. The block is appended to the blockchain only after all other nodes in the network validate the solution to the problem. As the problem is designed to be computationally challenging to solve yet straightforward to verify, the computation overhead is scalable.

Blockchain technology provides transparency and availability through its inherent structure and operation. All participating nodes in the blockchain network poccess identical blockchain databases except for unconfirmed transactions all the time, which ensures that all the nodes have access to the same information stored in the blockchain without centralized authorization. Also, typical implementations of blockchain involve relevant details such as timestamps, transaction amounts, and participating nodes’ identification (IP addresses or other IDs), enabling effortless tracking and validations.

Blockchain’s immutability is facilitated by cryptographic hash functions, while block validity is assured through consensus mechanisms. Cryptographic hash functions play vital roles in cybersecurity, serving numerous applications including data integrity validations, password storage, digital signatures, and blockchain technology, among others [117]. A

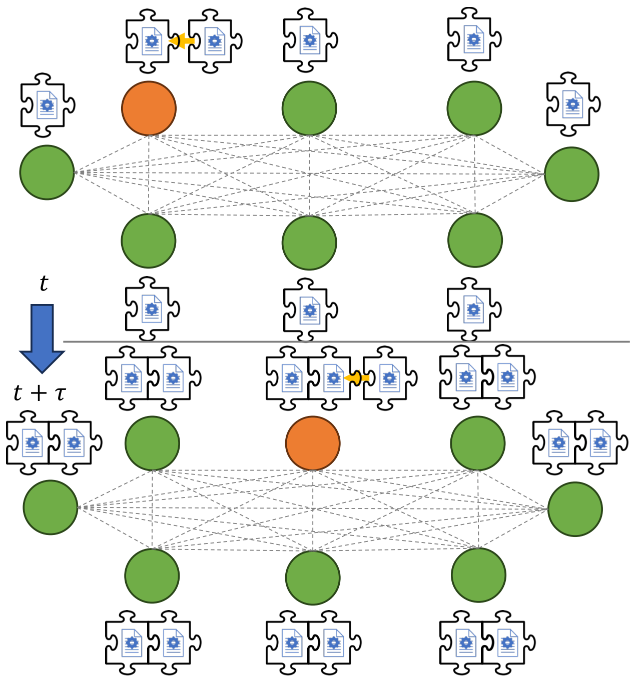


Figure 6.8 Blockchain with smart contracts and DPoW.

hash function, represented as $H = f_{\text{hash}}(\text{message})$, inputs a message and conducts a one-way computation that outputs a deterministic, fixed hash, typically through bitwise operations, modular arithmetic, and logical functions. Proof of Work (PoW), a consensus mechanism commonly used in blockchain networks, notably Bitcoin, mandates that participating nodes (also known as miners) carry out computational work, such as solving a complex mathematical problem, to authenticate and append new blocks of transactions to the blockchain. Consequently, once a transaction or block is appended and validated by the network, it transforms into a permanent and unmodifiable record, thereby conferring blockchain its inherent characteristic of immutability.

Moreover, smart contracts are encoded using dedicated programming languages such as Solidity for Ethereum [50]. These contract terms and conditions can be activated by elements in the blockchain, and their execution is ensured by transactions and blocks within the blockchain. Once deployed, smart contracts automatically execute according to predefined codes and upon the fulfillment of certain conditions, thereby facilitating automation in the cyber domain of smart factories, including Federated Learning (FL). Transactions and blocks governed by smart contracts bring about trust and integrity, as they provide a transparent and unalterable record of all contract-related events under specified terms. In addition, smart contracts can function as protocols defining the interaction among robots, such as data exchange and conditional actions. This capacity promotes interoperability among distinct robots from various manufacturers.

As shown in Algorithm 6.2 and Figure 6.8, the smart factory's MRS employs Delegated Proof of Work (DPoW) as a consensus mechanism to reduce computational demand, thereby enhancing scalability. This delegation process is both conducted and enforced via a smart contract. Algorithm 6.2 presents a round-robin style delegation of the PoW computation to the robot nodes. The mining process, or the addition of new blocks to the blockchain, is triggered by the fundamental steps inherent to FL and executed by delegated nodes.

Algorithm 6.2: Blockchain with Delegated Proof of Work (DPoW)

```
def DPoW ($M, N$, blockchain, pendingTXN, lastExecutor):
    lastRobot ← 0
    lastEdge ← 0
    while True:
        if pendingTXN is not empty:
            newBlock.TXN = pendingTXN
            newBlock.prevBlockHash = blockchain[-1].blockHash
            newBlock.miner = lastExecutor
            newBlock.nonce = findNonce(newBlock)
            newBlock.blockHash = $f_{\text{hash}}$(newBlock)
            blockchain.append(newBlock)
            pendingTXN = empty
            if lastExecutor is Edge:
                executor ← Next(last_edge)
                lastEdge ← executor
            elif lastExecutor is Robot:
                executor ← Next(last_robot)
                lastRobot ← executor
            yield executor
```

6.2.1 Smart Contracts for the MRS Operational Integrity

While the key management mechanism is specifically engineered to preserve privacy during model exchange, its execution—along with other steps in Federated Learning (FL)—is enforced by smart contracts to ensure integrity. The implementation of smart contracts in an FL framework, which incorporates M edge nodes (serving as FL controllers) and N robot nodes (acting as FL clients), provides a foundation for both the integrity and privacy preservation associated with the five critical stages of FL.

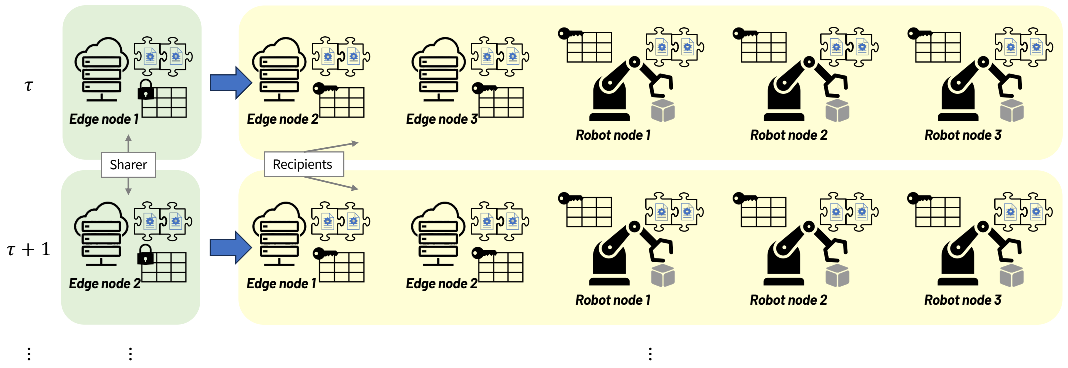


Figure 6.9 Global model distribution with delegated edge node as a sharer and all other nodes as recipients.

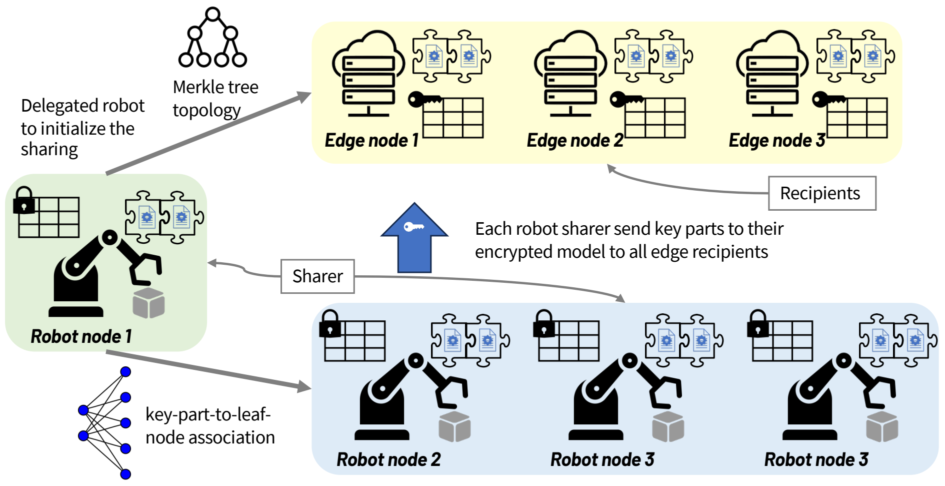

Merkle tree topology
Delegated robot to initialize the sharing
Edge node 1
Edge node 2
Edge node 3
Recipients
Each robot sharer send key parts to their encrypted model to all edge recipients
Sharer
Robot node 1
key-part-to-leaf-node association
Robot node 2
Robot node 3
Robot node 3

Figure 6.10 Local model upload with delegated robot node initialize the share, all robot nodes as shares and all edge nodes as recipients.

During the beginning global model distribution phase, as depicted in Figure 6.9, a smart contract is used to designate an edge node to carry out the model distribution. This is accomplished by following the process outlined in Algorithm 6.2. The updated global model is encrypted, and the corresponding key is split into several parts. This key fragmentation results in a Merkle tree with an associated key part-leaf node mapping. Each recipient is then sent a uniquely assigned key part, the Merkle tree topology, and a necessary hash set. Simultaneously, a blockchain transaction is added, involving the hash of the entire key to the encrypted model within it.

Algorithm 6.3: Smart contract for integrity FL.

```
   lastExecutor ← 0
   while True :
       /* Global model update, synchronization and distribution            */
1      $m \leftarrow$ DPoW($M, N$, blockchain, pendingTXN, lastExecutor)
2      $key \leftarrow$ KeyGen
3      DelegatedEdge($m, \boldsymbol{\theta}_m, key, N$) for $i$ in $M+N$ :
4          if $i == m$ :
5              continue
6          Recipients($i, MT_i, H_i, \gamma_i, \boldsymbol{\theta}_{m,E}$)
7          $i$ execute model distribution.
       /* Local training                                                   */
8      for $i$ in $N$ :
9          $dataset\_id, p_i \leftarrow i$ execute local training.
10         addTXN($i, dataset\_id, p_i$)
       /* Local model upload                                               */
11     $n \leftarrow$ DPoW($M, N$, blockchain, pendingTXN, $m$)
12     DelegatedRobot($n, N$)
13     for $i$ in $N$ :
14         $key \leftarrow$ KeyGen
15         RobotSharer($i, \mathbf{w}_i, key_i, \phi_n$)
16     for $i$ in $M$ :
17         EdgeRecipient($i, MT_i, \{\gamma_{1,i}, \dots, \gamma_{N,i}\}, \mathbf{w}_{1:N,E}$)
       /* Model aggregation                                                */
18     $m \leftarrow$ DPoW($M, N$, blockchain, pendingTXN, $n$)
19     $\tau \leftarrow m$ execute model aggregation.
20     addTXN($\tau, m$)
```

Each recipient node, including the remaining $M - 1$ edge nodes and the N robot nodes, responds by adding a transaction that includes the calculated Merkle root, derived from their individual key parts, the Merkle tree topology, and the necessary hash set. It is only when all the Merkle roots align consistently that all recipient nodes share their key parts with the other nodes. This step further integrates the global model update, synchronization and distribution procedures integral to the FL process.

During local training, each robot node adds a transaction on the blockchain. This transaction includes the dataset's identity and size (p_n), as well as the hash of the key corresponding to the encrypted, updated local model. Following this, the selected robot node generates a Merkle tree that associates key parts with leaf nodes. It then distributes the Merkle tree's topology to all edge nodes and transmits the key part-leaf node association to the remaining robot nodes.

Each robot node then transmits the key parts according to the key part-leaf node association. All edge nodes exchange hashes until they can compute the Merkle root. At this point, they append a transaction containing the Merkle root to the blockchain, a process visualized in Figure 6.10. Once again, the edge nodes only exchange key parts with all other edge nodes when all the Merkle roots align.

The model aggregation step requires the delegated edge node to add an additional transaction. This transaction incorporates the global FL iteration count and the identifier of the edge node.

The smart contracts presented in Algorithm 6.3 ensure integrity in the aforementioned steps by mapping each action to a corresponding transaction in the blockchain. This transparency and traceability of operations are critical in the setup. During the process of implementing the key management, recipients do not require knowledge of the key part-leaf node association within the Merkle tree while the models remain encrypted during transmission; all the keys are disseminated in parts.

Furthermore, the pseudocode provided in this study has been formulated with scalability in mind. The first argument of functions, including `addTXN`, `addTXNandMine`, `InitShare`, and `Distribute`, represents the identifier of the node executing the function. The scalability and immutability are both ensured since once the smart contracts are deployed, they should not be further modified [123].

6.2.2 Computational Experiments

The Federated Learning (FL) experiment is conducted using the "MixedWM38" dataset [113] in a smart factory setting within the semiconductor industry. As the fundamentals for the production of integrated circuits (ICs) and a variety of electronic components, the operation of wafers encompasses deposition, etching, photolithography, doping, dicing, assembly, and testing. These processes can be executed by a heterogeneous Multi-Robot System (MRS). Various studies, including [113], [128], and [101], have investigated Machine Learning (ML) based wafer defect pattern recognition, useful for distinguishing batch variations of wafers when they are processed by an MRS.

For the sake of generality, a unified model that represents different operations on wafers is proposed, which is solely for demonstrating the proposed integrity and privacy-preserving FL framework. In this model, the robot manipulates only normal dies or approaches as close as possible to such. A wafer is represented as a matrix $\mathbf{A}$, where $a_{ij} = 0$ signifies an empty space; $a_{ij} = 1$ indicates the presence of a normal die; and $a_{ij} = 2$ represents a failed die. The task execution parameter for the robot is defined as a matrix $\mathbf{B}$, which shares dimensions with $\mathbf{A}$, and the execution of the task is defined as $\mathbf{C} = \mathbf{A} + \mathbf{B}$. Further a "paradigm" matrix $\mathbf{D}$ that describes the perfect task execution outcome is assumed. $\mathbf{D} := \mathbf{A} - 2(\mathbf{A} == 1)$, resulting from a logical operation that introduces non-linearity mathematically. The FL's goal is to minimize the Euclidean distance between $\mathbf{C}$ and $\mathbf{D}$. This is equivalent to calculating the L2 norm of the difference between the two matrices, resulting in a convex problem as expressed in Equation (6.10):

$$\ell(\mathbf{C}, \mathbf{D}) = |C - D|F = \sqrt{\sum i \sum_{j} (c_{ij} - d_{ij})^2} \tag{6.10}$$

The batch variation of wafers (the raw materials for the MRS) is introduced by distributing one of the 38 categories of the dataset to all the robots in a time-varying manner, which remains undisclosed to the robots. When a global iteration number is reached, the simulation switches to the next dataset for all robots, indicating the exhaustion of the previous wafer batch and the arrival of a new one. The robots adapt to such batch variation through FL and determine the task execution parameters $\mathbf{B}$ to minimize the loss between $\mathbf{C} = \mathbf{A} + \mathbf{B}$ and $\mathbf{D}$. Figure 6.11 shows two wafer examples ($\mathbf{A}$) from pattern ID C7 and C13, along with the corresponding task execution parameters $\mathbf{B}$, task execution outcome $\mathbf{C}$, and the paradigm task execution outcome $\mathbf{D}$. The wafers from the same pattern shall be homogeneous and the wafers from different patterns shall be heterogeneous which the robots need to adapt to.

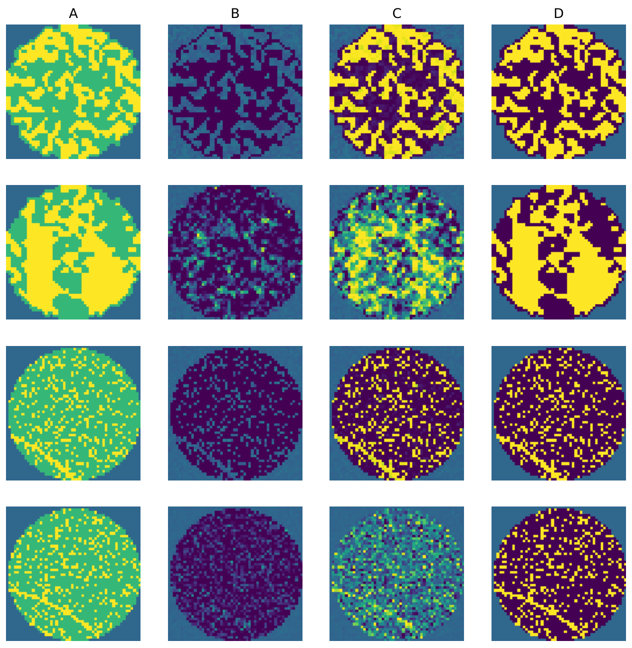


Figure 6.11 The wafer patterns as the experiment environment.

In the Federated Learning (FL) experiment, 17% of the dataset is allocated for validation to confirm the assumptions regarding the loss function. As the loss function is selected for proof-of-concept purposes, its absolute value does not carry physical significance. Both the training and validation losses are illustrated in Figure 6.12, with the vertical dotted line indicating the point of dataset switching in terms of FL global iterations.

Table 6.2 Parameters of smart contracts experiments.

Variable	Quantity
M	6
N	37
local training epochs	5
local batch size	32
learning rate	0.001

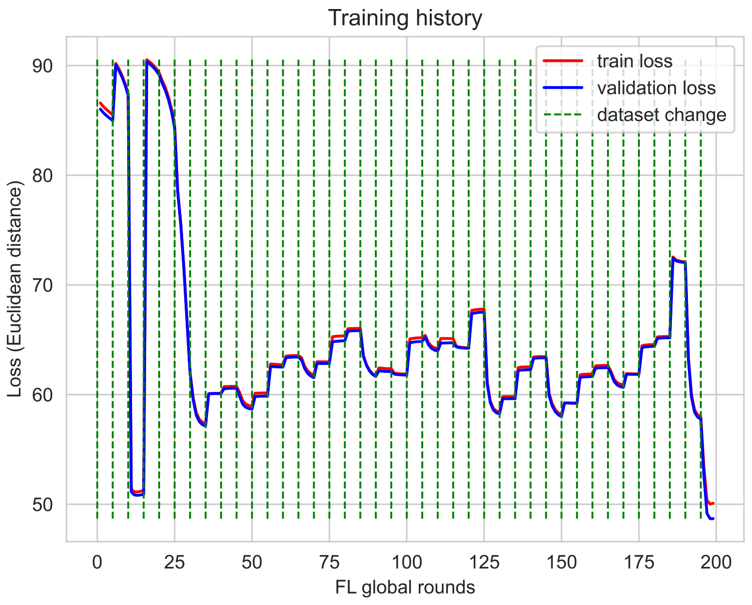
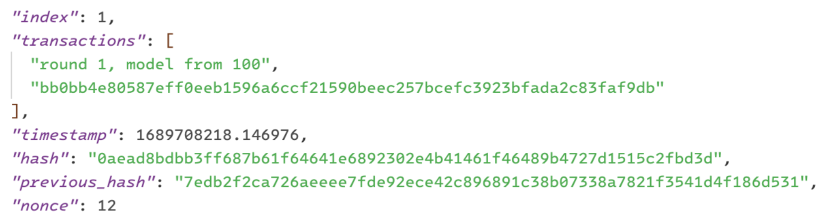


Figure 6.12 The training loss and validation loss of FL.

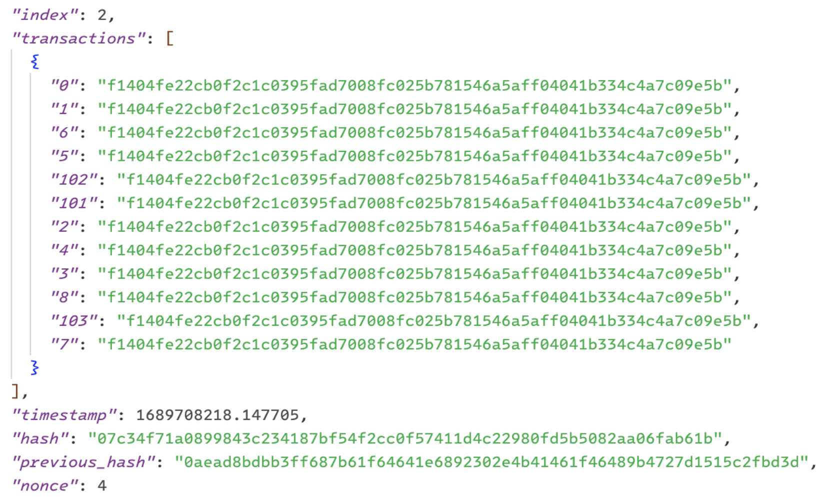
From the figure, it can be observed that both the training and validation losses decrease over the FL global iterations. Each time the dataset switches, there is an expected increase in loss, as the robots need time to adapt to the new batch of wafer patterns. The loss begins to decrease again once the robots have adjusted to the new batch. It should be noted that the optimal loss can differ since the method of modeling task execution $\mathbf{C} = \mathbf{A} + \mathbf{B}$ is not unique.

Figures 6.13 to 6.16 depict four distinct blocks in the blockchain. These were extracted from an experiment involving $M = 4$ edge nodes (ID 100, 101, 102, 103) and $N = 9$ robot nodes (ID 0, 1, ..., 8). The experiment settings have been simplified to these parameters rather than $M = 6$ and $N = 37$ purely for illustrative purposes.



```
"index": 1,
"transactions": [
  "round 1, model from 100",
  "bb0bb4e80587eff0eeb1596a6ccf21590beec257bcefc3923bfada2c83faf9db"
],
"timestamp": 1689708218.146976,
"hash": "0aead8bdbb3ff687b61f64641e6892302e4b41461f46489b4727d1515c2fbd3d",
"previous_hash": "7edb2f2ca726aeeee7fde92ece42c896891c38b07338a7821f3541d4f186d531",
"nonce": 12
```

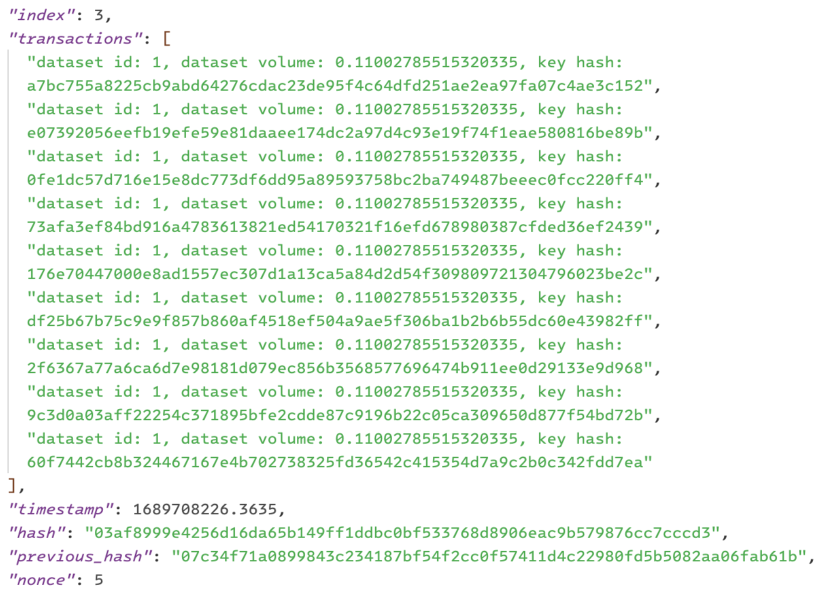
Figure 6.13 An block containing global model distribution transactions enforced by smart contracts.



```
"index": 2,
"transactions": [
  {
    "0": "f1404fe22cb0f2c1c0395fad7008fc025b781546a5aff04041b334c4a7c09e5b",
    "1": "f1404fe22cb0f2c1c0395fad7008fc025b781546a5aff04041b334c4a7c09e5b",
    "6": "f1404fe22cb0f2c1c0395fad7008fc025b781546a5aff04041b334c4a7c09e5b",
    "5": "f1404fe22cb0f2c1c0395fad7008fc025b781546a5aff04041b334c4a7c09e5b",
    "102": "f1404fe22cb0f2c1c0395fad7008fc025b781546a5aff04041b334c4a7c09e5b",
    "101": "f1404fe22cb0f2c1c0395fad7008fc025b781546a5aff04041b334c4a7c09e5b",
    "2": "f1404fe22cb0f2c1c0395fad7008fc025b781546a5aff04041b334c4a7c09e5b",
    "4": "f1404fe22cb0f2c1c0395fad7008fc025b781546a5aff04041b334c4a7c09e5b",
    "3": "f1404fe22cb0f2c1c0395fad7008fc025b781546a5aff04041b334c4a7c09e5b",
    "8": "f1404fe22cb0f2c1c0395fad7008fc025b781546a5aff04041b334c4a7c09e5b",
    "103": "f1404fe22cb0f2c1c0395fad7008fc025b781546a5aff04041b334c4a7c09e5b",
    "7": "f1404fe22cb0f2c1c0395fad7008fc025b781546a5aff04041b334c4a7c09e5b"
  }
],
"timestamp": 1689708218.147705,
"hash": "07c34f71a0899843c234187bf54f2cc0f57411d4c22980fd5b5082aa06fab61b",
"previous_hash": "0aead8bdbb3ff687b61f64641e6892302e4b41461f46489b4727d1515c2fbd3d",
"nonce": 4
```

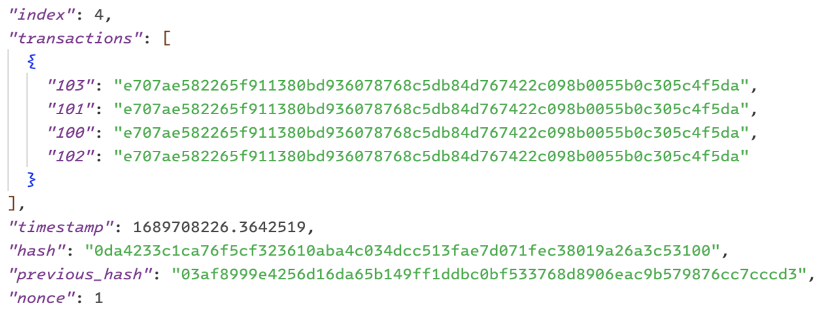
Figure 6.14 An block containing Merkle tree-based key management transactions enforced by smart contracts.

Figure 6.13 shows the delegation of global model distribution to edge node 1 (ID 100). This node adds a transaction to the blockchain that includes the FL iteration number, the executor node ID, and the hash of the key corresponding to the global model. Figure 6.14 shows transactions associated with the key management during the global model distribution process. All robot nodes and edge nodes, excluding the delegated edge node 1 (ID 100), add transactions to the blockchain. These transactions comprise the Merkle root calculated from the key parts, the Merkle tree topology, and the necessary hashes received from edge node 1 (ID 100). Only after all nodes verify all the Merkle roots, transactions containing the key parts are added to the blockchain. Figure 6.15 shows the local training transactions originating from the robots. These transactions include the dataset ID, dataset volume (p_n), and the hash of the key to the local model. Figure 6.16 shows the local model upload



```
"index": 3,
"transactions": [
  "dataset id: 1, dataset volume: 0.11002785515320335, key hash:
  a7bc755a8225cb9abd64276cdac23de95f4c64dfd251ae2ea97fa07c4ae3c152",
  "dataset id: 1, dataset volume: 0.11002785515320335, key hash:
  e07392056eefb19efe59e81daaee174dc2a97d4c93e19f74f1eae580816be89b",
  "dataset id: 1, dataset volume: 0.11002785515320335, key hash:
  0fe1dc57d716e15e8dc773df6dd95a89593758bc2ba749487beeec0fcc220ff4",
  "dataset id: 1, dataset volume: 0.11002785515320335, key hash:
  73afa3ef84bd916a4783613821ed54170321f16efd678980387cfded36ef2439",
  "dataset id: 1, dataset volume: 0.11002785515320335, key hash:
  176e70447000e8ad1557ec307d1a13ca5a84d2d54f3098097213047960 23be2c",
  "dataset id: 1, dataset volume: 0.11002785515320335, key hash:
  df25b67b75c9e9f857b860af4518ef504a9ae5f306ba1b2b6b55dc60e43982ff",
  "dataset id: 1, dataset volume: 0.11002785515320335, key hash:
  2f6367a77a6ca6d7e98181d079ec856b3568577696474b911ee0d29133e9d968",
  "dataset id: 1, dataset volume: 0.11002785515320335, key hash:
  9c3d0a03aff22254c371895bfe2cdde87c9196b22c05ca309650d877f54bd72b",
  "dataset id: 1, dataset volume: 0.11002785515320335, key hash:
  60f7442cb8b324467167e4b702738325fd36542c415354d7a9c2b0c342fdd7ea"
],
"timestamp": 1689708226.3635,
"hash": "03af8999e4256d16da65b149ff1ddbc0bf533768d8906eac9b579876cc7cccd3",
"previous_hash": "07c34f71a0899843c234187bf54f2cc0f57411d4c22980fd5b5082aa06fab61b",
"nonce": 5
```

Figure 6.15 An block containing local training transactions enforced by smart contracts.



```
"index": 4,
"transactions": [
  {
    "103": "e707ae582265f911380bd936078768c5db84d767422c098b0055b0c305c4f5da",
    "101": "e707ae582265f911380bd936078768c5db84d767422c098b0055b0c305c4f5da",
    "100": "e707ae582265f911380bd936078768c5db84d767422c098b0055b0c305c4f5da",
    "102": "e707ae582265f911380bd936078768c5db84d767422c098b0055b0c305c4f5da"
  }
],
"timestamp": 1689708226.3642519,
"hash": "0da4233c1ca76f5cf323610aba4c034dcc513fae7d071fec38019a26a3c53100",
"previous_hash": "03af8999e4256d16da65b149ff1ddbc0bf533768d8906eac9b579876cc7cccd3",
"nonce": 1
```

Figure 6.16 An block containing local model upload enforced by smart contracts.

transactions from the edge nodes, which include the Merkle roots they have calculated. Once the robot node delegated for local model uploading verifies all the Merkle roots, all edge nodes exchange their key parts for the full key.

All the transactions detailed above are immutable and accessible to all nodes in the blockchain network, ensuring the integrity of the FL process. Any malicious activities, including missing transactions, incorrect Merkle roots, and inconsistent dataset IDs, can be immediately detected by any node in the network, thereby preserving the integrity of FL.

Chapter 7: Summary and Future Technological Opportunities

Revisiting the research objectives outlined in Section 1.2, this study is anchored in the principles of smart factories and puts forward an MRS architecture tailored for flexible, on-demand, mass-customized production. This approach introduces new technical challenges related to intricate coordination and resilient operation. To address these challenges, the research proposes a decentralized architecture powered by cyber-physical AI agents, aimed at enhancing scalability.

Utilizing graph models, this research characterizes the heterogeneity of MRS, captures multiple dynamically adjusted production flows, and models operations within the smart factory environment. These graph models serve as domain models that enable AI agents to solve a range of problems. These include optimizing productivity, improving energy efficiency, coordinating transportation routes, detecting and mitigating cyber-physical errors, and recovering from localized failures. All of these tasks are accomplished through AI cognition, communication, learning, and collaboration.

Supported by comprehensive engineering problems, solution frameworks, and numerical results, this study substantiates the feasibility of the proposed MRS architecture as well as the efficacy of the AI agents developed.

Looking ahead, future research opportunities may involve integrating the proposed MRTA with predictive path coordination mechanisms for transportation robots, advancing predictive maintenance models that account for MRS reconfiguration time and energy expenditures, and extending the framework to multiple smart factories.

The proposed real-time MRTA offers dual benefits: task scheduling and task allocation, anchored in temporal-spatial MRS modeling. Yet, when viewed through the lens of AI planning, its application appears limited. The current real-time MRTA formulates plans based solely on immediate production demands, striving to optimize throughput. This approach, however, doesn't factor in variations in customer product demand. Tending to favor "easier" products for maximum throughput can misalign with customer preferences, leading to potential overproduction of less desired items. Consequently, a promising direction for future work would be to enhance the planning component of real-time MRTA. By calculating a series of dynamic task assignments, it can better mirror the diverse product quantities demanded by customers.

Moreover, this study operates under certain assumptions that may not hold in real-world scenarios. It presumes that reconfigurations are cost-free in terms of time, energy, and other resources. Such an assumption is overly optimistic and somewhat detached from practical realities. Similarly, the work downplays the temporal and energy costs associated with maintenance and calibration, treating them as negligible. Addressing these gaps, future technical endeavors should incorporate these costs. For instance, when reshaping the MRS, the AI agent could choose from a range of potential new physical topologies. These topologies, while being productivity and energy-efficient, should be structurally similar to the preceding configuration. Techniques like graph learning [121] and convolutional neural networks can be employed to concurrently evaluate productivity, energy efficiency, and reconfiguration costs. Such a strategy ensures that a majority of the production flows remain consistent, leading to fewer disruptions. Additionally, minimizing robot reconfiguration can mitigate the

Another potential area for technical exploration lies in the integration of production-MRS task execution and transportation-MRS task execution. The real-time MRTA introduced in this study allocates tasks to both MRS types, yet the subsequent sections delve into each MRS in isolation. The rationale behind this approach is the classic "divide and conquer" strategy. An evident distinction between the two systems is their differing time slot lengths.

While production robots can readily synchronize their time slots thanks to their multiplexing capability, transportation robots present a challenge. Both modeling and experimental results show variability in the execution time slots of transportation tasks, introducing the concept of task “difficulty”. Chapter 5 touches upon the notion of “fairness” in task assignments and path coordination. However, integrating this with production MRS introduces fresh complications. Specifically, task assignments for transportation robots need to factor in their instantaneous locations and the pickup locations of the production robots. Addressing this via a centralized MRTA presents computational challenges, evolving into an unscalable NP-hard problem.

A promising solution might involve fostering communication between production and transportation robots and empowering transportation robots with multiplexing abilities. This integration could be initiated by incorporating transportation-MRS into the broader framework of CPMRS.

References

[1] Shougi Suliman Abosuliman and Alaa Omran Almagrabi. Routing and scheduling of intelligent autonomous vehicles in industrial logistics systems. *Soft Computing*, 25:11975–11988, 2021.

[2] Mahbuba Afrin, Jiong Jin, Ashfaqur Rahman, Yu-Chu Tian, and Ambarish Kulkarni. Multi-objective resource allocation for edge cloud based robotic workflow in smart factory. *Future Generation Computer Systems*, 97:119–130, 2019.

[3] Imran Ahmed, Gwanggil Jeon, and Francesco Piccialli. From artificial intelligence to explainable artificial intelligence in industry 4.0: A survey on what, how, and where. *IEEE Transactions on Industrial Informatics*, 18(8):5031–5042, 2022.

[4] Ali Al Zoobi, David Coudert, and Nicolas Nisse. Space and time trade-off for the k shortest simple paths problem. In *SEA 2020-18th International Symposium on Experimental Algorithms*, volume 160, page 13. Schloss Dagstuhl–Leibniz-Zentrum f {\" u} r Informatik, 2020.

[5] Anton Averyanov, Shohin Aheleroff, Jan Polzer, and Xun Xu. Digitising a machine tool for smart factories. *Machines*, 10(11):1093, 2022.

[6] Abdullah Ayub Khan, Asif Ali Laghari, Zaffar Ahmed Shaikh, Zdzislawa Dacko-Pikiewicz, and Sebastian Kot. Internet of things (iot) security with blockchain technology: A state-of-the-art review. *IEEE Access*, 10:122679–122695, 2022.

[7] Adithya Balachandran, Anil Lal S, and Pramod Sreedharan. Autonomous navigation of an amr using deep reinforcement learning in a warehouse environment. In *2022 IEEE 2nd Mysore Sub Section International Conference (MysuruCon)*, pages 1–5. IEEE, 2022.

[8] David Barber. *Bayesian reasoning and machine learning*. Cambridge University Press, 2012.

[9] Luca Barbieri, Mattia Brambilla, Andrea Trabattoni, Stefano Mervic, and Monica Nicoli. Uwb localization in a smart factory: Augmentation methods and experimental assessment. *IEEE Transactions on Instrumentation and Measurement*, 70:1–18, 2021.

[10] Alessia Benevento, María Santos, Giuseppe Notarstefano, Kamran Paynabar, Matthieu Bloch, and Magnus Egerstedt. Multi-robot coordination for estimation and coverage of unknown spatial fields. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7740–7746. IEEE, 2020.

[11] Béla Bollobás. *Modern graph theory*, volume 184. Springer Science & Business Media, 1998.

[12] Alain Bretto. *Hypergraph theory*. Springer, 2013.

[13] Kai Cai. Warehouse automation by logistic robotic networks: a cyber-physical control approach. *Frontiers of Information Technology & Electronic Engineering*, 21(5):693–704, 2020.

[14] Zilong Cao, Pan Zhou, Ruixuan Li, Siqi Huang, and Dapeng Wu. Multiagent deep reinforcement learning for joint multichannel access and task offloading of mobile-edge computing in industry 4.0. *IEEE Internet of Things Journal*, 7(7):6201–6213, 2020.

[15] Matteo Capucci, Bruno Gavranović, Jules Hedges, and Eigil Fjeldgren Rischel. Towards foundations of categorical cybernetics. *arXiv preprint arXiv:2105.06332*, 2021.

[16] Guido Carnevale, Andrea Camisa, and Giuseppe Notarstefano. Distributed online aggregative optimization for dynamic multi-robot coordination. *IEEE Transactions on Automatic Control*, pages 1–8, 2022.

[17] George Casella and Roger L Berger. *Statistical inference*. Cengage Learning, 2021.

[18] Amit Chavda, Jason Dsouza, Sumeet Badgujar, and Ankit Damani. Multi-stage cnn architecture for face mask detection. In *2021 6th International Conference for Convergence in Technology (I2CT)*, pages 1–8. IEEE, 2021.

[19] K. C. Chen, S. C. Lin, J. H. Hsiao, C. H. Liu, A. F. Molisch, and G. P. Fettweis. Wireless networked multirobot systems in smart factories. *Proceedings of the IEEE*, 109(4):468–494, 2021.

[20] Kwang-Cheng Chen. *Artificial Intelligence in Wireless Robotics*. River, 2020.

[21] Kwang-Cheng Chen, Yingze Wang, Zixiang Nie, and Qimei Cui. Toward holistic integration of computing and wireless networking. In *IFIP International Internet of Things Conference*, pages 219–234. Springer, 2019.

[22] Kwang-Cheng Chen, Tao Zhang, Richard D. Gitlin, and Gerhard Fettweis. Ultra-low latency mobile networking. *IEEE Network*, 33(2):181–187, 2019.

[23] Yujing Chen, Fenghua Zhao, and Yunjiang Lou. Interactive model predictive control for robot navigation in dense crowds. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 52(4):2289–2301, 2022.

[24] Zhe Chen, Javier Alonso-Mora, Xiaoshan Bai, Daniel D. Harabor, and Peter J. Stuckey. Integrated task assignment and path planning for capacitated multi-agent pickup and delivery. *IEEE Robotics and Automation Letters*, 6(3):5816–5823, 2021.

[25] Zhenyi Chen, Kwang-Cheng Chen, Chen Dong, and Zixiang Nie. 6g mobile communications for multi-robot smart factory. *Journal of ICT Standardization*, 9(1):371–404, 2021.

[26] Shushman Choudhury, Jayesh Gupta, Mykel Kochenderfer, Dorsa Sadigh, and Jeannette Bohg. Dynamic Multi-Robot Task Allocation under Uncertainty and Temporal Constraints. In *Proceedings of Robotics: Science and Systems*, Corvalis, Oregon, USA, July 2020.

[27] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[28] Yu Du, Junqing Li, Chengdong Li, and Peiyong Duan. A reinforcement learning approach for flexible job shop scheduling problem with crane transportation and setup times. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–15, 2022.

[29] Ayan Dutta, Emily Czarnecki, Vladimir Ufimtsev, and Asai Asaithambi. Correlation clustering-based multi-robot task allocation: a tale of two graphs. *ACM SIGAPP Applied Computing Review*, 19(4):5–16, 2020.

[30] Y. Fang, C. Peng, P. Lou, Z. Zhou, J. Hu, and J. Yan. Digital-twin-based job shop scheduling toward smart manufacturing. *IEEE Transactions on Industrial Informatics*, 15(12):6425–6435, 2019.

[31] William Fedus, Prajit Ramachandran, Rishabh Agarwal, Yoshua Bengio, Hugo Larochelle, Mark Rowland, and Will Dabney. Revisiting fundamentals of experience replay. In *International Conference on Machine Learning*, pages 3061–3071. PMLR, 2020.

[32] Wenliang Gao, Jiarong Lin, Fu Zhang, and Shaojie Shen. A screen-based method for automated camera intrinsic calibration on production lines. In *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pages 392–398. IEEE, 2019.

[33] Zhen Gao, Tom Wanyama, Ishwar Singh, Anoop Gadhrri, and Reiner Schmidt. From industry 4.0 to robotics 4.0-a conceptual framework for collaborative and intelligent robotic systems. *Procedia manufacturing*, 46:591–599, 2020.

[34] E. Genender, C. L. Holloway, K. A. Remley, J. M. Ladbury, G. Koepke, and H. Garbe. Simulating the multipath channel with a reverberation chamber: Application to bit error rate measurements. *IEEE Transactions on Electromagnetic Compatibility*, 52(4):766–777, 2010.

[35] Brian P Gerkey and Maja J Matarić. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International journal of robotics research*, 23(9):939–954, 2004.

[36] Payam Ghassemi and Souma Chowdhury. Decentralized task allocation in multi-robot systems via bipartite graph matching augmented with fuzzy clustering. In *International design engineering technical conferences and computers and information in engineering conference*, volume 51753, page V02AT03A014. American Society of Mechanical Engineers, 2018.

[37] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[38] Mingrui Gu, Liang Luo, and Pengzhong Li. Study and development of intelligent kpi management system for discrete manufacturing enterprises in industry 4.0. In *2023 4th International Conference on Mechatronics Technology and Intelligent Manufacturing (ICMTIM)*, pages 19–25. IEEE, 2023.

[39] Zhengang Guo, Yingfeng Zhang, Xibin Zhao, and Xiaoyu Song. Cps-based self-adaptive collaborative control for smart production-logistics systems. *IEEE Transactions on Cybernetics*, 51(1):188–198, 2021.

[40] Ali Hatamizadeh, Hongxu Yin, Pavlo Molchanov, Andriy Myronenko, Wenqi Li, Prerna Dogra, Andrew Feng, Mona G Flores, Jan Kautz, Daguang Xu, and Holger R. Roth. Do gradient inversion attacks make federated learning unsafe? *IEEE Transactions on Medical Imaging*, 42(7):2044–2056, 2023.

[41] Christian Henkel, Jannik Abbenseth, and Marc Toussaint. An optimal algorithm to solve the combined task allocation and path finding problem. *arXiv preprint arXiv:1907.10360*, 2019.

[42] Chen Hou and Qianchuan Zhao. Optimal control of wireless powered edge computing system for balance between computation rate and energy harvested. *IEEE Transactions on Automation Science and Engineering*, pages 1–17, 2022.

[43] J. H. Hsiao and K. C. Chen. Network analysis of collaborative cyber-physical multi-agent smart manufacturing systems : Invited paper. In *2019 IEEE/CIC International Conference on Communications in China (ICCC)*, pages 219–224. IEEE, 2019.

[44] Yin Huang, Yi Zhang, and Hong Xiao. Multi-robot system task allocation mechanism for smart factory. In *2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC)*, pages 587–591. IEEE, 2019.

[45] Jeong-Yeon Hwang, Jiyoun Seo, and Chang-Hyeon Ji. Electromagnetic omnidirectional scanning micromirror with multi jet fusion printed structures for smart factory applications. *Additive Manufacturing*, 55:102868, 2022.

[46] Senthil Kumar Jagatheesaperumal, Mohamed Rahouti, Kashif Ahmad, Ala Al-Fuqaha, and Mohsen Guizani. The duo of artificial intelligence and big data for industry 4.0: Applications, techniques, challenges, and future research directions. *IEEE Internet of Things Journal*, 9(15):12861–12885, 2022.

[47] Malhar S. Jere, Tyler Farnan, and Farinaz Koushanfar. A taxonomy of attacks on federated learning. *IEEE Security & Privacy*, 19(2):20–28, 2021.

[48] Yupeng Jiang, Yong Li, Yipeng Zhou, and Xi Zheng. Sybil attacks and defense on differential privacy based federated learning. In *2021 IEEE 20th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 355–362. IEEE, 2021.

[49] Zengqiang Jiang, Yang Jin, Mingcheng E, and Qi Li. Distributed dynamic scheduling for cyber-physical production systems based on a multi-agent system. *IEEE Access*, 6:1855–1869, 2018.

[50] Jiao Jiao, Shuanglong Kan, Shang-Wei Lin, David Sanan, Yang Liu, and Jun Sun. Semantic understanding of smart contracts: Executable operational semantics of solidity. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 1695–1712. IEEE, 2020.

[51] Joran Jongerling, Sacha Epskamp, and Donald R Williams. Bayesian uncertainty estimation for gaussian graphical models and centrality indices. *Multivariate Behavioral Research*, 58(2):311–339, 2023.

[52] Tahera Kalsoom, Naeem Ramzan, Shehzad Ahmed, and Masood Ur-Rehman. Advances in sensor technologies in the era of smart factory and industry 4.0. *Sensors*, 20(23):6783, 2020.

[53] Abid Khan, Furqan Shahid, Carsten Maple, Awais Ahmad, and Gwanggil Jeon. Toward smart manufacturing using spiral digital twin framework and twinchain. *IEEE Transactions on Industrial Informatics*, 18(2):1359–1366, 2022.

[54] Reza Kia, Fahime Khaksar-Haghani, Nikbakhsh Javadian, and Reza Tavakkoli-Moghaddam. Solving a multi-floor layout design model of a dynamic cellular manufacturing system by an efficient genetic algorithm. *Journal of Manufacturing Systems*, 33(1):218–232, 2014.

[55] G Ayorkor Korsah, Anthony Stentz, and M Bernardine Dias. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, 32(12):1495–1512, 2013.

[56] Naveen Kumar and Jyoti Kumar. Efficiency 4.0 for industry 4.0. *Human Technology*, 15(1):55, 2019.

[57] Bahareh Lashkari and Petr Musilek. A comprehensive review of blockchain consensus mechanisms. *IEEE Access*, 9:43620–43652, 2021.

[58] Yann LeCun. A path towards autonomous machine intelligence version 0.9. 2, 2022-06-27. *Open Review*, 62, 2022.

[59] P.M. Lee. *Bayesian Statistics: An Introduction*. Wiley, 2012.

[60] Jiewu Leng, Xiaofeng Zhu, Zhiqiang Huang, Kailin Xu, Zhihong Liu, Qiang Liu, and Xin Chen. Manuchain ii: Blockchained smart contract system as the digital twin of decentralized autonomous manufacturing toward resilience in industry 5.0. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 53(8):4715–4728, 2023.

[61] Tianxu Li, Kun Zhu, Nguyen Cong Luong, Dusit Niyato, Qihui Wu, Yang Zhang, and Bing Chen. Applications of multi-agent reinforcement learning in future internet: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 24(2):1240–1279, 2022.

[62] Xiaomin Li, Di Li, Jiafu Wan, Athanasios V Vasilakos, Chin-Feng Lai, and Shiyong Wang. A review of industrial wireless networks in the context of industry 4.0. *Wireless networks*, 23(1):23–41, 2017.

[63] Zengpeng Li, Vishal Sharma, and Saraju P. Mohanty. Preserving data privacy via federated learning: Challenges and solutions. *IEEE Consumer Electronics Magazine*, 9(3):8–16, 2020.

[64] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *International Conference on Learning Representations (ICLR)*, 2016.

[65] Igor Linkov and Benjamin D Trump. *The science and practice of resilience*. Springer, 2019.

[66] Xiaopeng Liu, Huiyu Zhang, Jun Lin, Xuanrui Chen, Qingxin Chen, and Ning Mao. A queuing network model for solving facility layout problem in multifloor flow shop. *IEEE Access*, 10:61326–61341, 2022.

[67] Yonggang Liu, Bobo Zhou, Xiao Wang, Liang Li, Shuo Cheng, Zheng Chen, Guang Li, and Lu Zhang. Dynamic lane-changing trajectory planning for autonomous vehicles based on discrete global trajectory. *IEEE Transactions on Intelligent Transportation Systems*, 23(7):8513–8527, 2022.

[68] Zhixiang Liu, Youmin Zhang, Chi Yuan, and Jun Luo. Adaptive path following control of unmanned surface vehicles considering environmental disturbances and system constraints. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 51(1):339–353, 2021.

[69] Hang Ma, Jiaoyang Li, T.K. Satish Kumar, and Sven Koenig. Lifelong multi-agent path finding for online pickup and delivery tasks. In *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '17, page 837–845, Richland, SC, 2017. International Foundation for Autonomous Agents and Multiagent Systems.

[70] Abderraouf Maoudj, Brahim Bouzouia, Abdelfetah Hentout, Ahmed Kouider, and Redouane Toumi. Distributed multi-agent scheduling and control system for robotic flexible assembly cells. *Journal of Intelligent Manufacturing*, 30(4):1629–1644, 2019.

[71] Vincenzo Matta, Virginia Bordignon, Augusto Santos, and Ali H. Sayed. Interplay between topology and social learning over weak graphs. *IEEE Open Journal of Signal Processing*, 1:99–119, 2020.

[72] Vincenzo Matta, Augusto Santos, and Ali H. Sayed. Graph learning with partial observations: Role of degree concentration. In *2019 IEEE International Symposium on Information Theory (ISIT)*, pages 1312–1316. IEEE, 2019.

[73] Douglas C Montgomery, Cheryl L Jennings, and Murat Kulahci. *Introduction to time series analysis and forecasting.* John Wiley & Sons, 2015.

[74] Anuj Nandanwar, Vibhu Kumar Tripathi, and Laxmidhar Behera. Fault-tolerant control for multi-robotics system using variable gain super twisting sliding mode control in cyber-physical framework. In *2021 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 1147–1152. IEEE, 2021.

[75] Zixiang Nie and Kwang-Cheng Chen. Distributed coordination by social learning in the multi-robot systems of a smart factory. In *2021 IEEE Global Communications Conference (GLOBECOM)*, pages 01–06. IEEE, 2021.

[76] Zixiang Nie and Kwang-Cheng Chen. Hypergraphical real-time multirobot task allocation in a smart factory. *IEEE Transactions on Industrial Informatics*, 18(9):6047–6056, 2022.

[77] Felipe Orellana and Romina Torres. From legacy-based factories to smart factories level 2 according to the industry 4.0. *International Journal of Computer Integrated Manufacturing*, 32(4-5):441–451, 2019.

[78] Michael Otte, Michael J Kuhlman, and Donald Sofge. Auctions for multi-robot task allocation in communication limited environments. *Autonomous Robots*, 44(3):547–584, 2020.

[79] Iván Palomares, Luis Martínez, and Francisco Herrera. A consensus model to detect and manage noncooperative behaviors in large-scale group decision making. *IEEE Transactions on Fuzzy Systems*, 22(3):516–530, 2014.

[80] Martin Pech, Jaroslav Vrchota, and Jiří Bednář. Predictive maintenance and intelligent sensors in smart factory. *Sensors*, 21(4):1470, 2021.

[81] Muleilan Pei, Hao An, Bo Liu, and Changhong Wang. An improved dyna-q algorithm for mobile robot path planning in unknown dynamic environment. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 52(7):4415–4425, 2022.

[82] Ana Petrovska, Sergio Quijano, Ilias Gerostathopoulos, and Alexander Pretschner. Knowledge aggregation with subjective logic in multi-agent self-adaptive cyber-physical systems. In *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, pages 149–155, 2020.

[83] Jeremy Pitt. *Self-organising multi-agent systems: Algorithmic foundations of cyber-anarcho-socialism*. World Scientific, 2021.

[84] Rafael Figueiredo Prudencio, Marcos R. O. A. Maximo, and Esther Luna Colombini. A survey on offline reinforcement learning: Taxonomy, review, and open problems. *IEEE Transactions on Neural Networks and Learning Systems*, pages 1–0, 2023.

[85] Cheng Qian, Yingfeng Zhang, Chen Jiang, Shenle Pan, and Yiming Rong. A real-time data-driven collaborative mechanism in fixed-position assembly systems for smart manufacturing. *Robotics and Computer-Integrated Manufacturing*, 61:101841, 2020.

[86] Guixiu Qiao. Advanced sensor and target development to support robot accuracy degradation assessment. In *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pages 54–59. IEEE, 2019.

[87] Gholamreza Ramezan and Cyril Leung. Analysis of proof-of-work-based blockchains under an adaptive double-spend attack. *IEEE Transactions on Industrial Informatics*, 16(11):7035–7045, 2020.

[88] L. Romeo, A. Petitti, R. Marani, and A. Milella. Internet of robotic things in industry 4.0: Applications, issues and challenges. In *2020 7th International Conference on Control, Decision and Information Technologies (CoDIT)*, volume 1, pages 177–182. IEEE, 2020.

[89] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: a modern approach*. Pearson, 4 edition, 2020.

[90] Mutaz Ryalat, Hisham ElMoaqet, and Marwa AlFaouri. Design of a smart factory based on cyber-physical systems and internet of things towards industry 4.0. *Applied Sciences*, 13(4):2156, 2023.

[91] Kiran V Sagar and J Jerald. Real-time automated guided vehicles scheduling with markov decision process and double q-learning algorithm. *Materials Today: Proceedings*, 64:279–284, 2022.

[92] Doruk Sahinel, Simon Rommel, and Idelfonso Tafur Monroy. Resource allocation with vickrey-dutch auctioning game for c-ran fronthaul. In *2022 IEEE Future Networks World Forum (FNWF)*, pages 597–601. IEEE, 2022.

[93] Hiroyuki Sawada, Yoshihiro Nakabo, Yoshiyuki Furukawa, Noriaki Ando, Takashi Okuma, Hitoshi Komoto, and Keijiro Masui. Digital tools integration and human resources development for smart factories. *International Journal of Automation Technology*, 16(3):250–260, 2022.

[94] Ali H Sayed. Adaptation, learning, and optimization over networks. *Foundations and Trends in Machine Learning*, 7(ARTICLE):311–801, 2014.

[95] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *International Conference on Learning Representations (ICLR)*, 2016.

[96] Zhan Shi, Yongping Xie, Wei Xue, Yong Chen, Liuliu Fu, and Xiaobo Xu. Smart factory in industry 4.0. *Systems Research and Behavioral Science*, 37(4):607–617, 2020.

[97] Yoav Shoham and Kevin Leyton-Brown. *Multiagent systems: Algorithmic, game-theoretic, and logical foundations.* Cambridge University Press, 2008.

[98] Joakim Skarding, Bogdan Gabrys, and Katarzyna Musial. Foundations and modeling of dynamic networks using dynamic graph neural networks: A survey. *IEEE Access*, 9:79143–79168, 2021.

[99] Mohammad Divband Soorati and Heiko Hamann. Robot self-assembly as adaptive growth process: Collective selection of seed position and self-organizing tree-structures. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5745–5750. IEEE, 2016.

[100] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction.* MIT press, 2018.

[101] Terence Sweeney, Sonya Coleman, and Dermot Kerr. Deep learning for semiconductor defect classification. In *2022 IEEE 20th International Conference on Industrial Informatics (INDIN)*, pages 572–577. IEEE, 2022.

[102] Aaron Hao Tan and Goldie Nejat. Enhancing robot task completion through environment and task inference: A survey from the mobile robot perspective. *Journal of Intelligent & Robotic Systems*, 106(4):73, 2022.

[103] Gang Tang, Congqiang Tang, Christophe Claramunt, Xiong Hu, and Peipei Zhou. Geometric a-star algorithm: An improved a-star algorithm for agv path planning in a port environment. *IEEE Access*, 9:59196–59210, 2021.

[104] Cristian Tatino, Nikolaos Pappas, and Di Yuan. Multi-robot association-path planning in millimeter-wave industrial scenarios. *IEEE Networking Letters*, 2(4):190–194, 2020.

[105] Veniamin Tereshchuk, John Stewart, Nikolay Bykov, Samuel Pedigo, Santosh Devasia, and Ashis G. Banerjee. An efficient scheduling algorithm for multi-robot task allocation in assembling aircraft structures. *IEEE Robotics and Automation Letters*, 4(4):3844–3851, 2019.

[106] H. Touzani, H. Hadj-Abdelkader, N. Séguy, and S. Bouchafa. Multi-robot task sequencing automatic path planning for cycle time optimization: Application for car production line. *IEEE Robotics and Automation Letters*, 6(2):1335–1342, 2021.

[107] Kishor S Trivedi and Andrea Bobbio. *Reliability and availability engineering: modeling, analysis, and applications*. Cambridge University Press, 2017.

[108] Edward Tunstel, Manuel J. Cobo, Enrique Herrera-Viedma, Imre J. Rudas, Dimitar Filev, Ljiljana Trajkovic, C. L. Philip Chen, Witold Pedrycz, Michael H. Smith, and Robert Kozma. Systems science and engineering research in the context of systems, man, and cybernetics: Recollection, trends, and future directions. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 51(1):5–21, 2021.

[109] Natesha B V and Ram Mohana Reddy Guddeti. Fog-based intelligent machine malfunction monitoring system for industry 4.0. *IEEE Transactions on Industrial Informatics*, 17(12):7923–7932, 2021.

[110] Alberto Villalonga, Gerardo Beruvides, Fernando Castaño, and Rodolfo E. Haber. Cloud-based industrial cyber–physical system for data-driven reasoning: A review and use case on an industry 4.0 pilot line. *IEEE Transactions on Industrial Informatics*, 16(9):5975–5984, 2020.

[111] J. Wan, X. Li, H. N. Dai, A. Kusiak, M. Martínez-García, and D. Li. Artificial-intelligence-driven customized manufacturing factory: Key technologies, applications, and challenges. *Proceedings of the IEEE*, 109(4):377–398, 2021.

[112] Jin Wang, Yingfeng Zhang, Yang Liu, and Naiqi Wu. Multiagent and bargaining-game-based real-time scheduling for internet of things-enabled flexible job shop. *IEEE Internet of Things Journal*, 6(2):2518–2531, 2019.

[113] Junliang Wang, Chuqiao Xu, Zhengliang Yang, Jie Zhang, and Xiaoou Li. Deformable convolutional networks for efficient mixed-type wafer defect pattern recognition. *IEEE Transactions on Semiconductor Manufacturing*, 33(4):587–596, 2020.

[114] Shiyong Wang, Jiafu Wan, Di Li, and Chunhua Zhang. Implementing smart factory of industrie 4.0: an outlook. *International journal of distributed sensor networks*, 12(1):3159805, 2016.

[115] Taotao Wang, Chonghe Zhao, Qing Yang, Shengli Zhang, and Soung Chang Liew. Ethna: Analyzing the underlying peer-to-peer network of ethereum blockchain. *IEEE Transactions on Network Science and Engineering*, 8(3):2131–2146, 2021.

[116] Wenbo Wang, Yingfeng Zhang, Jinan Gu, and Jin Wang. A proactive manufacturing resources assignment method based on production performance prediction for the smart factory. *IEEE Transactions on Industrial Informatics*, 18(1):46–55, 2022.

[117] Susila Windarta, Suryadi Suryadi, Kalamullah Ramli, Bernardi Pranggono, and Teddy Surya Gunawan. Lightweight cryptographic hash functions: Design trends, comparative study, and future directions. *IEEE Access*, 10:82272–82294, 2022.

[118] David D Woods. *Resilience engineering: concepts and precepts*. Crc Press, 2017.

[119] Yulei Wu, Hong-Ning Dai, Haozhe Wang, Zehui Xiong, and Song Guo. A survey of intelligent network slicing management for industrial iot: Integrated approaches for smart transportation, smart energy, and smart factory. *IEEE Communications Surveys & Tutorials*, 24(2):1175–1211, 2022.

[120] Dan Xia, Chun Jiang, Jiafu Wan, Jiong Jin, Victor CM Leung, and Miguel Martínez-García. Heterogeneous network access and fusion in smart factory: A survey. *ACM Computing Surveys*, 55(6):1–31, 2022.

[121] Feng Xia, Ke Sun, Shuo Yu, Abdul Aziz, Liangtian Wan, Shirui Pan, and Huan Liu. Graph learning: A survey. *IEEE Transactions on Artificial Intelligence*, 2(2):109–127, 2021.

[122] Wanqing Xia, Joshua Goh, Carlos Aguilera Cortes, Yuqian Lu, and Xun Xu. Decentralized coordination of autonomous agvs for flexible factory automation in the context of industry 4.0. In *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, pages 488–493. IEEE, 2020.

[123] Dibo Xian and Xuetao Wei. Icoe: A lightweight group-consensus based off-chain execution model for smart contract based industrial applications. *IEEE Transactions on Industrial Informatics*, pages 1–12, 2023.

[124] Yong Xiao, Dusit Niyato, Kwang-Cheng Chen, and Zhu Han. Enhance device-to-device communication with social awareness: a belief-based stable marriage game framework. *IEEE Wireless Communications*, 23(4):36–44, 2016.

[125] Tomoki Yamauchi, Yuki Miyashita, and Toshiharu Sugawara. Standby-based deadlock avoidance method for multi-agent pickup and delivery tasks. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '22, page 1427–1435, Richland, SC, 2022. International Foundation for Autonomous Agents and Multiagent Systems.

[126] Shengluo Yang and Zhigang Xu. Intelligent scheduling for permutation flow shop with dynamic job arrival via deep reinforcement learning. In *2021 IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, volume 5, pages 2672–2677. IEEE, 2021.

[127] Honghan Ye, Xi Wang, and Kaibo Liu. Adaptive preventive maintenance for flow shop scheduling with resumable processing. *IEEE Transactions on Automation Science and Engineering*, 18(1):106–113, 2021.

[128] Jianbo Yu and Jiatong Liu. Multiple granularities generative adversarial network for recognition of wafer map defects. *IEEE Transactions on Industrial Informatics*, 18(3):1674–1683, 2022.

[129] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. Network representation learning: A survey. *IEEE transactions on Big Data*, 6(1):3–28, 2018.

[130] Feng Zhang, Min Liu, and Weiming Shen. Operation modes of smart factory for high-end equipment manufacturing in the internet and big data era. In *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 152–157. IEEE, 2017.

[131] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of reinforcement learning and control*, pages 321–384, 2021.

[132] Yuxian Zhang, Tiefeng Sheng, Suhong Zhong, and Mengting Zhou. Research on management optimization of intelligent discrete production system of just-in-time production and assembling in time. In *2022 International Conference on Intelligent Manufacturing, Advanced Sensing and Big Data (IMASBD)*, pages 7–12. IEEE, 2022.

[133] Bin Zhao, Kai Fan, Kan Yang, Zilong Wang, Hui Li, and Yintang Yang. Anonymous and privacy-preserving federated learning with industrial big data. *IEEE Transactions on Industrial Informatics*, 17(9):6314–6323, 2021.

[134] Jing Zhao, Wenfeng Li, Chuan Hu, Ge Guo, Zhengchao Xie, and Pak Kin Wong. Robust gain-scheduling path following control of autonomous vehicles considering stochastic network-induced delay. *IEEE Transactions on Intelligent Transportation Systems*, 23(12):23324–23333, 2022.

[135] Liang Zhao and Kwang-Cheng Chen. The game theoretic consensus in a networked multi-agent system. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–7. IEEE, 2018.

[136] Pai Zheng, Liqiao Xia, Chengxi Li, Xinyu Li, and Bufan Liu. Towards self-x cognitive manufacturing network: An industrial knowledge graph-based multi-agent reinforcement learning approach. *Journal of Manufacturing Systems*, 61:16–26, 2021.

About the Author

Zixiang (Sean) Nie received a B.E. degree in Electronic Information Engineering at Beijing University of Technology, Beijing, China in 2016 and an M.S. degree in Electrical and Computer Engineering at the University of Florida in 2019. Since 2019, he has been a PhD student and research assistant in Electrical Engineering at the University of South Florida. His research involves autonomous AI agent system solutions to wireless networked multi-agent systems in the context of smart factories and his research interests include domain-specific artificial intelligence, autonomous machine intelligence, multi-agent systems, and multi-agent reinforcement learning. He was a co-op of the Nokia CNS Tech Advanced Technology Group from May 2022 to August 2023. He is awarded a Dissertation Completion Fellowship by the Office of Graduate Studies at the University of South Florida for fall 2023.