Università degli Studi di Genova
scuola politecnica

DIME

Dipartimento di Ingegneria Meccanica,
Energetica, Gestionale e dei Trasporti

tesi di laurea

in

ingegneria gestionale

Distributed coordination mechanisms of AGVs in Flexible Manufacturing Systems.

Relatori:

Prof.ssa Cecilia Caterina Pasquale

Prof.ssa Silvia Siri

Prof. Enrico Zero

Correlatori:

Dott. Alessandro Bozzi

Prof. Jose-Fernando Jimenez

Allievo:

Lorenzo Rittore

Marzo 2026

Meccanismi di coordinamento distribuito per AGV nei Sistemi di Produzione Flessibili.

Sommario

La tesi propone un framework per l'implementazione e il confronto di meccanismi di coordinamento online per AGV in sistemi manufatturieri flessibili. Si osserva come il coordinamento di mezzi autonomi per la movimentazione in ambito manifatturiero comporti l'insorgenza di fenomeni poco considerati come la congestione. Viene quindi formalizzata una architettura per un digital twin basato su agenti per il controllo distribuito di AGV e l'implementazione di quattro meccanismi di coordinamento: *Congestion avoidance*, *Delay triggered reassignment*, *Reservation board* e *Virtual platooning*. Viene quindi proposto un indice a somma pesata che tenga conto, a seconda delle preferenze del decisore o del contesto, sia del tempo di produzione che di congestione media nella movimentazione. Il framework viene quindi testato tramite simulazione nella cella di produzione AIP-PRIMECA Valenciennes riprodotta tramite NetLogo e controllata tramite Python. Vengono testati diversi scenari composti da diversi prodotti, preferenze del decisore e sottoposti a perturbazioni. Vengono infine confrontati i meccanismi di coordinamento utilizzando il metodo di valutazione proposto.

Distributed coordination mechanisms of AGVs in Flexible Manufacturing Systems.

Abstract

The thesis proposes a framework for implementing and comparing online coordination mechanisms for AGVs in Flexible Manufacturing Systems. It can be observed how coordinating autonomous vehicles for handling in manufacturing environments leads to the emergence of rarely considered phenomena such as congestion. An architecture is then formalized for an agent-based digital twin for the distributed control of AGVs and the implementation of four coordination mechanisms: *Congestion avoidance*, *Delay triggered reassignment*, *Reservation board* and *Virtual platooning*. A weighted-sum index is then proposed that takes into account, depending on the decision maker's preferences or the context, both production time and average handling congestion. The framework is then tested through simulation in the AIP-PRIMECA Valenciennes production cell, reproduced using NetLogo and controlled using Python. Different scenarios consisting of different products, decision maker preferences, and perturbations are tested. Finally, the coordination mechanisms are evaluated using the proposed evaluation method.

Contents

Chapter 1

Introduction

Industry 4.0 is characterized by the digitalization and interconnection of components in production processes. A further step beyond simple automation in manufacturing environments is therefore creating an ecosystem in which machines, products, sensors, and people communicate constantly in real time. In this context, more and more smart factories are competing to make production flexible, modular, and capable of self-organization.

Interconnected automation makes production and logistics systems more responsive, but at the same time more complex and therefore susceptible to unforeseen events and changes. Not only does the interconnection of systems not eliminate uncertainty, it also introduces the possibility that local unforeseen events can spread elsewhere. For this reason, systems capable of adapting during execution and reacting to disruptions are being designed.

In this context, automated decision-making moves from the design phase to the operational phase: small adjustment decisions made during execution. These decisions typically include prioritization, job assignment to available resources, and continuous updates to detect unexpected events or changes. This level of control does not replace design but integrates it into real, non-deterministic contexts.

Flexible Manufacturing systems (FMS) are born to meet these needs: integrated and automated production systems designed to manufacture a variety of products, handle changes in lot size or in product design, and continue to operate

in the event of failures.

In automated FMS systems, parts or batches travel through the production plant or warehouse to reach various internal resources, making internal handling an integral part of the process. A plant produces efficiently if parts arrive at the right place at the right time, so performance depends not only on the operating speed of the machines but also on the continuity of the flow between them.

A product within a flexible manufacturing plant completes a cycle of processing, transfer, and waiting that can be analytically managed through a transportation network consisting of connections, nodes, intersections and stations, and each transfer constitutes a temporal and organizational constraint. Stations and routes are shared resources between products, which generate conflicts. The simplest example of this phenomenon is road traffic: multiple vehicles share the road, and without coordination, traffic jams form, increasing the risk of accidents. Contrary to what one might think, traffic jams are not caused by too many vehicles on the roadway, but by the variability of their speed. If absurdly, vehicles could coordinate with each other, a number of vehicles close together that filled the length of the roadway in question could travel without slowing down. This coordination is impossible for road vehicles driven by a human, but it is actually possible for autonomous vehicles in manufacturing.

In the most advanced production facilities, such autonomy and coordination are already implemented: devices such as AGVs (Automated Guided Vehicles) and AMRs (Autonomous Mobile Robots) can navigate a network of defined routes and stations, communicating with each other and with the environment to achieve common goals through coordination mechanisms. Unlike urban traffic, coordination mechanisms can be designed: priority, assignment, and routing criteria can be introduced to reduce conflicts and delays. Consequently, the design and evaluation of coordination mechanisms is crucial, especially when the system is subject to variability and unforeseen events.

A coordination mechanism refers to the set of decision rules that, during execution, enable communication and thus decision-making between the handling equipment and the plant's logistics infrastructure. These mechanisms can be

implemented through distributed control systems, which are better suited for modeling local and negotiated decisions than centralized systems that potentially reach optimal solutions but are not suited for real-time action. Therefore, this study focuses on distributed control systems, because it aims to analyze performance that is critically dependent on local interactions and emerging phenomena.

The agent-based paradigm is the way these mechanisms have been studied and implemented in this work. An agent is a software entity associated with a physical resource in the system and endowed with a certain degree of autonomy. This entity ensures the distribution of decision-making and is capable of perceiving the environment, communicating with other agents in the system, and acting accordingly. Therefore, a multi-agent system (MAS) is defined when numerous agents interact with each other and the environment and have common goals. In this thesis, modeling handling systems and manufacturing machines as multi-agent systems enables the study of how coordination occurs when multiple resources compete on the same routes and service points.

Coordination mechanisms operate in real time, and in the presence of variability and perturbations, different coordination mechanisms can produce more or less satisfactory results and differ in terms of time, congestion, and flow regularity. Therefore, there is a need for universal criteria for evaluating and comparing coordination mechanisms, assessing not only performance under nominal conditions but also their ability to maintain acceptable performance when perturbations occur. For this reason, this thesis presents a method for evaluating and comparing coordination mechanisms that takes into account production time and congestion in a weighted sum.

This thesis stems from the activities carried out within the CoCeDi (Comparative Analysis of Centralized and Distributed Scheduling Approaches in Flexible Manufacturing Systems) project, which was part of an Erasmus Traineeship experience held at the SYMME Laboratoire at the University of Savoie Mont Blanc in Annecy, France, from September 1 to November 30, 2025. The objective of the project was to investigate the same case study through two complementary work packages: a centralized, offline optimization solution, and a distributed approach

particularly using agent-based model. This thesis focuses on the second package.

The framework for this work package is provided by the project section covered using robust offline optimization techniques, specifically solving the Flexible Job Shop Scheduling Problem with stochastic transport times, addressed using a hybrid Genetic Algorithm-Monte Carlo approach. The results of this work provided the foundation for the execution and comparison of the simulations that will be introduced later in the thesis.

The aim of this thesis is to propose an evaluation framework for the comparative evaluation of the coordination mechanisms used in internal production flows in manufacturing systems.

Chapter 2 reviews the existing literature on flexible, agent-based production systems and existing and under-studied coordination methods; Chapter 3 presents the proposed methodology for the comparative analysis of coordination mechanisms; Chapter 4 presents the AIP PRIMECA case study and the simulations carried out to evaluate the proposal made; Chapter 5 analyzes the results of the experiments; Conclusions are drawn in Chapter 6.

Chapter 2

State of the Art

2.1 Literature Review Methodology

In this section the techniques used for searching, categorizing, clustering, and selecting the reviewed papers will be analyzed. In the preliminary phase of the research work, the PRISMA (Preferred Reporting Items for Systematic Reviews and Meta-Analyses) paradigm was followed, which provides precise methodological guidelines for systematic reporting.

This section is structured around the main phases of the review process: literature search, study selection, and final selection. These three phases correspond to the 3 phases of the PRISMA paradigm *Identification*, *Screening* and *Inclusion*.

In Figure 2.1 you can see the PRISMA flow diagram with the steps followed for the paper selection.

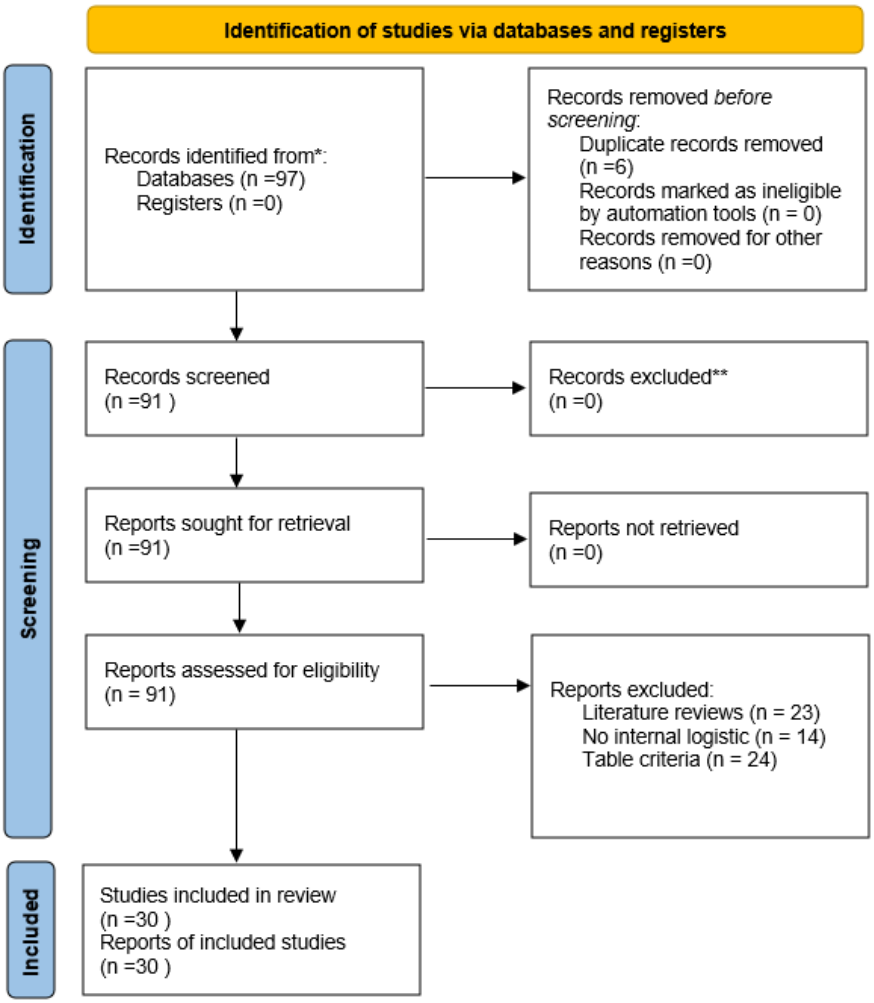

Figure 2.1: PRISMA flow diagram

Literature Search and Record Identification

In this phase, 97 articles were identified from Google Scholar and Scopus databases using the following keywords: *Swarm Intelligence, Collective Behavior, Multi-Agent Systems, Agent-Based Modeling, Distributed Intelligence, Emergent Behavior, Decentralized Systems, Collective Intelligence, Self-Organizing Systems, Complex Adaptive Systems, Virtual Platooning, Internal Logistics AGVs.* At this stage, an initial check was carried out and 6 papers that were found to be duplicates or proposed the same study were removed.

Study Screening and Selection Criteria

This phase involves screening abstracts and, if necessary, other key sections of the papers to determine their characteristics and choose the most suitable ones for the research. First of all, 23 papers were excluded because they were literature reviews and therefore not useful for the purpose. Other 14 were discarded because they did not address internal logistics or smart manufacturing as a primary application field. For the resulting 54 articles, a classification table using Microsoft Excel was used to determine the characteristics of the analyzed papers and therefore decide which studies to include and how to group them. In Figure 2.1 the header of the table is shown.

Objective	• Optimization • Cost reduction • Energy efficiency • Risk management • Quality improvement • Resilience enhancement
Solution method	• Mathematical programming • Heuristic / metaheuristic • Rule-based control • Simulation-based • Scenario-based
Decision supported	• Scheduling • Routing • Charging management • Fleet clustering / platooning • Task allocation • Resource allocation • Not specified
Architecture	• Distributed-based control • Semi-heterarchical control • Centralized-based control • Offline decision-making • Online decision-making
Emergence approach	• Task-level emergence • Resource-level emergence • Temporal emergence

Table 2.1: Header of the classification table

Table 2.1 reports the header of the classification table together with the categories defined for each column.

The last 3 categories are what represent what the study intends to analyze: autonomous coordination through emergent behaviors. Then it was possible to insert the papers in the table from the most recent to the oldest, marking with

an asterisk the characteristics that can be observed within.

Final Study Selection and Inclusion

At this point, 30 articles were selected as the most relevant. The columns related to the searched emergence were considered as a selection criterion. Specifically, all papers with at least two of the three searched characteristics were selected, along with the five most recent ones with only one of the three, thus discarding all those that did not contain any of these characteristics.

The most relevant papers were then divided into clusters, which served as a framework for compiling the state of the art. Specifically, four clusters were defined: two regarding the methodologies used and two regarding the applications of the studies. Each paper was assigned two clusters: one for each type through a weighted score given by the characteristics of each. Using a correlation matrix, it was possible to identify the characteristics most frequently present simultaneously in the papers and then determine the groupings as follows:

Methodological clusters (most correlated characteristics in Solution Methodology and Architecture): *Agent-based strategies* and *Mathematical programming*.

Application clusters (most correlated characteristics in Objective and Decision-Supported): *Operational Planning* and *Energy-saving Routing*.

The state of the art will therefore contain the 30 articles selected as explained and analyzed using the structure given by the clusters.

2.2 Literature review

The topic of Distributed Control Systems falls within the context of organizational regimes. Control authority can be organized as hierarchical, semi-heterarchical, or heterarchical.

In terms of control influence, a hierarchical system allows planning and coordination to flow vertically and centralized (top-down), while in heterarchies local autonomy and negotiation of decision nodes are guaranteed (peer-to-peer). Semi-heterarchical architectures can dynamically switch from more hierarchical

to more heterarchical modes based on the state of the system [1], [2], [3].

This way the degree of centralization varies according to state and objectives, combining global planning and local autonomy. Hierarchical organization facilitates global plans, sometimes, getting close to the achievement of optimal control solutions, while more heterarchical configurations favor reactivity and robustness: the system can rapidly adapt and is more suitable for online decisions in dynamic contexts where disturbances and failures occur frequently [2], [4].

This organizational trade-off motivates the focus of this thesis on exploring it in the FMS context: how flow regularity and congestion phenomena emerge during execution. That may not be reflected by offline plans given by a centralized regime [5], [6].

Having established the organizational regimes context, it is now possible to examine the control strategies that operationalize them on the shop floor.

2.2.1 Agent-based strategies

The agent-based paradigm is a cornerstone of distributed control, this is used to represent local decisions made through interaction and negotiation. Here Multi Agent Systems (MAS) are considered as a practical implementation of heterarchy and semi-heterarchy, the goal is to combine reactivity and global performance [7], [8].

In particular, [7] defines MAS as a software technology that is able to model and implement individual and social behavior of so called agents in distributed systems.

In the manufacturing field, MAS is translated into Holonic Manufacturing Systems (HMS) a set of autonomous, self-reliant manufacturing units, called holons. Holons are autonomous and cooperative units that can transform, transport, and store physical objects or information, and they are organized in holarchies (semi-heterarchies) to achieve production goals cooperating with common rules where neither pure hierarchy nor pure heterarchy is enough [6].

The reference architecture for HMS is PROSA (Product-Resource-Order-Staff

Architecture) [9]. In a Flexible Manufacturing systems assigning information and decision responsibilities is an important stage when designing decisional architecture.

This architecture distinguishes between Basic Holons [5], [6]:

- **Product Holons**: handle information related to the product process and lifecycle and act as information servers;
- **Resource Holons**: encapsulate the control system within the resource;
- **Order Holons**: (often coinciding with the workpiece and managing its state) negotiate with Resource Holons to manage instances and products;
- **Staff Holons** (optional): facilitate information exchange and efficiency, reducing the need for centralized control.

PROSA is not just a conceptual framework, the distinction between agents of different nature can also be useful at the implementation level, this mapping can help clarify what is autonomous from what requires coordination.

[5] extends the PROSA conceptual framework with a model for systematic identification of manufacturing Holons (Product, Resource, Order, and Staff) and their interactions; this framework enables the dynamic composition and reconfiguration of holarchies over time.

Alongside HMS architectures, some recent literature integrates concepts of interoperability and modularity to obtain self-description interface for components/agents and to support flow recovery strategies in the presence of disturbances. An example in [10], which design a semi-heterarchical MAS/HMS for modular production that uses Asset Administration Shells (AAS) as interoperable interfaces for self-description of components/agents and flow recovery under disturbances. That couples global discrete planning with local decentralized execution within a holonic, recursive structure.

An agent multilayer architecture used in HMS is *InteRRaP*, structured in three layers: CPL (cooperative planning and negotiation), LPL (local problem solving), BBL (execution, interface with the world). This decomposition makes

it natural to combine local decisions and global coordination without forcing absolute centralization [8].

Some architectures provide the roles and degrees of autonomy, while MAS (for example, InteRRaP) can offer the negotiation and execution mechanisms.

The literature also proposes holonic and MAS platforms oriented towards industry. A representative case is *Factory Broker*, a holonic platform with a three-tier architecture: Physical Layer, Execution Layer and Coordination Layer that realizes the Product/Resource/Order roles as workpiece. It represents industrial evidence of the applicative effectiveness of MAS tested on real industrial HMS pilot line. It has been implemented and tested on a DaimlerChrysler cylinder line with results consistent with the simulations and has the aim of enabling negotiation and coordination between agents and integrates legacy systems (PLC/CNC on DCOM/Ethernet) [11].

From the point of view of the coordination mechanisms of the agents, conflict-free access to shared resources is handled via distributed reservation protocols, convoy-formation strategies (especially for vehicle fleets (AGV/AMR)), and coordination at intersections through priority, speed of approach and decentralized conflict resolution [12]. Complementary convoy-centric designs implement distributed control of material-handling networks [13] , and are evolving toward integrated scheduling–plus–trajectory formulations in fully automated settings [14].

Another way to perceive the elements in the system is the DIND (Distributed Intelligent Networked Device), this type of architecture is based on an 'intelligent space' with sensing, processing and networking functions reducing on-board computational complexity moving sensing and processing to the environment [15].

Complementarily, at the local-policy layer, cooperative obstacle, collision avoidance and deadlock-prevention schemes act as the safety substrate for distributed motion coordination in order to operate reducing operational blocks or risks [16], [17].

The BDI (Belief–Desire–Intention) approach proposed for FMS by [3], thanks to adjustable autonomy, shows that tuning the autonomy improves performance compared to a static scheme, in the presence of perturbations.

Another three-tier MAS architecture is presented in [18]: a knowledge-based ERP which manages global planning interacting with external systems, High-level Controller which analyses the system performance and provides recommendations for optimization and the Low-Level Controller executes direct actions on the field (machines, resources), with a certain decision-making autonomy. This way, WIP control is improved while maintaining resource optimization capabilities. Under realistic variability and faults, comparative analyses show decentralized/heterarchical control outperforming centralized baselines [4].

Additionally, [19] propose an architecture that combines process mining and MAS validated by a simulation of a Job-shop scheduling problem in order to observe emergent behaviours in MAS-driven shop floors (AGV job-shop case), offering a data-driven way to assess decentralized decision rules through event logs that feed agent behavior. Complementary contributions model collective behavior via consensus/leader–follower (eg. Leader-Vicsek Model) [20] dynamics and self-organization, introduce adaptive policies at the supervisory level for Automated Material Handling Systems (AMHS) [21], and formalize decision making as multi-agent reinforcement learning or agent-oriented planning with correctness properties [22], [23] [24], [25]. Agent-based architectures offer scalability, responsiveness, and resilience, especially when coupled with effective coordination and formal security guarantees.

2.2.2 Mathematical programming

In studies on the organization of AGVs in manufacturing contexts, mathematical programming is often used as an offline benchmark.

The mathematical programming approach proves weak for online control: [26] proposes a complexity framework for comparing MIP and MAS via simulation, finding as a result that the computation time of the centralized one grows with demand and size, while the distributed one remains almost constant.

Multi-objective (MILP) problems are formalized, often linearized, using compromise approaches such as interactive fuzzy programming [27]. In the presence of complex and multi-load networks, heuristics, task grouping by adjacency and

shortest-path, priority rules and reservation timetables for conflicts are used, while empirically showing that exact optimization typically does not scale online [28].

As a robustness benchmark, a MILP flow-shop [29] minimizes makespan and balances machine utilization to limit emergent over-utilization behaviors and evaluates reliability by introducing delays on process times and using disjunctive graphs.

Methodologically, MP offers ground-truth baselines and stress-tests; online control is better supported by distributed mechanisms.

2.2.3 Operations planning

Studies on operations are numerous and address real-time decisions in intralogistics: task allocation, routing/traffic management, scheduling and conflict/deadlock mitigation, as well as resilience and monitoring aspects. One way to avoid conflicts hierarchically is to use a centralized reservation timetable with priority rules to hierarchically resolve node conflicts across loading/middle/unloading layers, a variable-neighborhood search further refines multi-AGV task assignment demonstrating the effectiveness on large and dense networks of AGVs [28]. These reservation-based strategies are useful to coordinate local flow and prevent conflict convergence.

A more flexible way of handling conflicts was studied by [17] via agent-based on AMRs. The evaluation compares flexible versus dedicated design (fixed zone, no conflict possible), showing substantial improvements on flow performance.

Beyond conflict prevention, in a dynamic environment, local carrier sequencing at the unloading station can be achieved without a global controller, for example AGVs can have the ability to learn a local probability matrix and update it via relative evaluation of neighbors within a strategy range [23].

[15] uses DIND nodes (see paragraph on Agent based) in order to avoid conflicts at intersections by detecting dynamic obstacles, even people, applying precedence and reservation rules.

Regarding fleet management in variable environments, adaptive workstation

clustering based on material flow and layout is studied by [21]: they combine network analytics (modularity) with optimization tools and greedy refinement showing comparative advantages in resource utilization.

In [19], agents read models and re-calibrate decisions for operational diagnosis and management of emergent behaviors.

The scheduling complexity framework of [26] is based on environment variables, evaluated via MAS on different job shops. In the AGV job shop, model miners detect bottlenecks and the impact of dispatching rules on throughput time. To assess operational reliability in the event of disturbances. [29] reformulated a flexible job shop as a flow-shop by explicitly limiting emergent behaviors due to excessive use of shared resources and testing robustness by injecting delays.

For fast replanning on the dispatching side, [24] formalize an agent-oriented planning with a meta-agent that quickly decomposes/allocates tasks and re-plans with feedback via a reward model, useful when assignments and scheduling need to be adjusted under time constraints.

On the operational resilience and adjustable autonomy side, through AAS, [10] proposes a resilient production flow in modular systems in which agents coordinate cross-level decisions and reassign processes and resources to maintain KPIs in the presence of deviations and disturbances.

In the internal handling field, [3] demonstrates that the adjustable autonomy of AGVs improves performance compared to static levels, allowing the LOA to be adjusted in critical situations.

In the presence of disturbances and failures, [4] compare centralized vs heterarchical operation on assembly and material handling: also in this case the heterarchical one shows lower standard deviation of throughput time, operating more stably under uncertainty.

See also [2] for self-organizing hierarchical-heterarchical behaviors that improve operation on complex orders. In conclusion, in operational planning, local rules, anti-deadlock mechanisms, and online replanning stabilize flows and reduce lead time variance, improving robustness and service levels compared to centralized baselines.

2.2.4 Energy-saving routing

Coordinated routing and velocity profiles can improve the efficiency of manufacturing systems employing distributed mobile robots, through multi-objective optimization or dynamic coordination [12].

[25] proposes a learning-based multi-AGV coordination by formulating a MARL (Multi-Agent Reinforcement Learning) problem in CTDE (Centralized Training, Decentralized Execution) scheme to integrate task assignment and path planning. The results show a reduction in energy consumption compared to the standards, while maintaining obstacle avoidance and fast convergence capabilities.

On the optimization side [27] addresses AGV scheduling in a FMS with a MILP model that minimizes energy consumption and maximizes workstation satisfaction. Energy is a function of load and speed, and the energy–service tradeoff is solved via Interactive Fuzzy Programming (IFP), providing more energy-efficient scheduling solution without sacrificing service level.

For coordinated routing, a model such as Leader–Vicsek for multi-AGVs or HSI (Human–Swarm Intelligence) generates group trajectories with simple neighbor rules [20], [22].

In this section, platooning is discussed as a coordinated routing strategy [27] who propose a distributed control for AMHS (Automated Material Handling Systems) which in simulation has been shown to decrease energy and waiting times and also plan charging without congesting the fleet.This is based on "time-of-arrival" and potential fields: an interesting concept in which the shop floor is crossed by a field of forces that are attractive for objectives and repel obstacles.

For plant adoption, the safety and security requirements of AGV (Automated Guided Vehicle) platooning in Industry 4.0 contexts are systematized and supported by industrial communication architectures [30].

As a cross-layer application, align routing with charging by planning SoC (State of Charge) thresholds and pre-booking charging slots along the route [14], preventing queues and detours while keeping energy per mission low.

At intersections, this is referred to as virtual platooning [12] with distributed

coordination of approach speeds, increasing capacity/throughput and therefore decreasing the required network energy.

As regards obstacle avoidance, there are examples in the literature of standard ETSI ITS messaging (CAM/DENM/CPM) that allow safe group maneuvers around static and dynamic obstacles [16]. In energy-efficient routing, platooning is the critical lever that reduces stop-and-go and consumption, increasing capacity without sacrificing safety.

2.3 Research gap and thesis contribution

The literature review conducted in Section 2.2 confirms the relevance of distributed and agent-based control for coordination in the FMS context. It also highlights how the offline mathematical programming approach is more suitable for scheduling problems in the absence of perturbations or for obtaining baseline comparison scenarios.

Another aspect that emerges from the literature review is that performance is affected by emerging flow phenomena such as congestion, which represent objectives complementary to operational efficiency.

However, the review also reveals a multitude of different approaches to distributed control in the FMS context and therefore a lack of universal evaluation procedures capable of comparing heterogeneous coordination mechanisms and assessing their behavior under disturbances. Another under-researched aspect is the evaluation of emerging phenomena regarding the fluidity of resource flows within production plants, phenomena that can impact operational efficiency as well as overall safety.

To address these gaps, this thesis proposes a reproducible framework of agent-based digital twins for online coordination experiments. Four different coordination mechanisms for AGV flows in FMS are proposed. A method for evaluating coordination mechanisms based on a weighted-sum index is also introduced, combining makespan and average congestion, modulating the weights according to the decision maker's preferences.

In Chapter 3 the method proposed by the thesis is explained in detail and in 4 it is tested through the AIP-PRIMECA Valenciennes Case study.

Chapter 3

Proposed Methodology

From the scientific literature in the field of distributed control systems in manufacturing and logistics, a gap emerges regarding shared systems for evaluating coordination mechanisms.

Therefore, the aim of this work is to propose a comparative method for evaluating online coordination mechanisms in FMS. It also lays the foundation for analyzing the performance degradation under perturbations.

The following paragraph presents the application of four coordination mechanisms via a digital twin following the agent-based paradigm. A comparative method is then introduced for evaluating the coordination mechanisms through the weighted sum of two fundamental KPIs for FMS performance.

3.1 Digital twin agent-based framework

This section presents a digital twin framework that uses the agent-based paradigm to implement online coordination mechanisms in FMS. It enables online mechanism evaluation, production data capture, and experiments reproduction.

It is called a digital twin because it perceives key parameters from the production cell in real time via independent sensors, effectively producing a digital copy of it.

In this case, the digital twin is used to implement online logic and subsequently measure KPIs, such as production times and congestion. Specifically, the role

of the digital twin is to receive real-time information from the production cell, calculate the behavior of the agents, and implement actions without interrupting the production flow. For example, the system can receive the completion status of products as input and assign a priority to each as output.

As reported in the literature review (Chapter 2) the agent-based modeling allows each agent in the system to perceive the environment in real time and act accordingly negotiating with each other. In this way the implementation of coordination mechanisms is autonomous and guided by common objectives at the same time.

Agents are all entities capable of independent perception and action within the system. In an FMS, agents can include:

- **Product/AGV agents**: based on the perception of the environment and other agents, the product or what transports it (e.g. AGV), decide routing and speed while moving around the plant;
- **Resource agents**: workstations that expose local state and capabilities and act accordingly;
- **Infrastructure agents**: nodes, intersections, and conveyors that sense congestion and direct jobs within the production cell.

In this case, agents are modeled as entities that provide inputs to the coordination mechanisms (perception) and receive their outputs in the form of actions.

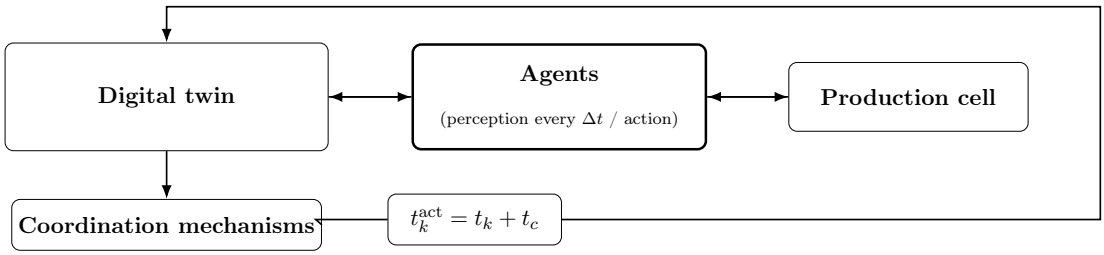

Figure 3.1: Conceptual schema of the digital twin.

The agents act as an interface between the two levels: the digital level that orchestrates the control and the real level of the production cell. In this way

the perception of the environment's state is provided for each individual agent as input for decision-making, and conversely, the actions calculated are applied to the production cell.

Given the online nature of the control, let Δt be the time interval between two consecutive perceptions of the system state by the digital twin, and let t_c be the computation time of the coordination mechanism. Therefore, an action computed at perception time t_k is implemented at:

$$t_k^{\text{act}} = t_k + t_c. \tag{3.1}$$

This delay in applying the coordination mechanism requires attention in the choice of sampling interval Δt and computational burden. It is important that the following inequality is respected:

$$t_c < \Delta t. \tag{3.2}$$

Otherwise, the subsequent perception of the environment and of other agents would occur before the system has acted, sometimes causing oscillations in the behavior of the agents.

Figure 3.1 shows the conceptual diagram of the proposed production cell-software communication framework. The physical layer of the framework is represented by the production cell, while the controller is represented by the digital twin. Note the agents as software entities that bridge the physical and digital layers independently through system perception and guided action. The action is computed by coordination mechanisms, which provide reactive and cooperative rules via algorithms that channel the agents into actions. Here is where the agents communicate with each other to achieve coordination.

In the next section, four coordination mechanisms specific to FMS are proposed.

3.2 Coordination mechanisms

This section presents the coordination mechanisms proposed in this study. These online logics were conceived starting from ideas in literature and were reworked by combining and tailoring complementary concepts such as congestion-aware routing, schedule-driven rerouting, booking, and speed coordination, into a coherent set of mechanisms for decentralized routing selection and congestion management in the considered FMS. The goal is to introduce methods for dynamic individual routing selection and congestion management through decentralized coordination.

Consider a production cell equipped with workstations (or machines) and products that must perform a predetermined sequence of operations on them to be completed. The flexibility of the cell is given by the feasibility relationship of these operations: each machine performs only some operations, while multiple machines can perform the same operation.

Products are assumed to have a predetermined and optimized default nominal routing through the cell to reach the workstations for operations.

For comparison, a rule-free configuration is also considered, in which each job follows the nominal routing without applying any reactive logic. The coordination mechanisms studied are described in detail below; therefore, some common parameters are defined here (job refers to the single product):

- p: job being processed by the coordination mechanism;
- h: current target workstation the job is routed to;
- d_h: distance between the job and workstation h;
- op: next operation required by the job;
- $\mathcal{W}$: set of workstations included in the production cell;
- $f(m)$: workstation m is free (not processing any job) at the current step;
- $u(m)$: workstation m is processing a job at the current step.
- n_m: local congestion/crowding metric observed for workstation m.

In the following, each job p is assumed to be carried by an AGV; therefore, decisions on destination h and distance d_h are implemented by the AGV transporting p. Please note that coordination mechanisms are designed to achieve results in terms of limiting production time and congestion, in some cases simultaneously and in others exclusively.

3.2.1 *Congestion avoidance*

The *Congestion avoidance* coordination mechanism, for each job, implements a reassignment of the target workstation with the objective to route the AGV carrying each job toward less congested areas of the cell, thus distributing congestion more evenly. Consider the metric n_m as congestion, which is perceived locally by each workstation as the number of jobs within a radial distance corresponding to a waiting area at workstation m. This corresponds to the crowding in the vicinity of the resource.

To avoid instability caused by changes in destination, it is important to ensure the logic does not intervene when the product is close to the destination workstation; the distance threshold between the job and the workstation within which the rule does not intervene is defined as $D_{\min}(h)$. In this case, the routing is maintained.

Therefore, each job compares the current congestion of the target workstation h with that of alternative workstations that can perform the next operation *op*. Among the eligible candidates, the workstation with the lowest congestion metric n_m is selected. If there is a workstation with a congestion n^* lower than that of h, the job updates its target to the corresponding workstation m^*; otherwise, it maintains h.

Algorithm 1 schematically shows how *Congestion avoidance* allows product agents to communicate with resource agents to achieve less congested routing.

Algorithm 1 *Congestion avoidance*

Parameters: $D(\cdot)$
if $d_h < D(h)$ **then**
    return
end if
$m^* \leftarrow h$
if $u(h)$ **then**
    $n^* \leftarrow +\infty$
else
    $n^* \leftarrow n_h$
end if
for all $m \in \mathcal{W}$ **do**
    if $u(m)$ **then**
        continue
    end if
    if *op* is not executable on m **then**
        continue
    end if
    if $n_m < n^*$ **then**
        $m^* \leftarrow m$
        $n^* \leftarrow n_m$
    end if
end for
if $m^* \neq h$ **then**
    $h \leftarrow m^*$
end if

3.2.2 *Delay triggered reassignment*

The *Delay triggered reassignment* coordination mechanism redirects jobs that are behind schedule to a high-flexibility workstation m_u, which is a resource with broader capabilities than standard workstations. The goal is to reduce the

propagation of delays between jobs and therefore reducing the total production time.

This coordination mechanism requires that the nominal routing plan include, for each job, not only the workstations to be visited, but also the expected completion times of each operation.

For each job, the difference between the actual completion time of the last operation and the planned completion time is considered. If this difference is greater than a predetermined threshold Del, the job is considered late and is redirected to the high-flexibility workstation for further processing.

Algorithm 2 reports the pseudocode for *Delay triggered reassignment*. Consider `seq_order` as the cardinality of the next operation to be performed by the job being processed. Note how the last completed operation and any delay in its completion are taken into account.

Algorithm 2 *Delay triggered reassignment*

Parameters: Del
$i \leftarrow$ `seq_order` $- 1$
if `real_completion`$[i]$ − `planned_completion`$[i] > Del$ **then**
    mark job as **Delayed**
    if *op* is executable on m_u **then**
        $h \leftarrow m_u$
    end if
else
    mark job as **On-time**
end if

3.2.3 *Reservation board*

The *Reservation board* coordination mechanism aims to prevent multiple jobs from converging on the same resource, thus reducing routing fluctuations. To

achieve this, a virtual reservation table is defined and, through a booking mechanisms, jobs coordinate themselves in the cell in order to avoid congestion.

The booking logic is as follows: the table contains all the workstations. Jobs reserve the workstation they are headed to h, only if it has not already been reserved by another job, and then proceed to that workstation to perform the operation.

If the machine specified in the nominal plan is already booked, the job searches among the available machines for one that is not booked. The candidate machines set are indicated as C in the pseudocode. If it finds one (C is not empty), it books it; otherwise, it wanders without a destination in the cell until one of the available machines becomes free. This way jobs that are not booked don't stop avoiding obstruction for the booked jobs.

Algorithm 3 reports the pseudocode. Note that RB represents the *Reservation board*, which can be modeled as a dictionary with as many keys as workstations and is initially empty. Whenever a reservation is made in the table cell corresponding to the machine in question, the ID of the booking job (`who`) and its current `seq_order` are recorded. This sequence order is used by the mechanism to free the reservation: if the job has finished processing the machine, its `seq_order` is incremented respect to the one recorded on the board and it leaves the space on it.

Algorithm 3 *Reservation board*

Parameters: reservation table $RB[m] \in \{\emptyset, (\texttt{who}, \texttt{seq\_order})\}$ for each workstation m
$j \leftarrow \texttt{seq\_order} - 1$
for all $m \in \mathcal{W}$ **do** ▷ reservation maintenance
    if $RB[m] = (p, k) \wedge \texttt{completed\_ops}(p) > k$ **then**
        $RB[m] \leftarrow \emptyset$
    end if
end for
if $f(h) \wedge (RB[h] = \emptyset \ \vee \ RB[h]$ assigned to $p)$ **then**
    set $RB[h] \leftarrow (p, j)$
    return
end if
$\mathcal{C} \leftarrow \{m \in \mathcal{W} : m \neq h, \ f(m), \ op$ executable on $m, \ RB[m] = \emptyset\}$
if $\mathcal{C} \neq \emptyset$ **then**
    choose $m^* \in \mathcal{C}$
    set $RB[m^*] \leftarrow (p, j)$
    $h \leftarrow m^*$
else
    remove the current h (job keep moving)
end if

3.2.4 *Virtual platooning*

The *Virtual platooning* coordination mechanism is designed to control the speed of AGVs carrying jobs, specifically aiming to modulate the transport speed of each job so that it arrives at the workstation ideally exactly when it becomes available. This avoids queues near the processing machines and the resulting congestion.

The concept of *Virtual platooning* comes from the literature on autonomous vehicles at intersections. It is a way for autonomous vehicles to coordinate pri-

orities and adjust speeds accordingly, avoiding stop-and-go traffic. In this case, it is being applied in manufacturing to coordinate the arrival of AGVs carrying jobs at workstations, thus limiting congestion caused by stop-and-go traffic near them.

In this case platoons are formed by dividing the jobs into groups based on their current destination workstation. The jobs in each platoon are then ordered in increasing order of distance from the workstation, and the closest job is designated as the leader, the others according to ordinality. The leader proceeds towards the workstation at a constant speed, while the remaining jobs in the platoon adjust their speed based on the time the preceding job finishes its last operation at the workstation and its distance from it.

Specifically, each job in the platoon, via the digital twin, communicates to its follower its speed, distance from the workstation, and its next operation. The job receiving this information can calculate the arrival time of its predecessor based on its speed and distance from the workstation, and the completion time of the operation based on the next operation. This latter time corresponds to the ideal time to reach the workstation. Based on this objective and the distance from the machine, the job adjusts its speed. And so on for the other followers in the platoon.

In Algorithm 4, $SF(p)$ is a *speed factor* applied to the nominal transport speed of job p: $SF_{\min}$ and $SF_{\max}$ are the minimum and maximum admissible values in the system. The parameter α is a scaling factor that maps distance into travel time under nominal conditions.

Algorithm 4 *Virtual platooning*

Parameters: $SF_{\min}$, $SF_{\max}$, α
$G_h \leftarrow \{p : \text{jobs with target } h\}$
sort G_h by increasing d_h to obtain $[p_0, p_1, \ldots, p_n]$
$SF(p_0) \leftarrow SF_{\max}$ ▷ leader
$T_{arr}(p_0) \leftarrow \dfrac{d_h(p_0)}{SF_{\max}\,\alpha}$
$T_{op}(p_0) \leftarrow$ remaining processing time of p_0 at workstation h
for $i \leftarrow 1$ **to** n **do**
    $T_{arr}(p_i) \leftarrow T_{arr}(p_{i-1}) + T_{op}(p_{i-1})$
    $T_{op}(p_i) \leftarrow$ remaining processing time of p_i at workstation h
    $SF(p_i) \leftarrow \text{clip}\left(\alpha\,\dfrac{d_h(p_i)}{T_{arr}(p_i)},\ SF_{\min},\ SF_{\max}\right)$
    apply $SF(p_i)$
end for

3.3 Evaluation framework and composite performance index

In this section, a method for evaluating and comparing online coordination mechanisms is proposed. The method consists of a weighted sum of two key parameters in this type of FMS: total production time and average congestion.

To comparatively evaluate coordination mechanisms, a scenario-based approach is adopted. A scenario is considered a combination of three components:

- **Order set**: the unordered set of jobs to be executed.
- **Coordination mechanism**: one of the mechanisms explained in Section 3.2. Scenarios to which, for comparative purposes, no coordination mechanism is applied are referred to as *Rule-free*.
- **Perturbation**: an unforeseen event that requires the reorganization of production respect to the *a priori* plan. Scenarios without disturbances are also considered.

As said before, for each scenario, the overall production time and average congestion are considered as two key metrics: the first in terms of production efficiency, the second in terms of routing security. Congested routing can lead to an irregular flow of components in the production cell, which can cause unexpected events and malfunctions. The metrics and methodology adopted are as follows.

3.3.1 Makespan

The production makespan refers to the time in seconds that elapses between the start of the first operation (loading) of the first job launched and the end of the last operation (unloading) of the last remaining job; the formal definition is provided below.

Let $\mathcal{P}$ be the set of jobs released in the production scenario. For each job $p \in \mathcal{P}$, the following are defined: t_p^{rel} as the release time in the system and t_p^{out} as the completion time of the last operation.

$$M = \max_{p \in \mathcal{P}} t_p^{\text{out}} - \min_{p \in \mathcal{P}} t_p^{\text{rel}}. \tag{3.3}$$

The makespan M therefore provides a measure of the system's speed in the overall order set production process. A lower M corresponds to greater system time efficiency, making this metric fundamental for evaluating online coordination mechanisms in various scenarios.

However, it can be argued that this data alone does not capture all the information necessary for the evaluation but focuses only on efficiency, neglecting aspects such as routing security, job transport fluidity, and distributed use of resources. Therefore, an average congestion metric is introduced in the next section.

3.3.2 Average congestion

In this section, average congestion is explored as a metric indicating the fluidity of job movements in the production cell, with particular reference to the queues

at the workstations and therefore their utilization.

This aspect, as highlighted in the literature, is little considered, but important as it generates stop and go and deadlocks, which are directly linked to safety problems and the complexity of managing unexpected events in highly autonomous environments.

Congestion is defined as the number of closely spaced AGVs that interfere with each other during routing (queues of AGVs carrying jobs in waiting areas). Since in the case of FMS such congestion primarily occurs near the workstations, where the jobs are waiting to be processed, the following definition applies.

Let $\mathcal{W}$ be the set of workstations in the production cell. The following definitions are introduced: $n_m(t)$ as the number of jobs within a radius Rad of workstation m at time t, and $c(t)$ as the maximum $n_m(t)$ among all workstations at time t.

$$c(t) \;=\; \max_{m \in \mathcal{W}} n_m(t). \tag{3.4}$$

By sampling the production time at constant intervals Δt and denoting the set of sampled time instants as $\mathcal{T}$, the average congestion is defined as:

$$C \;=\; \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} c(t). \tag{3.5}$$

Therefore C is the time average of the maximum number of jobs within radius Rad of the system's workstations W.

Note that in the case of the digital twin discussed above (Section 3.1), Δt corresponds to the interval between the perceptions of the agents and environment. Similarly, in the workstation agents, the perception of congestion occurs at Δt time intervals.

3.3.3 Metric normalization

The metrics M and C have different units and scales: M is a time in seconds on the order of thousands, and C is an average number of jobs on the order of tens. Therefore, in order to combine them into a single indicator, a min-max normalization is applied within each scenario, after aggregation across the replicas.

Let R be the number of independent replications of the scenario considered. The sample means are defined as follows:

$$\bar{M} \;=\; \frac{1}{R}\sum_{r=1}^{R} M^r, \qquad \bar{C} \;=\; \frac{1}{R}\sum_{r=1}^{R} C^r. \tag{3.6}$$

For each scenario, the minimum and maximum values of the metric are identified with respect to the set of coordination mechanisms compared (rule-free and the other coordination mechanisms):

$$M_{\min} = \min\{M\}, \quad M_{\max} = \max\{M\}, \qquad C_{\min} = \min\{C\}, \quad C_{\max} = \max\{C\}. \tag{3.7}$$

The min-max normalizations are therefore:

$$\tilde{M} \;=\; \frac{\bar{M} - M_{\min}}{M_{\max} - M_{\min}}, \qquad \tilde{C} \;=\; \frac{\bar{C} - C_{\min}}{C_{\max} - C_{\min}}. \tag{3.8}$$

Note how this type of normalization causes the values of $\tilde{M}$ and $\tilde{C}$ to be between 0 and 1, and that in both cases values close to 0 indicate good performance of a coordination mechanism in the scenario and vice versa for values close to 1.

In the next section, the values of $\tilde{M}$ and $\tilde{C}$ are combined into a weighted sum that constitutes a unique index for evaluating the performance of the coordination mechanisms within each scenario.

3.3.4 Weighted sum index

To compare the mechanisms under the same scenario, the two normalized metrics $\tilde{M}$ and $\tilde{C}$ are combined into a weighted sum I.

M and C represent two different objectives, both to be minimized: completion time and traffic congestion of jobs within the production cell. The first measures the system speed while the second measures the safety of the routing.

Both of these aspects are central to the efficiency of an FMS; however, depending on circumstances, priorities, and preferences, one aspect may be given more weight than the other in the evaluation of the coordination mechanisms.

For example, if an order set has a due date that's significantly later in time, then makespan has less weight than congestion in the comparison. In this case, the weight of the coordination mechanisms evaluation falls more heavily on congestion.

This aspect has been modeled using the weights ω_M and ω_C in the weighted sum, whose sum must be equal to 1 and whose respective values represent the importance of one metric compared to the other in the evaluation.

$$I = \omega_M \, \tilde{M} + \omega_C \, \tilde{C}. \tag{3.9}$$

$$\omega_M + \omega_C = 1, \qquad \omega_M, \omega_C \in [0, 1]. \tag{3.10}$$

Since $\tilde{M}$ and $\tilde{C}$ are obtained through min-max normalization, then $\tilde{M}, \tilde{C} \in [0, 1]$ and, due to the condition $\omega_M + \omega_C = 1$, also $I \in [0, 1]$.

Since both metrics are formulated as values to be minimized, lower values of I indicate a better trade-off between completion speed and congestion level given the preferences of the decision maker in the scenario considered. This allows for comparing and ranking coordination mechanisms under the same scenario.

Chapter 4

The AIP-PRIMECA Valenciennes Case Study

This chapter presents the case study and the related experimental model used to analyze the performance of the coordination mechanisms.

In section 4.1, the AIP-PRIMECA benchmark from which the experimental model is developed is presented. This is followed by a definition of the terminology and assumptions made during the experimental phase, the layout of the production cell used in the experiments, and the flexibility of production resources. Finally, the implementation of the digital twin and the instantiation of the experiments are discussed.

The aim of the case study was to test the proposals in Chapter 3: the digital twin, the related proposed coordination mechanisms, and the methodology for evaluating the experiments.

4.1 AIP-PRIMECA benchmark

The case study is based on the AIP-PRIMECA benchmark [31], proposed by Trentsaux et al. to provide a recognized reference for comparing scheduling and control methods in FMS. This benchmark consists of a realistic production cell with machinery and rails on which the products in motion move. In this thesis, AIP-PRIMECA is used to test the framework proposed in Chapter 3, with some

modifications to adapt it to the case study.

First, the benchmark used for the experimental phase of the study is presented. The assumptions and parameters applied to the benchmark to adapt it to the study are then specified. It then explains how the digital twin was implemented in compliance with the framework in Section 3.1. Finally, it explains how the simulations were set up.

The experiment was conducted by developing a digital twin of an FMS cell based on the AIP-PRIMECA benchmark [31], designed via NetLogo 6.2.2 to simulate agent-based control distribution. NetLogo is a programmable modeling environment designed for simulating natural and social phenomena, developed at the Center for Connected Learning and Computer-Based Modeling (Northwestern) and is widely used for building multi-agent models. The challenge, therefore, was to observe in the NetLogo environment emerging phenomena such as congestion and formalize them analytically.

The AIP-PRIMECA Valenciennes benchmark, describes a real manufacturing cell located at the University of Valenciennes, France (UVHC) at the TEMPO research center. This benchmark is well-known in the literature and used as a reference for comparing optimization, scheduling, and control strategies in the Flexible Manufacturing Systems (FMS) field. In this thesis, the benchmark is used as the basis for building an agent-based digital twin, as it is particularly suited for comparative analysis between centralized and distributed scheduling solutions. As previously mentioned, this dissertation focuses on the distributed online perspective on the case study. A photo of the real cell is reported in Figure 4.1.

Figure 4.1: Photo of the AIP-PRIMECA manufacturing cell [31] at the University of Valenciennes (UVHC), TEMPO research center.

4.1.1 Benchmark terminology

The following explains the terminology used in the AIP-PRIMECA and therefore in the case study to avoid terminological ambiguity.

Four levels are distinguished to describe what's produced in the cell: components, jobs, products, and client orders.

- **Components**: basic elements `plate`, `axis`, `r_comp`, `I_comp`, `L_comp`, `screw` (see Figure 4.2), which are assembled to form jobs through assembly operations. The system is assumed to have infinite availability of components as the focus is not on material management.

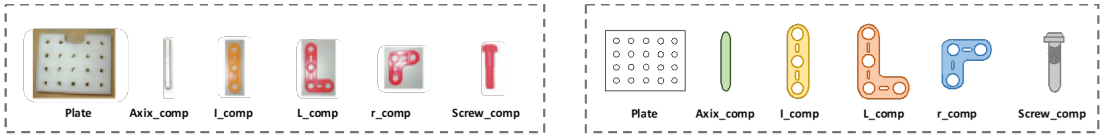

Figure 4.2: Components defined in the AIP-PRIMECA benchmark [31].

- **Jobs**: letters made up of components. The letters that the components can form are contained in the set: {B,E,L,T,A,I,P}; each job is defined by a sequence of elementary operations and component-assembly operations. Figure 4.3 shows the benchmark job types and their representations. As shown, jobs are made up of assembled components. Below, the operational sequences for assembling the components in each job will be discussed.

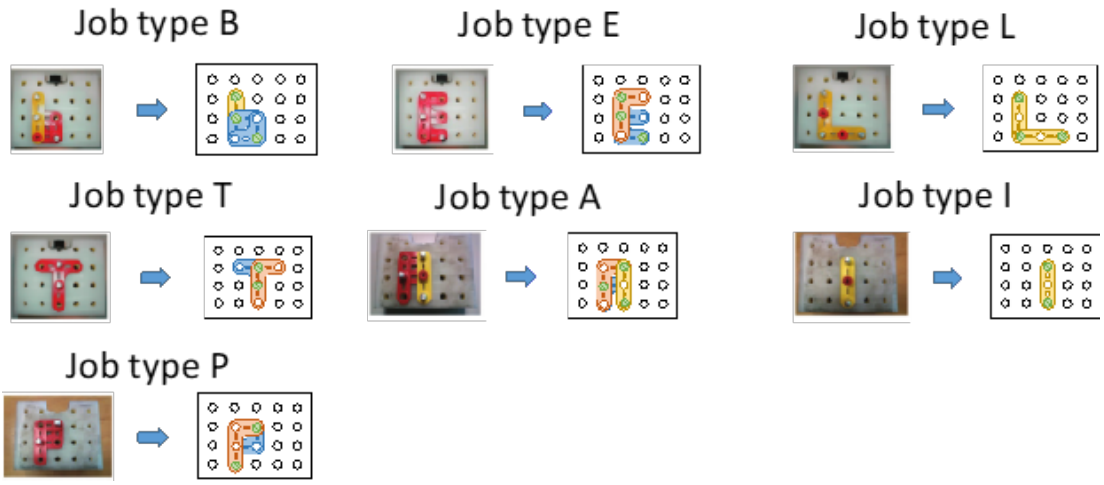

Figure 4.3: Jobs defined in the AIP-PRIMECA benchmark [31].

- **Products**: in the benchmark, a product is defined as a set of jobs that make up one of the following three job sets: BELT, AIP, LATE. In the remainder of this thesis, however, the term *order set* is used to refer to: any combination of jobs (letters). The jobs that make up an order set can be produced in an order not necessarily constrained by the sequence in which the jobs are defined, provided that the operational sequences are respected.

- **Client orders**: in the benchmark it is defined as a set of products with two associated dates (deadlines); however, this aspect is not considered in the present thesis, as the focus is on online coordination rather than scheduling.

To avoid every terminological ambiguity, in this thesis the term *job* always indicates a letter (B, E, L, T, A, I, P), while *order set* indicates any combination of letters. Having defined the entities managed by the benchmark, the following

are descriptions of the layout, resources, operations, and transport network that constrain the system's behavior.

4.1.2 Operations and production sequences

In the benchmark, the processes are described through a finite set of elementary operations. The operations are as follows and serve the assembly of the components in jobs shown in Figure 4.2 (Section 4.1.1):

- **Plate loading:** The job enters the production cell via *plate*;
- **Axis mounting:** The *axis* component is mounted as required by the job type;
- **r_comp mounting:** The *r_comp* component is mounted as required by the job type;
- **I_comp mounting:** The *I_comp* component is mounted as required by the job type;
- **L_comp mounting:** The *L_comp* component is mounted as required by the job type;
- **Screw_comp mounting:** The *Screw_comp* component is mounted as required by the job type, typically used to complete or lock part of the assembly depending on the job type;
- **Inspection:** Once all assembly operations are completed, jobs are inspected (when quality issues are modeled in dynamic scenarios); a failed inspection may trigger a recovery operation before unloading;
- **Plate unloading:** After completing its required operation sequence, the finished job leaves the system and the *plate* is unloaded.

These operations define the job progress logic: each job has a predefined sequence of operations to complete. These sequences are listed in Table 4.1

#	B	E	L	T	A	I	P
1	Plate loading	Plate loading	Plate loading	Plate loading	Plate loading	Plate loading	Plate loading
2	Axis mounting	Axis mounting	Axis mounting	Axis mounting	Axis mounting	Axis mounting	Axis mounting
3	Axis mounting	Axis mounting	Axis mounting	Axis mounting	Axis mounting	Axis mounting	Axis mounting
4	Axis mounting	Axis mounting	Axis mounting	r_comp mounting	Axis mounting	I_comp mounting	r_comp mounting
5	r_comp mounting	r_comp mounting	I_comp mounting	L_comp mounting	r_comp mounting	Screw_comp mounting	L_comp mounting
6	r_comp mounting	r_comp mounting	I_comp mounting	Inspection	L_comp mounting	Inspection	Inspection
7	I_comp mounting	L_comp mounting	Screw_comp mounting	Plate unloading	I_comp mounting	Plate unloading	Plate unloading
8	Screw_comp mounting	Inspection	Screw_comp mounting		Screw_comp mounting		
9	Inspection	Plate unloading	Inspection		Inspection		
10	Plate unloading		Plate unloading		Plate unloading		

Table 4.1: Sequence of operations for each job type in the AIP-PRIMECA benchmark.

Figure 4.4 shows an example of the operation sequence for job T, this representation shows how the components are assembled to generate a letter.

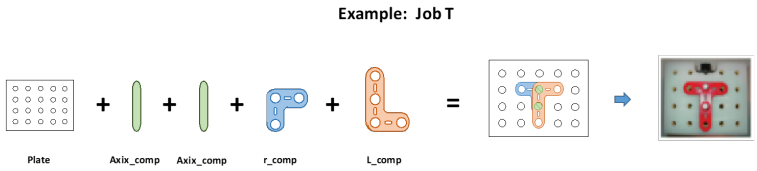

Figure 4.4: Example of the sequence of operations for job T (from the AIP-PRIMECA benchmark) [31].

4.1.3 AIP-PRIMECA Production cell layout

Figure 4.5 shows the layout of the AIP PRIMECA production cell. This FMS production cell aims to recreate a real handling, assembly, and inspection system served by shared resources, aimed at efficiency but also a prelude to interference and congestion. Represented as red rectangles, you can see seven production machines from $M1$ to $M7$, which will be analyzed in detail later. The machines are connected by rails/conveyors on which the jobs travel via shuttles which are rail-guided AGVs, in this case material-handling vehicles whose routing and speed are coordinated. Each conveyor section has a direction of travel represented by the arrows in the figure and is connected to adjacent ones via an intersection, which allows the shuttles to be directed based on their current target workstation.

The layout therefore represents a one-way monorail transport network with transfer gates at the nodes: the unidirectionality and the switching points con-

centrate conflicts in a few hubs, making the emergence of congestion phenomena plausible.

Below is explained how resource sharing is also a risk factor for congestion.

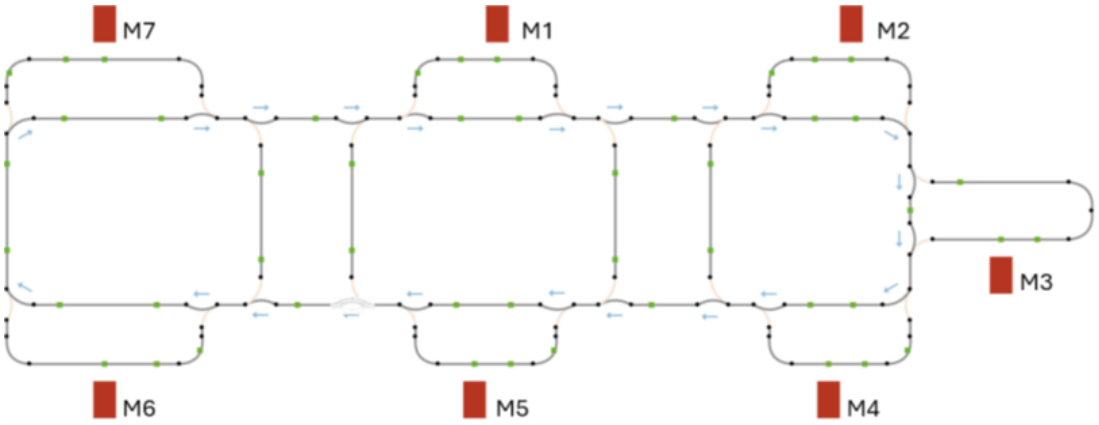

Figure 4.5: Layout AIP PRIMECA Valenciennes [31].

4.1.4 Workstations and machine functions

Each workstation in the production cell has a different function and consequently different operations that can be performed (loading/unloading, assembly, inspection, recovery). Consequently, each machine can perform some of the operations described in Section 4.1.2:

- **M1 (Loading/Unloading)**: system job entry/exit station; manages initial loading and final unloading operations. It performs the operations *Plate loading* and *Plate unloading*.
- **M2, M3, M4 (Assembly)**: assembly stations, each dedicated to a subset of assembly operations envisaged by the benchmark. They perform some of the following the operations: *Axis mounting*, *r_ comp mounting*, *I_ comp mounting*, *L_ comp mounting*, *Screw_ comp mounting*.
- **M5 (Inspection)**: automatic job inspection station, used for benchmark-prescribed quality checks. It performs the operations *Inspection*
- **M6 (Recovery)**: manual recovery/restoration station, used in scenarios with defects or anomalies after inspection.

- **M7 (Optional station)**: optional station activated only in specific scenarios, not necessarily present/necessary in the base scenario. It is often used as a universal machine which can perform almost all the operations.

Figure 4.6 reports an image taken from the AIP-PRIMECA benchmark from [31], showing the different stations/machines of the cell and their respective functions.

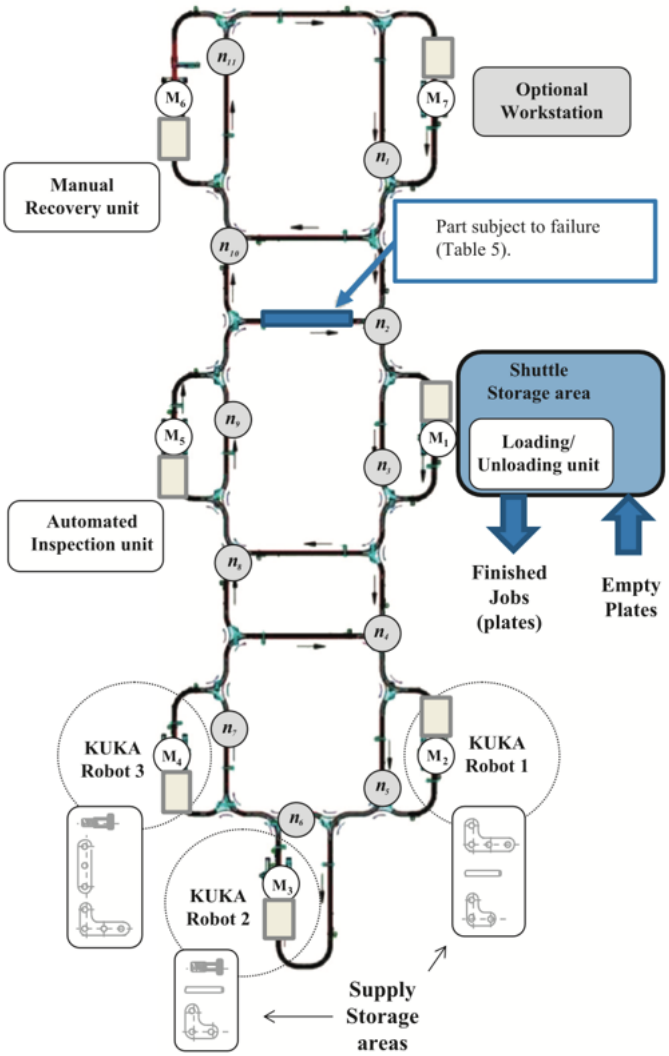

Figure 4.6: Image taken from the AIP-PRIMECA benchmark [31] describing the functions of the different machines/stations in the cell.

Below there are operational details regarding the operations that can be performed by each machine. It is clear that the different operations associated with

the set of machines is one of the factors that makes the system flexible: multiple machines can perform the same operation, giving jobs the freedom to choose between them, especially if the jobs are modeled as agents.

4.1.5 Flexibility and transport network

Each of the operations presented can be performed on one or more machines, therefore there is an operation-machine admissibility relationship that makes the system eligible for the analysis of coordination mechanisms.

In particular, two aspects make the case study suitable for the proposed methodology: the feasibility relationships between operations and machines, which makes the system flexible, and the unique direction of travel in the rail transport network, which makes the system prone to congestion. For this reason, this section focuses on flexibility and the transport network in the case study.

Jobs travel on shuttles, which transport them from one machine to another in the production cell, traveling in a network that can be modeled as a directed graph.

The network consists of forward-facing rails that connect machines and route jobs to them via intersections. Rails, intersections, and machines constitute shared system resources, which can create conflicts and congestion. The benchmark sets a maximum system capacity in terms of shuttles, an experimental parameter that can be fixed or relaxed.

As can be seen in Figure 4.6, there are waiting areas at the workstations for shuttles to perform operations. These areas have limited capacity; an excessive number of waiting shuttles can block traffic in that area and cause congestion and production delays. The benchmark reports travel times in seconds between adjacent nodes in the absence of congestion.

The movement network can be modeled as a weighted directed graph $G = (V, E)$, where:

$$V = \mathcal{N} \cup \mathcal{M}, \qquad \mathcal{N} = \{N1, \ldots, N11\}, \quad \mathcal{M} = \{M1, \ldots, M7\}.$$

The set of directed arcs E is determined by the physical connectivity of the

network (Figure 4.5), that is, an arc $(u, v) \in E$ exists if the network allows direct movement from u to v in the indicated direction. Each arc $(u, v) \in E$ is associated with a travel time $\tau(u, v)$, which constitutes a scenario parameter specified in the model parameterization section (Section 4.2).

The combination of transportation constraints and machine-operation flexibility underlies the emergence of congestion and makes the comparison between online coordination mechanisms meaningful.

4.2 Modeling assumptions and parameterization

Having defined the layout, resources and constraints of the benchmark in Section 4.1.3- 4.1.5, in this paragraph the input data, assumptions and parameterization actually instantiated for the execution of the comparative scenarios in the NetLogo simulations are presented. In the simulations carried out in this study, some modeling choices were introduced that were functional to the objective of comparing online coordination mechanisms.

Table 4.2 shows the correspondence between operations and their associated codes in the simulations. This operation nomenclature has been adopted in the NetLogo code and will be retained for the rest of the discussion.

ID	Operation
O1	Axis mounting
O2	r_comp mounting
O3	I_comp mounting
O4	L_comp mounting
O5	Screw_comp mounting
O6	Inspection
O7	Plate loading
O8	Plate unloading

Table 4.2: Coding of elementary operations adopted in simulations (AIP-PRIMECA).

In the experimental model, production defects are not considered; consequently, the resources dedicated to inspection and recovery ($M5$ and $M6$) are not active and the inspection operation ($O6$) is not included. Therefore, the following are defined as the set of machines M and operations O considered in the experiments:

$$\mathcal{M} = \{M1, M2, M3, M4, M7\}, \qquad \mathcal{O} = \{O1, O2, O3, O4, O5, O7, O8\}.$$

Each operation can be performed by multiple machines, excluding *Plate loading* and *Plate unloading* jobs into the system. This, combined with the different processing times for the same operation on different machines, ensures the system's flexibility. Table 4.3 shows which machines can perform each operation and the corresponding processing times.

Op.	M1	M2	M3	M4	M7
O1	–	271.6	305.4	–	290.5
O2	–	271.6	305.4	–	290.5
O3	–	–	–	271.6	290.5
O4	–	271.6	–	271.6	290.5
O5	–	–	305.4	271.6	–
O7	137.5	–	–	–	–
O8	132.6	–	–	–	–

Table 4.3: Eligibility Operation-machine and processing times

Table 4.3 represents the operation-machine feasibility relationship and processing times: the rows correspond to the operations, the columns to the machines. A numeric value indicates that the operation can be performed by the corresponding machine in that time, while "-" indicates non-executability.

In terms of notation, m_u is the workstation introduced in section 3.2.2 as the machine that can perform the greatest number of operations, in this case corresponds to machine $M7$. It can be noticed from Table 4.3 that $M7$ can perform 4 operations (O1, O2, O3, O4), more than the others which can perform a maximum of 3.

Table 4.4 reports the job sequences derived from the benchmark (Table 4.1) recoded according to the IDs in Table 4.2. Since the inspection operation (O6) is not included in the experimental model, its occurrences have been removed; the resulting sequences are reported in Table 4.4.

Job	Sequence of operations (ID)
B	07-01-01-01-02-02-05-03-08
E	07-01-01-01-02-02-04-08
L	07-01-01-01-05-05-03-03-08
T	07-01-01-02-04-08
A	07-01-01-01-02-04-05-03-08
I	07-01-01-05-03-08
P	07-01-01-02-04-08

Table 4.4: Job operation sequences expressed through the IDs defined in Table 4.2.

These adjustments were made to the benchmark to simulate the behavior of the AIP-PRIMECA production cell in its reproduction in NetLogo. The implementation of the digital twin is explored in Section 4.3.

4.3 Agent-based digital twin implementation (NetLogo–Python)

The analysis of online coordination mechanisms was made possible by developing a digital twin of the AIP-PRIMECA production cell. To reproduce the behavior of the real cell under consideration, a pre-existing NetLogo simulation was used (NetLogo is a multi-agent programmable modeling environment for agent-based simulation; see Section 3.1).

Thanks to the implementation logic of `breeds` and procedures that characterize the NetLogo simulation environment, it was possible to implement the agent-based paradigm.

Starting from the NetLogo reference model for the AIP-PRIMECA simulation, and made the necessary changes, the digital twin was developed in Python, which allowed for the integration of coordination logic and the systematic collection of outputs. This combination of NetLogo and Python is particularly suited to agent-

based modeling. It allows each agent in the system to monitor the environment in real time and act accordingly in an autonomous and coordinated manner through common mechanisms.

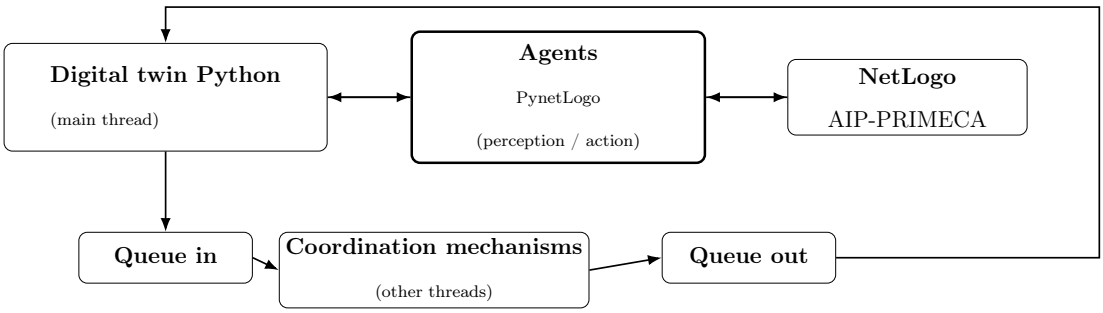

Figure 4.7: Conceptual scheme of the digital twin.

Figure 4.7 is the implemented counterpart of the scheme in Figure 3.1 shown in Section 3.1. It shows the architecture adopted for communication between the Python digital twin and the reproduction of the production cell in NetLogo to separate the simulation level from the control level.

NetLogo simulates the dynamics of the AIP-PRIMECA cell, while the Python Digital Twin orchestrates the experiment, specifically managing initialization, step-by-step progression, state collection, and executing online coordination logic.

The agents act as an interface between the two levels, ensuring the perception of the environment's state and that of each individual agent (machines and jobs) as input for decision-making. Conversely, the actions calculated are applied as commands to the simulator. In the simulation, each mobile agent represents an AGV transporting a job. Communication between Python and NetLogo takes place via the Pynetlogo library in Python.

The components *Queue in / Coordination mechanisms / Queue out* model the asynchronous pipeline described in Section 3.1: the observed state of the agents is queued as a decision request, processed by coordination mechanisms executed in separate threads, and returned to the main thread as a set of actions to be applied in the next step. Consequently, the digital twin performs perception and calculation of coordination mechanisms in parallel with the advancement of the NetLogo simulation, introducing a controlled delay between perception and

action application of the agents.

The Python codes have been transcribed in the appendices: Appendix A (Digital Twin), Appendix B (Agents), and Appendix C (coordination mechanisms).

Now that the case study and the experimental methods have been explored, it only remains to define how the experiments were managed and with which setup parameters they were instantiated.

4.4 Experimental scenario design and simulation setup

The experiments were structured as a comparison between scenarios. As said in Section 3.3, a scenario is defined by the combination of: order set, adopted coordination mechanism and perturbation condition. In this section, the scenarios tested are presented in terms of these three variables.

- **Order set.** Five order sets were considered, identified by the letters produced as jobs (not necessarily in the order of release): `AIPBELT`, `BELTBELTBELT`, `AIPAIPAIPAIP`, `BBBBBBAAAAAA`, `BBBBBBBBBB`.

  Before describing the order sets considered, it is necessary to specify the baseline input with which the digital twin instantiates the nominal plan.

  The digital twin receives a CSV file as input, similar to the one shown in Figure 4.8. This input contains the order set, its routing, and the start and end times of each operation. This data is the output of a hybrid Genetic Algorithm and Monte Carlo approach on FJSP (Flexible Job Shop Problem) with stochastic transportation times.

  This thesis does not delve into the methodologies adopted to achieve offline optimized scheduling. However, within the CoCeDi project, this scheduling represents a result for comparing an offline solution with the online distributed control discussed here.

  The optimized production plan is used in the simulation as a baseline for coordination mechanisms, for the default routing of jobs, and for launching

the products of the order set. This plan is common to all scenarios with the same order set; the coordination mechanisms may deviate from this routing baseline, while the *No rules* scenarios follow it.

type	machines	starts	completions
A	["M1" "M2" "M3" "M2" "M3" "M4" "M4" "M4" "M1"]	[137.5, 317.1, 835.0, 1406.9, 1751.2, 2064.3, 2946.7, 3218.3, 3632.5]	[275.0, 588.7, 1140.4, 1678.5, 2056.6, 2335.9, 3218.3, 3489.9, 3765.1]
B	["M1" "M2" "M3" "M3" "M2" "M7" "M3" "M7" "M1"]	[412.5, 592.1, 1140.4, 1445.8, 1950.1, 2505.1, 2972.8, 3533.3, 3897.7]	[550.0, 863.7, 1445.8, 1751.2, 2221.7, 2795.6, 3278.201, 3823.8, 4030.3]
E	["M1" "M7" "M2" "M3" "M2" "M2" "M4" "M1"]	[825.0, 1798.3, 2221.7, 2667.4, 3075.4, 3347.0, 3654.5, 4068.7]	[962.5, 2088.801, 2493.3, 2972.8, 3347.0, 3618.6, 3926.1, 4201.3]
I	["M1" "M3" "M3" "M4" "M7" "M1"]	[0.0, 224.2, 529.6, 842.7, 1507.8, 3765.1]	[137.5, 529.6, 835.0, 1114.3, 1798.3, 3897.7]
L	["M1" "M7" "M7" "M7" "M4" "M3" "M4" "M7" "M1"]	[275.0, 636.3, 926.8, 1217.3, 1699.7, 2362.0, 2675.1, 3156.8, 3483.3]	[412.5, 926.8, 1217.3, 1507.8, 1971.301, 2667.4, 2946.7, 3447.3, 3615.9]
P	["M1" "M2" "M2" "M3" "M4" "M1"]	[550.0, 863.7, 1678.5, 2056.6, 2369.7, 2783.9]	[687.5, 1135.301, 1950.1, 2362.0, 2641.3, 2916.5]
T	["M1" "M2" "M7" "M2" "M2" "M1"]	[687.5, 1135.301, 2088.801, 2498.101, 2769.701, 3257.101]	[825.0, 1406.9, 2379.3, 2769.701, 3041.3, 3389.701]

Figure 4.8: Example of input for the order set `AIPBELT`.

Figure 4.8 shows an example of an input CSV file for scenarios using AIP-BELT as the order set. Four columns are visible: the first defines the order set, the second specifies the machines on which the operations are performed and consequently the routing, and the third and fourth columns contain the prescribed start and end times for the operations, respectively.

- **Coordination mechanisms.** For each order set, the baseline "*No rules*" scenario corresponding to the absence of reactive logic is tested. Then, the four online coordination mechanisms are tested: *Congestion avoidance*, *Delay triggered reassignment*, *Reservation board*, and *Virtual platooning*.

  The logic calculations take place in the Coordination mechanism block of the code shown in Figure 4.7, and occur every time the agents' perceptions are processed and added to the queue (Queue in). In this case, an interval of 5 NetLogo ticks (the simulation's discrete time steps in NetLogo) was chosen, which corresponds to approximately 1.6 seconds.

  Each pair (order set, mechanism) constitutes a comparison case, to which the perturbation conditions defined in this section are then applied.

- **Perturbations.** Two controlled disturbances were tested:

  - **Perturbation 1 (machine breakdown):** Machine M3 becomes unavailable until the end of production;
  - **Perturbation 2 (unplanned product):** An unscheduled job is reinserted into the system, obtained by restarting the first job of the set.

The perturbation is injected at two relative time points, defined as 20 and 80 percent of the nominal makespan of the same order set under undisturbed and *No rules* conditions. In this way, the temporal severity of the perturbation (early/late) is comparable across different order sets.

Each scenario was run for $R = 30$ independent replications in order to capture the variability due to emergent phenomena, in this case consisting of congestion and coordination mechanisms calculation times.

In total 3750 simulations were performed, which correspond to 5 Order sets, for 4 coordination mechanisms plus "*No rules*", for 5 perturbations (Non-perturbed scenario, Perturbation 1 at 20 percent, Perturbation 1 at 80 percent, Perturbation 2 at 20 percent, Perturbation 2 at 80 percent) all for 30 replications each.

In the next chapter, the results of the aforementioned simulations are presented, using the metrics proposed in Section 3.3.

Chapter 5

Results

This chapter presents the results of the experiments carried out through simulations. First, an analysis of the makespan results and their variability is provided via boxplots; then, the trade-off between the two metrics M and C is analysed through scatter plots; finally, general considerations are drawn in terms of comparison between the different coordination mechanisms.

5.1 Makespan and variability analysis

The makespan results are analysed using boxplots in order to appreciate their distribution across the $R = 30$ replications for each scenario. The boxplots are grouped by order set: each plot represents the distribution of makespan values within the same order set under different coordination mechanisms. The undisturbed conditions are analysed first, followed by Perturbation 1 and Perturbation 2, applied at 20% and 80% of the makespan of the reference scenario without disturbances and without a coordination mechanism.

5.1.1 Baseline makespan (no disturbances)

Figure 5.1 shows the makespan boxplots for the five tested order sets in the undisturbed scenario.

Note that the "*No rules*" scenarios exhibit very low or nearly zero variance; this is because with the same input, the system's behavior does not change. Vari-

ability in makespan comes into play with coordination mechanisms because online decisions are state-dependent: perception–action delay, changes the timing of decisions and the trajectory of the jobs. This way jobs can change the congestion pattern and consequently the makespan differs between iterations.

From Figure 5.1 it is evident that the coordination mechanisms with the largest makespan variability are *Congestion avoidance* and *Reservation board*: these are the two mechanisms that intervene the most on job routing, and the timing of the destination-machine choice affects the routing decision and therefore the transport time.

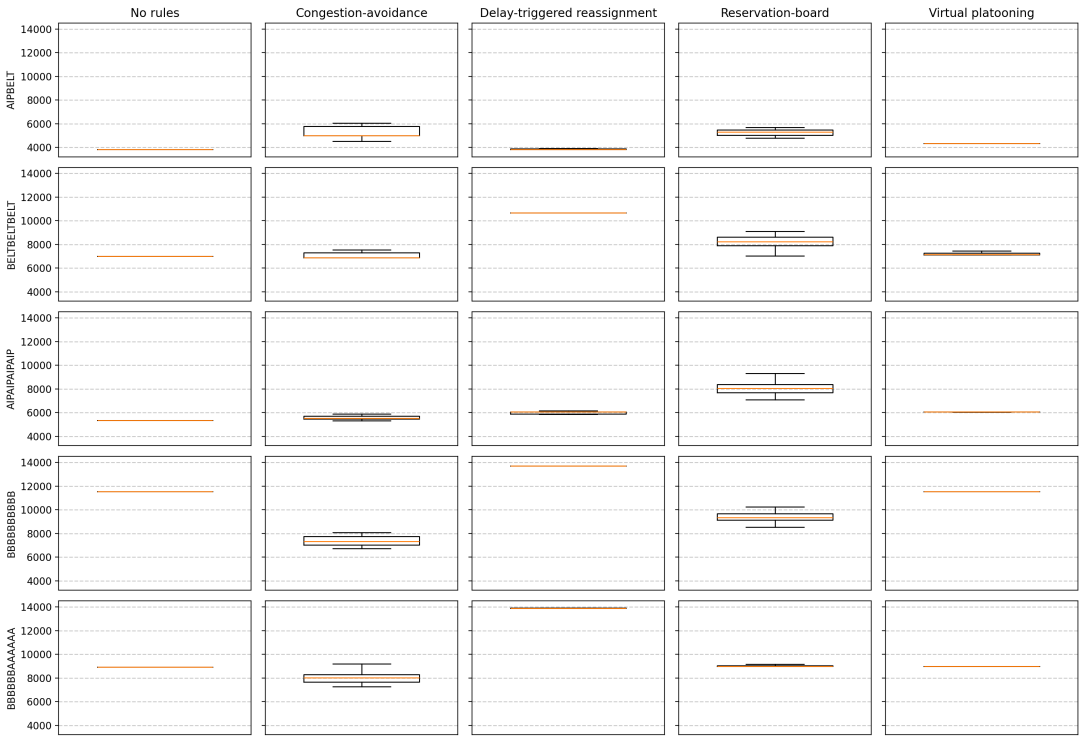

Figure 5.1: Makespan boxplots in the undisturbed scenario for the five tested order sets (rows) and the five coordination mechanisms (columns).

In absence of disturbances, the intervention of some coordination mechanisms can introduce additional waiting times due to unnecessary reassignments. This is the case, for example, of *Reservation board*, which in these scenarios yields a higher average makespan than "*No rules*", except for the order set BBBBBBBBBB. It can also be observed that the effect of coordination mech-

anisms in terms of makespan depends strongly on the order set: an example is *Delay triggered reassignment*, which shows good performance only for small order sets in terms of the number of processed jobs, such as AIPBELT. Finally, *Virtual platooning* shows performance very similar to "*No rules*" for all order sets; this is because this coordination mechanism acts on AGV travel speeds rather than on job routing, and its main goal is congestion reduction rather than execution speed.

5.1.2 Perturbation 1: effect of injection time on makespan

Early injection (20%). Figure 5.2 shows the makespan boxplots for the five tested order sets under Perturbation 1, applied at 20% of the makespan of the reference scenario without disturbances and without coordination mechanisms.

The loss of capacity due to the breakdown leads to an increased makespan in the "*No rules*" scenario for all order sets.

A general improvement in the effectiveness of the coordination mechanisms can also be observed; in particular, *Congestion avoidance* stands out because, by systematically distributing jobs to alternative resources that are less congested than the unavailable $M3$, it achieves an advantage over a random redistribution. *Reservation board* also performs well in terms of job redistribution; however, it generally remains worse than *Congestion avoidance*: reservations reduce conflicts, but they can introduce waiting times and reduced flexibility precisely when rapid adaptation is needed. In complex order sets such as BBBBBBAAAAAA or BBBBBBBBBB (Figure 5.2), while "*No rules*" sees the makespan increase sharply, the rerouting mechanisms manage to contain the degradation by finding alternative paths to functioning machines. *Delay triggered reassignment* worsens its performance, likely due to higher congestion of the AGV fleet around machine $M7$, and *Virtual platooning* remains stable, close to "*No rules*" in terms of makespan, as it is not designed to compensate for a breakdown.

Late injection (80%). Figure 5.3 shows the makespan boxplots for the five tested order sets under Perturbation 1, applied at 80% of the makespan of the reference scenario without disturbances. The impact of the delayed break-

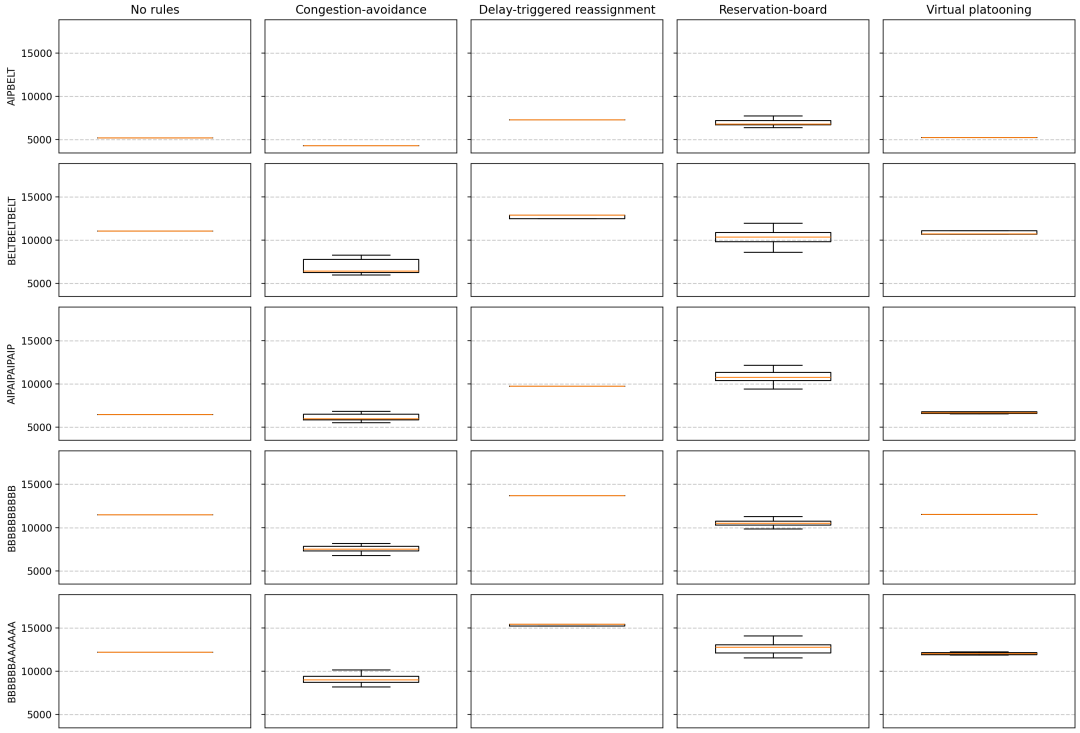

Figure 5.2: Makespan boxplots with Perturbation 1 applied at 20% (early injection) for the five tested order sets (rows) and the five coordination mechanisms (columns).

down is clearly smaller on the makespan in all scenarios. Nevertheless, even though the failure occurs late, the "*No rules*" scenario still shows a visibly higher median makespan than the undisturbed condition (Section 5.1.1), especially in dense order sets such as BBBBBBBAAAAAA or BELTBELTBELT. Coordination mechanisms that modify routing lose some effectiveness compared to early-perturbation scenarios, because a significant portion of the jobs has already been processed when the breakdown occurs, and there is less time for delays to propagate. Here as well, dispersion remains more evident for mechanisms that intervene on routing, and *Virtual platooning* continues to show a makespan similar to "*No rules*", since it acts on transport speeds rather than on routing.

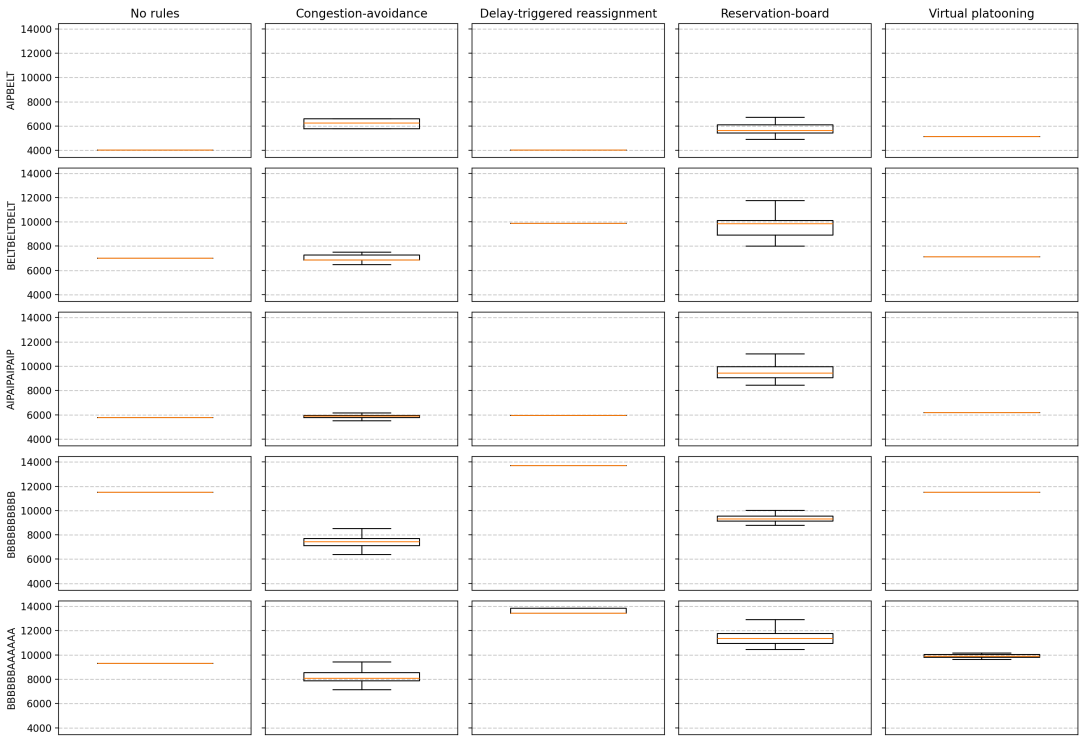

Figure 5.3: Makespan boxplots with Perturbation 1 applied at 80% (late injection) for the five tested order sets (rows) and the five coordination mechanisms (columns).

5.1.3 Perturbation 2: effect of injection time on makespan

Early injection (20%). Figure 5.4 shows the makespan boxplots for the five tested order sets under Perturbation 2, applied at 20% of the makespan of the

reference scenario without disturbances. This perturbation consists of producing one additional job compared to those specified by the order set; in particular, the first released job is released again. This type of perturbation poses a different problem for the coordination mechanisms: no longer avoiding a broken machine, but absorbing an extra job without creating congestion. The unplanned order increases the makespan in the "*No rules*" scenarios, but not to the levels observed with the loss of machine M3. Another difference compared to Perturbation 1 can be seen in the performance of *Congestion avoidance*: with an extra job, it helps only in already dense order sets, where the additional load actually makes congestion emerge. The same holds for *Reservation board*, although with slightly inferior performance. Both show the usual dispersion for the reasons discussed above. The observations for *Delay triggered reassignment* are confirmed, as it is counterproductive in terms of makespan, and for *Virtual platooning*, which does not act on the management of the extra job.

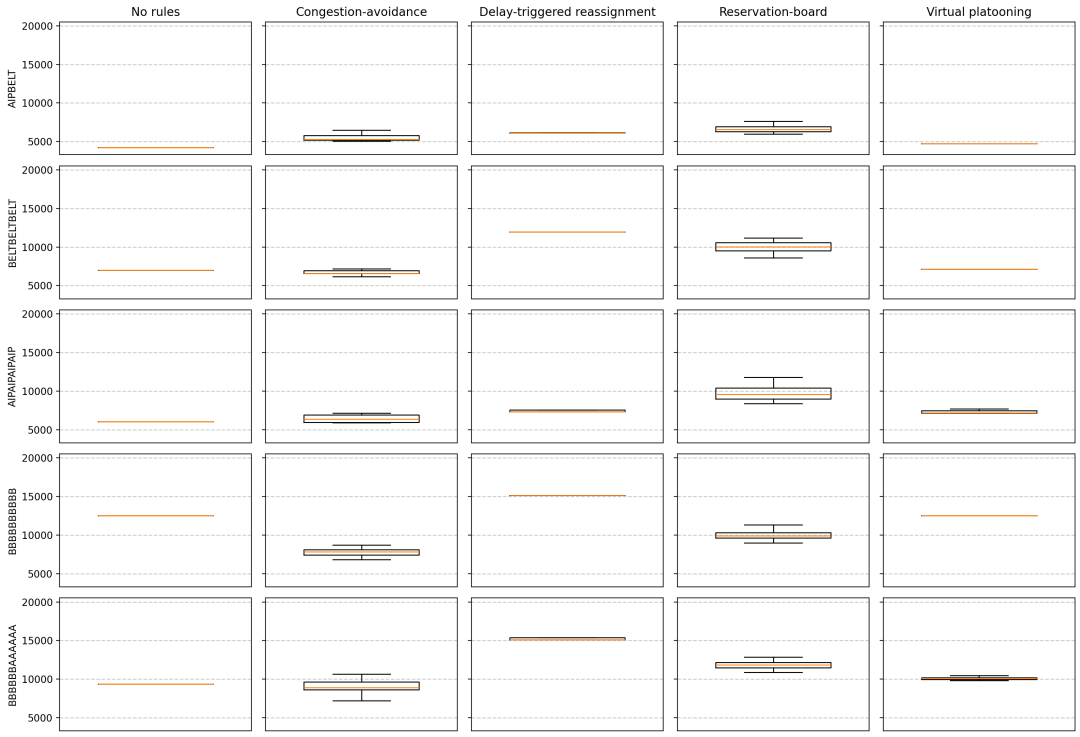

Figure 5.4: Makespan boxplots with Perturbation 2 applied at 20% (early injection) for the five tested order sets (rows) and the five coordination mechanisms (columns).

Late injection (80%). Figure 5.5 shows the makespan boxplots for the five tested order sets under Perturbation 2, applied at 80% of the makespan of the reference scenario without disturbances. Note that, as in the case of Perturbation 1, the extra job released late has less time to generate and propagate congestion and to increase the makespan. As a consequence, the added value of rerouting mechanisms is visible only for order sets that are already congested. A decrease in variability compared to the early-injection case (20%) discussed above can also be observed. This happens because stochastic interactions and resource conflicts are minimal during a "draining" phase of the cell, making congestion less influential on the final completion time. It can be concluded that, in these scenarios, since most of the original jobs are already in completion or unloading, the new product finds the system almost free, essentially behaving like the last element of a single queue.

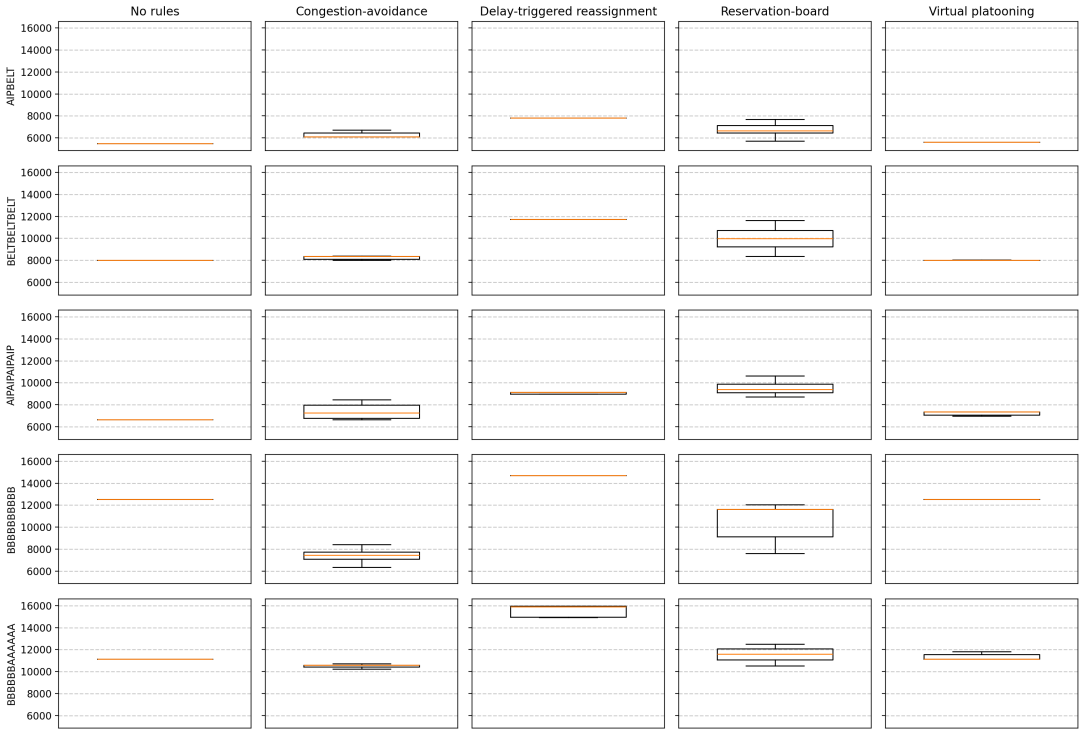

Figure 5.5: Makespan boxplots with Perturbation 2 applied at 80% (late injection) for the five tested order sets (rows) and the five coordination mechanisms (columns).

5.2 Trade-off in the (M, C) space

This section analyses the weighted-sum results in the (M, C) space using scatter plots. Five weight combinations w_m and w_c are considered to visualise the influence of the decision maker's preferences: 50/50, 30/70, 70/30, 20/80, 80/20. The legend used to interpret the scatter plots is reported directly in Figure 5.6 and it is valid for all the graphs in this section. In these plots, the coordination mechanism with the lowest weighted-sum value is highlighted by a black outline.

5.2.1 Baseline trade-off in the (M, C) space (no disturbances)

Figure 5.6 shows the solutions, split by order set and varying the weights, in the (M, C) space under undisturbed conditions. A trade-off emerges: lowering C often requires increasing M, and vice versa, highlighting the strong influence of the weights in selecting the best coordination mechanism. The mechanisms that most frequently emerge as preferred solutions are *Congestion avoidance* and *Reservation board*, especially in scenarios with more jobs; in particular, *Reservation board* is favoured in scenarios where the importance assigned by the weights is skewed towards congestion. It can also be observed that in simpler scenarios such as AIPBELT, *Virtual platooning* emerges as a coordination solution.

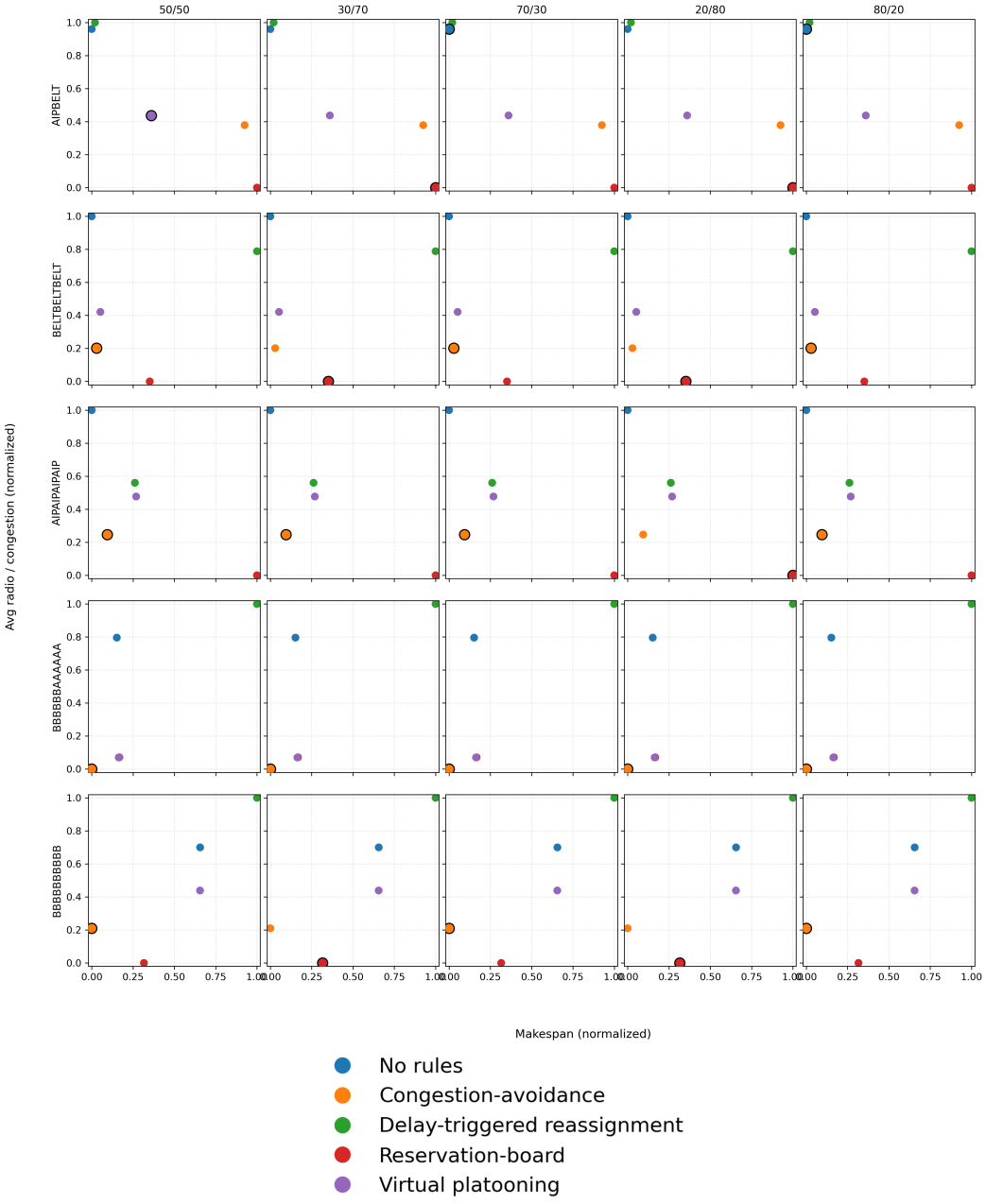

Figure 5.6: Scatter plots in the (M, C) space in the undisturbed scenario for the five tested order sets (rows) and varying weight configurations. The colour code (legend) is shown below; the best mechanism (lowest weighted-sum) is highlighted by a black outline.

5.2.2 Perturbation 1: effect of injection time on the (M, C) trade-off

Early injection (20%). Figure 5.7 shows the scatter plots of the five scenarios under Perturbation 1 at 20%. For four out of five order sets, the choice of the optimal mechanism via the weighted sum concentrates on two alternatives: *Congestion avoidance* as a compromise solution, and *Reservation board* when the weights are skewed towards congestion reduction, especially for the 20/80 combination. Therefore, what was observed in the undisturbed situation is confirmed, with the exception of AIPBELT, where the perturbation makes *Congestion avoidance* dominant, highlighting that, in the presence of failures, rerouting capability becomes more relevant than acting only on transport.

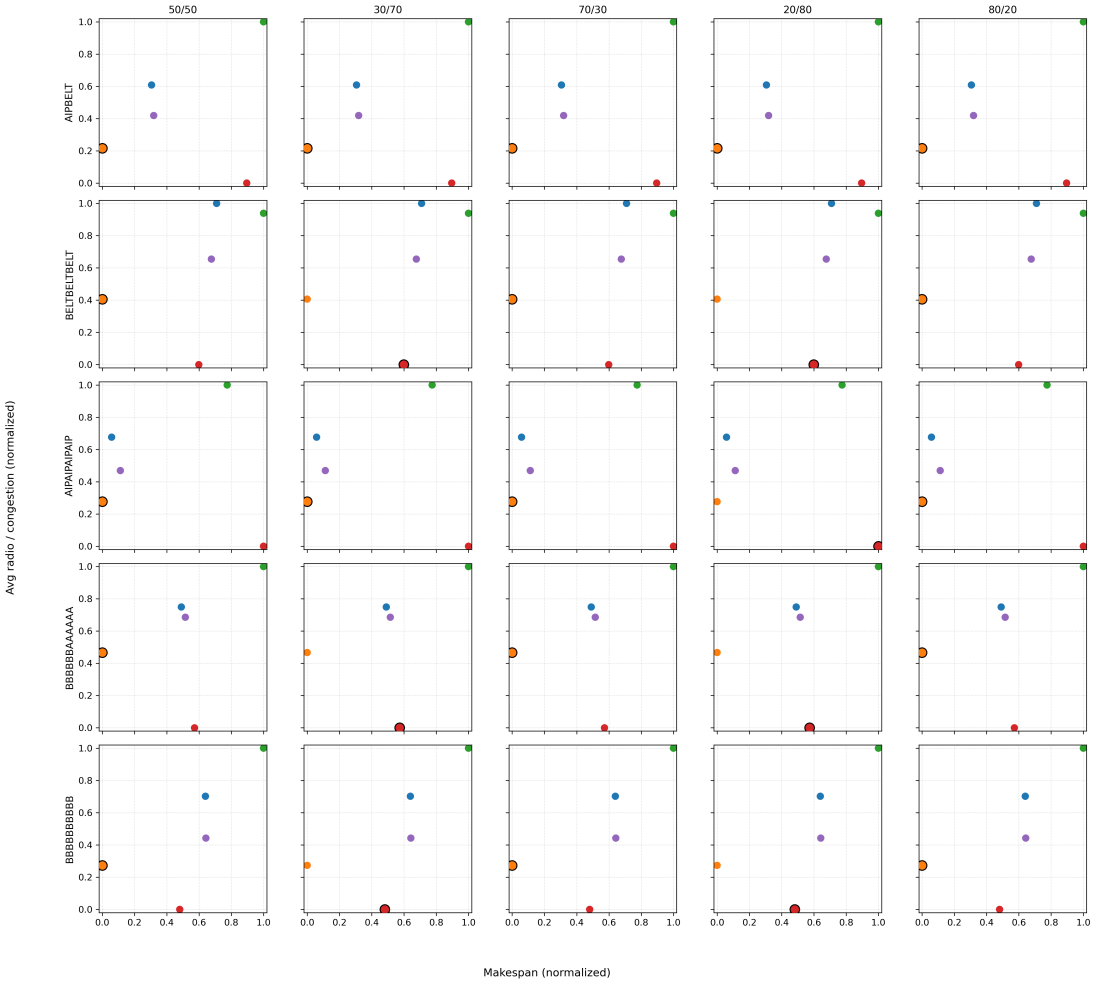

Figure 5.7: Scatter plots in the (M, C) space with Perturbation 1 applied at 20% (early injection) for the five tested order sets (rows) and varying weight configurations.

Late injection (80%). Figure 5.8 reports the scatter plots under Perturbation 1 at 80%. For several order sets, due to the late breakdown, the makespan differences among some mechanisms shrink: the points tend to align towards similar M values. In these cases, congestion becomes the main discriminant. Despite the late failure, *Congestion avoidance* still positions itself as the best compromise. Moreover, *Virtual platooning* still occupies the minimum-congestion region, confirming that the effectiveness of traffic management through platooning is independent of machine availability. This also confirms that, as production approaches completion, online coordination mechanisms have less room to act.

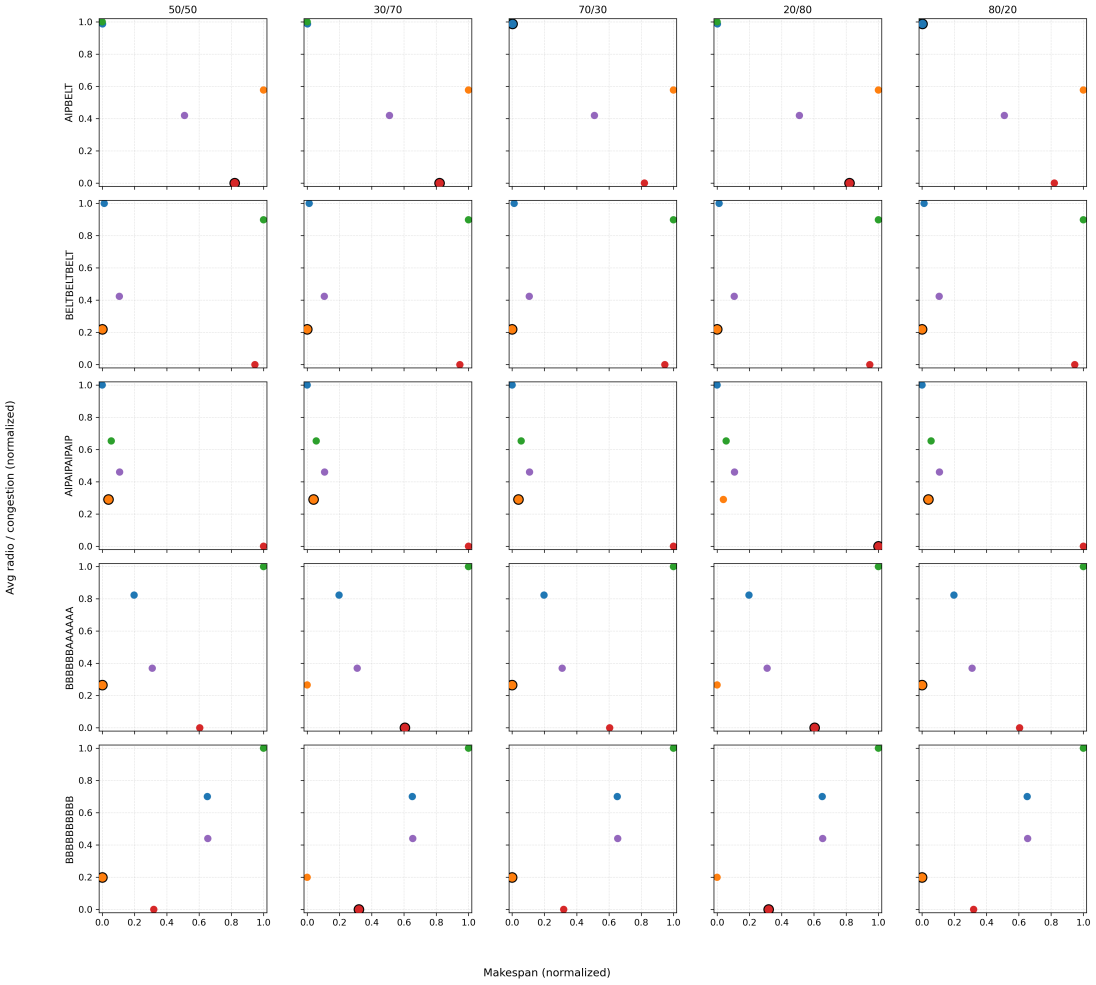

Figure 5.8: Scatter plots in the (M, C) space with Perturbation 1 applied at 80% (late injection) for the five tested order sets (rows) and varying weight configurations.

5.2.3 Perturbation 2: effect of injection time on the (M, C) trade-off

Early injection (20%). Figure 5.9 shows the scatter plots for scenarios with Perturbation 2 at 20% of the reference makespan. Once again, the best solution remains a compromise between *Congestion avoidance* and *Reservation board*. The AIPBELT scenario is the case where the chosen weights truly change the decision: *Virtual platooning* emerges as the winning compromise solution, although *Reservation board* remains dominant if the priority is congestion reduction.

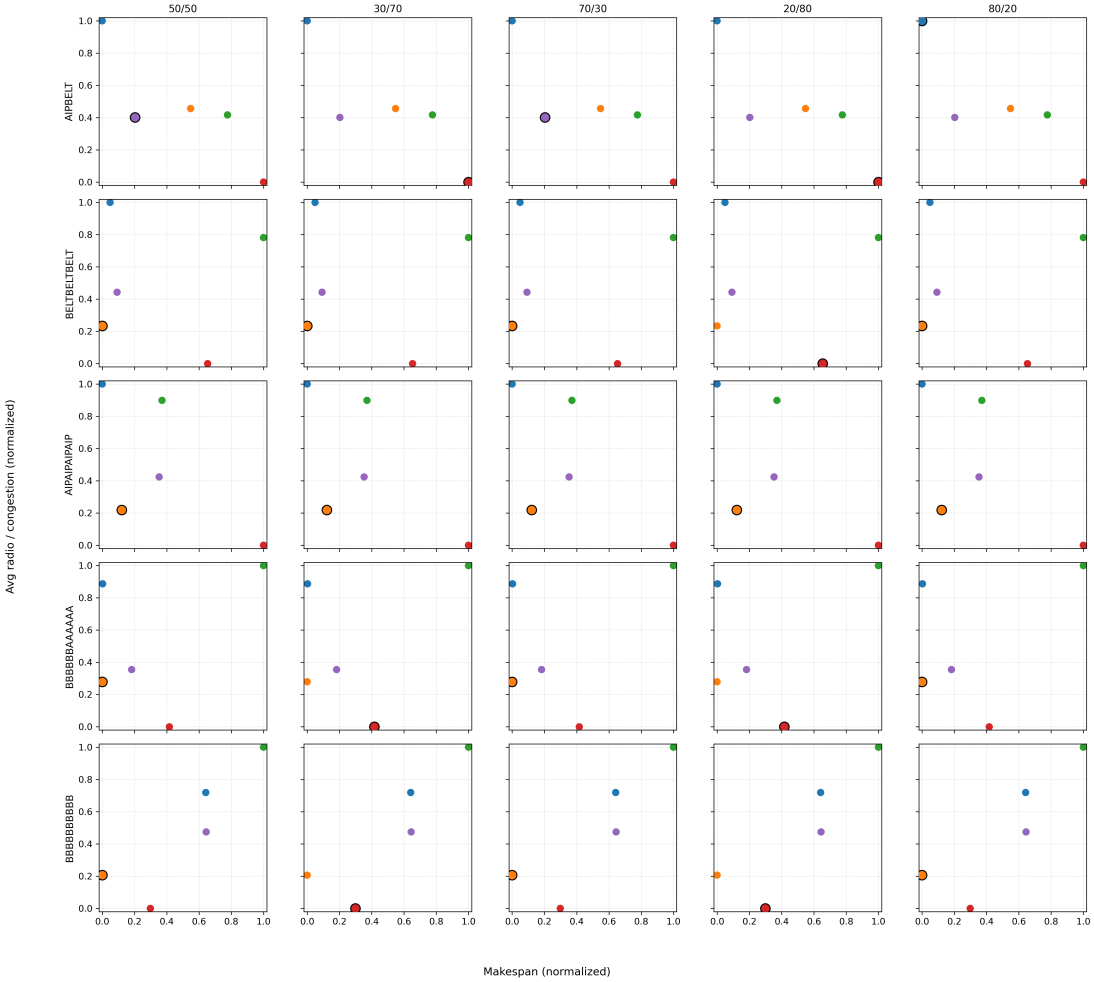

Figure 5.9: Scatter plots in the (M, C) space with Perturbation 2 applied at 20% (early injection) for the five tested order sets (rows) and varying weight configurations.

Late injection (80%). Figure 5.10 shows the scatter plots for scenarios with

Perturbation 2 at 80% of the reference makespan. Coordination mechanisms tend to be distributed more "extremely" in the plots; consequently, in these scenarios the choice of weights has a larger impact on the final solution. As in the other late-perturbation cases, the impact is more limited: adding one element does not trigger the "chain" congestion phenomena that are observed when the system is fully loaded. *Congestion avoidance* and *Reservation board* are confirmed as the two most effective mechanisms in allocating the last job to the machine that is most available at that moment.

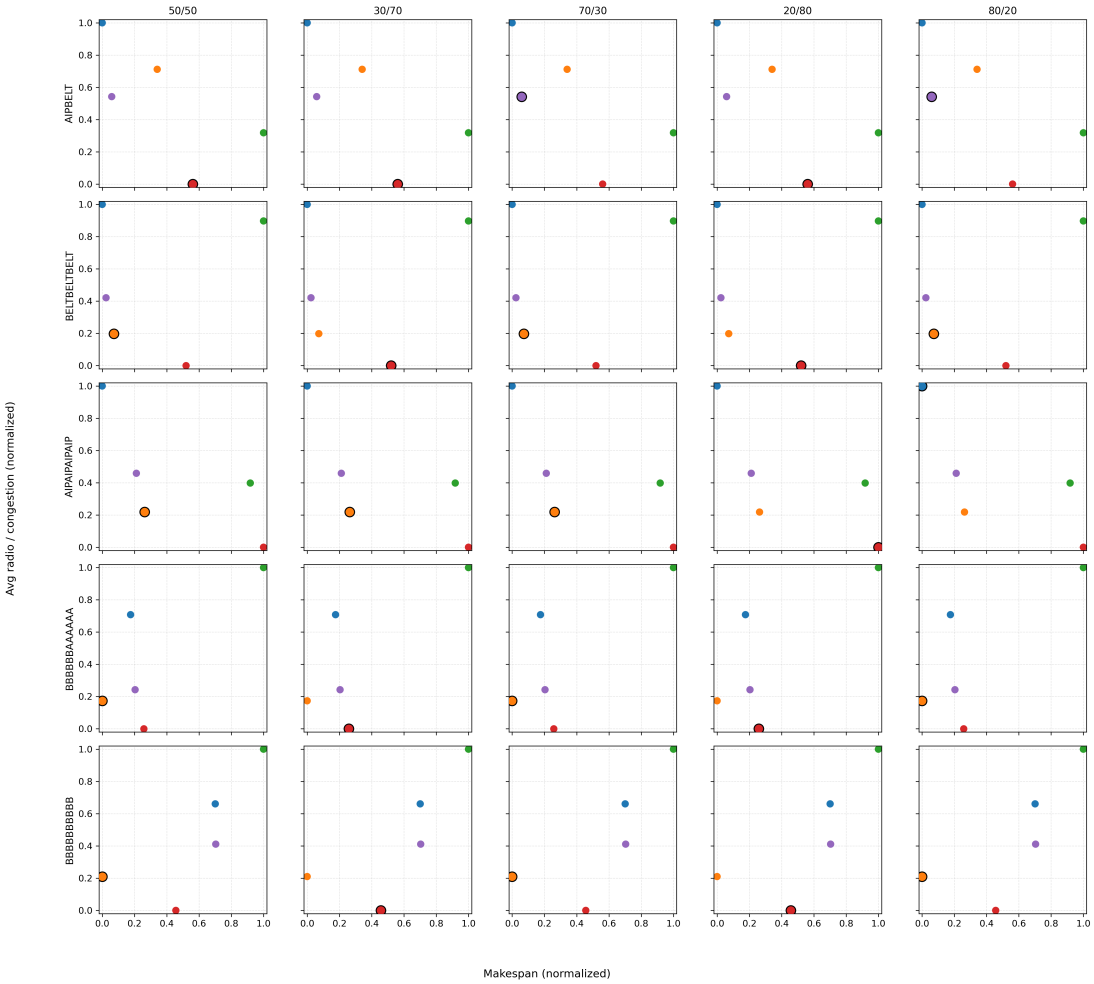

Figure 5.10: Scatter plots in the (M, C) space with Perturbation 2 applied at 80% (late injection) for the five tested order sets (rows) and varying weight configurations.

5.3 Considerations on graphical analysis

The graphical analysis of the results concludes with some general considerations.

The low variance of the makespan highlights the deterministic nature of the simulations, with the exception of the mechanisms that affect routing (*Congestion avoidance* and *Reservation board*). The boxplot and scatterplot analysis highlights the effect of the order set of the coordination mechanisms: the effectiveness of the coordination mechanisms is influenced by the complexity and number of jobs prescribed by the scenario.

At first glance, *Congestion avoidance*, *Reservation board*, and *Virtual platooning* are already identified as the three best coordination mechanisms in most scenarios.

In the next section, the results obtained in terms of the weighted sum proposed in Section 3.3.4 are explored, then some more precise considerations on the coordination mechanisms and their comparison are drawn.

5.4 Indicator results

This section presents the results obtained from the weighted-sum indicator I introduced in Section 3.3.4.

For each scenario (order set $\times$ disturbance condition), for the purpose of a better interpretation, only the coordination mechanism that minimises I is selected as the preferred compromise under the chosen weights.

$\tilde{M}$ and $\tilde{C}$ are values obtained from min-max normalization and then combined into a weighted sum of I. Therefore, the aim is not to compare the values of this indicator in different scenarios, but rather identify the most appropriate coordination mechanism for each of them. Once the coordination mechanisms with the lowest value of I have been identified for each scenario it can be observed, through the occurrences in Table 5.6, which coordination mechanisms are most suitable in multiple different scenarios.

The goal of I is to transform the trade-off between production time and congestion observed in the scatter plots in Section 5.2, into a criterion to support the

decision of choosing the coordination mechanism. Therefore, for each scenario s, the optimal indicator value is considered.

$$I^*(s) = \min I(s),$$

which represents the best compromise given the ratio between the weights w_m and w_c in that scenario.

Tables 5.1–5.5 show, for each order set and disturbance state, the outcome corresponding to $I^*(s)$ and the coordination mechanism with which this result was obtained. In particular, for readability purposes, each coordination mechanism is associated with the same color used in Section 5.2; the legend is reported in Table 5.1.

From the Tables 5.1–5.5 it can be seen that the order sets with the highest number of jobs (AIPAIPAIPAIP, BBBBBBAAAAAAA, BBBBBBBBBBB, BELTBELTBELT) have a stable outcome: the dominant coordination mechanism is *Congestion avoidance* when the decision maker has a greater interest in limiting the makespan or the two criteria are equally important, while *Reservation board* is the most appropriate mechanism in cases where limiting congestion has a greater weight in the decision. In the scenario with the fewest jobs (AIPBELT), *Virtual platooning* also appears as a winning coordination mechanism, especially when more weight is given to the makespan and in 50/50 configurations.

Table 5.6 shows, for each coordination mechanism, how many times it results in a lower value of I for each combination of weights. This provides an aggregate view of the most appropriate coordination mechanisms as the decision maker's preferences vary.

In particular, it is noted that in contexts where makespan is more influential in the decision than congestion (80/20, 70/30) or even when the two criteria have equal weight in the evaluation (50/50), *Congestion avoidance* is the best mechanism with 20 or 21 minimum values of I. However, when congestion becomes the primary decision criterion (20/80, 30/70), *Reservation board* is the best coordination mechanism with 16 and 21 minimum values of I.

The analysis of the values of I shows that when choosing a coordination mech-

Order set	No perturbations	P1 20%	P1 80%	P2 20%	P2 80%
AIPAIPAIPAIP	● 0.171	● 0.138	● 0.164	● 0.170	● 0.241
AIPBELT	● 0.399	● 0.108	● 0.410	● 0.302	● 0.281
BBBBBBAAAAAA	● 0.000	● 0.234	● 0.133	● 0.140	● 0.087
BBBBBBBBBBBB	● 0.105	● 0.137	● 0.100	● 0.103	● 0.105
BELTBELTBELT	● 0.115	● 0.203	● 0.110	● 0.117	● 0.135

- *No rules*
- *Congestion avoidance*
- *Delay triggered reassignment*
- *Reservation board*
- *Virtual platooning*

Table 5.1: Winner mechanism (colored dot) and minimum I (50/50 weights) by order set and perturbation.

anism in a makespan-versus-congestion trade-off context, the most appropriate coordination mechanisms vary based on the decision maker's preferences. Specifically, it is clear that in conditions where production time is the most important criterion, *Congestion avoidance* is the most effective coordination mechanism; in the other cases, *Reservation board* is more suitable. It is also noted that *Virtual platooning* may be a compromise solution to be considered in order-set scenarios with few jobs. In the next section, for a more complete overview, the results in terms of performance degradation following perturbations are analyzed.

Order set	No perturbations	P1 20%	P1 80%	P2 20%	P2 80%
AIPAIPAIPAIP	0.201	0.194	0.215	0.190	0.232
AIPBELT	0.300	0.152	0.246	0.300	0.168
BBBBBBAAAAAA	0.000	0.172	0.181	0.125	0.077
BBBBBBBBBB	0.095	0.144	0.096	0.090	0.137
BELTBELTBELT	0.105	0.180	0.153	0.164	0.156

Table 5.2: Winner mechanism (colored dot) and minimum I (30/70 weights) by order set and perturbation.

Order set	No perturbations	P1 20%	P1 80%	P2 20%	P2 80%
AIPAIPAIPAIP	0.140	0.083	0.114	0.151	0.250
AIPBELT	0.288	0.065	0.298	0.262	0.204
BBBBBBAAAAAA	0.000	0.140	0.080	0.084	0.052
BBBBBBBBBB	0.063	0.082	0.060	0.062	0.063
BELTBELTBELT	0.081	0.122	0.066	0.070	0.110

Table 5.3: Winner mechanism (colored dot) and minimum I (70/30 weights) by order set and perturbation.

Order set	No perturbations	P1 20%	P1 80%	P2 20%	P2 80%
AIPAIPAIPAIP	0.200	0.200	0.200	0.200	0.200
AIPBELT	0.200	0.173	0.164	0.200	0.112
BBBBBBAAAAAA	0.000	0.114	0.121	0.083	0.052
BBBBBBBBBB	0.063	0.096	0.064	0.060	0.091
BELTBELTBELT	0.070	0.120	0.175	0.131	0.104

Table 5.4: Winner mechanism (colored dot) and minimum I (20/80 weights) by order set and perturbation.

Order set	No perturbations	P1 20%	P1 80%	P2 20%	P2 80%
AIPAIPAIPAIP	● 0.125	● 0.055	● 0.089	● 0.141	● 0.200
AIPBELT	● 0.192	● 0.043	● 0.199	● 0.200	● 0.156
BBBBBBAAAAAA	● 0.000	● 0.093	● 0.053	● 0.056	● 0.035
BBBBBBBBBB	● 0.042	● 0.055	● 0.040	● 0.041	● 0.042
BELTBELTBELT	● 0.064	● 0.081	● 0.044	● 0.047	● 0.097

Table 5.5: Winner mechanism (colored dot) and minimum I (80/20 weights) by order set and perturbation.

Mechanism	80/20	70/30	50/50	30/70	20/80
Congestion avoidance	20	21	21	9	4
Delay triggered reassignment	0	0	0	0	0
No rules	4	2	0	0	0
Reservation board	0	0	2	16	21
Virtual platooning	1	2	2	0	0

Table 5.6: Number of wins (minimum I) across all scenarios (5 order sets $\times$ 5 disturbance conditions = 25) for each weight configuration (ω_M/ω_C).

5.5 Analysis of the results

This section summarizes the results shown in the previous sections of Chapter 5, with particular reference to some key findings. The coordination mechanisms were tested on five order sets of varying complexity, with two different perturbations tested at two different simulation times, each of these scenarios for 30 replicates.

What emerges is that the coordination mechanisms that generate the most variability in production time are those that affect job routing the most, while those that affect speed achieve less variable results.

The scatter plots in the space (M, C) of the solutions reveal the trade-off between the two criteria, the dependence of the solution on the order set, perturbation, and preferences in the choice. Therefore, the need for an index I emerges, which captures the trade-off and transforms it into a choice. In particular, *Congestion avoidance* emerged as a suitable coordination mechanism when the two criteria are equally important and when makespan carries more weight in the decision, *Reservation board* if congestion reduction is a primary objective, and *Virtual platooning* for small order sets.

Chapter 6

Conclusions

The purpose of this Thesis was to explore online coordination mechanisms in Flexible Manufacturing systems and provide a method for their comparative evaluation. Specifically, the aim was to study a method that considered not only the execution time of operations but also the fluidity of material flow in the production plant.

To this end, an evaluation framework was developed that included the implementation of the four distinct coordination mechanisms and a method for comparing them. The agent-based paradigm was used to develop the framework, which guaranteed the necessary degree of autonomy for agents to coordinate actions. This framework represents a reproducible agent-based digital twin for online experimentation of coordination mechanisms.

The proposed method for comparative evaluation of coordination mechanisms consists of a weighted sum of makespan and average congestion, two parameters identified as fundamental and complementary by the literature.

The flexibility and congestion tendency appropriate for the case study were provided by the AIP-PRIMECA benchmark, which allowed the experiments to be conducted. The experimental phase was conducted by implementing the proposed framework through a NetLogo-Python interface where Python is used as an orchestration layer that executes coordination logic while the simulation progresses via NetLogo. This architecture allowed for agent-based modeling and their relative autonomy, as well as the collection of KPIs useful in the results

analysis phase through the proposed indicator.

The results were positive, demonstrating the expected trade-off between makespan and congestion, confirming the need for the latter in evaluating online coordination mechanisms in FMS.

The experiments also clarify the benefits of coordination mechanisms depending on the scenario considered: Under undisturbed conditions, mechanisms that intervene on routing introduce the greatest variability, as state-dependent decisions alter job trajectories and thus congestion patterns; in the event of a workstation failure, rerouting is an effective strategy, congestion avoidance, and reservation-based logics provide the largest gains by redirecting distant jobs to available or less congested resources; unscheduled job insertion has a smaller impact on coordination mechanisms' performance, and rerouting is primarily beneficial in the set of already congested orders, while late injections generally have less time to propagate delays and congestion.

Future developments of this work could include an evaluation of the monetary and energy costs of congestion and production times. Other assessments can be made in terms of the digital twin's perception sampling interval and consequences on coordination actions. Another direction could be to extend the absolute comparative evaluation metric to a relative metric between perturbed and unperturbed scenarios to assess the robustness of coordination solutions.

Bibliography

[1] T. Borangiu, S. Răileanu, F. Anton, C. Tahon, T. Berger, D. Trentesaux, "Product-driven manufacturing control with embedded decisional entities," in *Preprints of the 18th IFAC World Congress*, Milano, Italy, Aug. 28–Sep. 2, 2011, pp. 3986–3991.

[2] M. Cantamessa, "Hierarchical and heterarchical behaviour in agent-based manufacturing systems," *Computers in Industry*, vol. 33, no. 3, pp. 265–316, 1997.

[3] S. R. Gonzalez, G. M. Zambrano, and I. F. Mondragon, "Semi-heterarchical architecture to AGV adjustable autonomy within FMSs," *IFAC-PapersOnLine*, vol. 52, no. 10, pp. 7–12, 2019.

[4] A. R. Boccella, P. Centobelli, R. Cerchione, T. Murino, and R. Riedel, "Evaluating centralized and heterarchical control of smart manufacturing systems in the era of Industry 4.0," *Applied Sciences*, vol. 10, no. 3, 755, pp. 1–18, Jan. 2020.

[5] H. Van Brussel, L. Bongaerts, J. Wyns, P. Valckenaers, and T. Van Ginderachter, "A conceptual framework for holonic manufacturing: Identification of manufacturing holons," *Journal of Manufacturing Systems*, vol. 18, no. 1, pp. 35–51, 1999.

[6] P. Valckenaers, H. Van Brussel, J. Wyns, L. Bongaerts, and P. Peeters, "Designing holonic manufacturing systems," *Robotics and Computer-Integrated Manufacturing*, vol. 14, no. 5, pp. 455–464, 1998.

[7] S. Bussmann, "An agent-oriented architecture for holonic manufacturing control," in *Proceedings of the First Open Workshop IMS Europe*, Lausanne, Switzerland, 1998, organized by ESPRIT Working Group on IMS.

[8] K. Fischer, "Agent-based design of holonic manufacturing systems," *Robotics and Autonomous Systems*, vol. 27, no. 1–2, pp. 3–13, 1999.

[9] H. Van Brussel, J. Wyns, P. Valckenaers, L. Bongaerts, and P. Peeters, "Reference architecture for holonic manufacturing systems: PROSA," *Computers in Industry*, vol. 37, no. 3, pp. 255–274, 1998.

[10] S. Komesker, W. Motsch, J. Popper, A. Sidorenko, A. Wagner, and M. Ruskowski, "Enabling a multi-agent system for resilient production flow in modular production systems," *Procedia CIRP*, vol. 107, pp. 991–998, 2022.

[11] A. W. Colombo, R. Schoop, and R. Neubert, "An agent-based intelligent control platform for industrial holonic manufacturing systems," *IEEE Transactions on Industrial Electronics*, vol. 53, no. 1, pp. 322–337, Feb. 2006.

[12] E. D. Lambert, R. Romano, and D. Watling, "Intersection platooning for distributed conflict resolution of an AGV fleet," in *Proceedings of the IEEE 16th International Conference on Automation Science and Engineering (CASE)*, online conference, Aug. 20–21, 2020.

[13] A. Bozzi, J.-F. Jimenez, C. Hernandez-Rodriguez, E.-M. Gonzalez-Neira, and D. Trentesaux, "Platoon-based distributed control for automated material handling systems," in *Proceedings of the 2023 9th International Conference on Control, Decision and Information Technologies (CoDIT)*, Rome, Italy, Jul. 3–6, 2023, pp. 2257–2262.

[14] A. Bozzi, S. Graffione, J.-F. Jimenez, R. Sacile, and E. Zero, "A platoon-based approach for AGV scheduling and trajectory planning in fully automated production systems," *IEEE Transactions on Industrial Informatics*, vol. 21, no. 1, pp. 594–603, Jan. 2025.

[15] J.-H. Lee and H. Hashimoto, "Controlling mobile robots in distributed intelligent sensor network," *IEEE Transactions on Industrial Electronics*, vol. 50, no. 5, pp. 890–902, Oct. 2003.

[16] J. Grosset, A.-J. Fougères, M. Djoko-Kouam, C. Couturier, and J.-M. Bonnin, "Collision and obstacle avoidance for industrial autonomous vehicles – simulation and experimentation based on a cooperative approach," in *Advances in Robotics*, vol. 3, pp. 1–24, 2022.

[17] E. Eroglu Turhanlar, B. Y. Ekren, and T. Lerher, "Autonomous mobile robot travel under deadlock and collision prevention algorithms by agent-based modelling in warehouses," *International Journal of Logistics Research and Applications*, vol. 27, no. 8, pp. 1322–1341, 2024.

[18] A. Grassi, G. Guizzi, L. C. Santillo, and S. Vespoli, "A semi-heterarchical production control architecture for Industry 4.0-based manufacturing systems," *Manufacturing Letters*, vol. 24, pp. 43–46, 2020.

[19] R. H. Bemthuis, M. Koot, M. R. K. Mes, F. A. Bukhsh, M.-E. Iacob, and N. Meratnia, "An agent-based process mining architecture for emergent behavior analysis," in *2019 IEEE 23rd International Enterprise Distributed Object Computing Workshop (EDOCW)*, Paris, France, Oct. 28–31, 2019, pp. 54–63.

[20] S. Lin, A. Liu, J. Wang, and X. Kong, "Development of swarm intelligence Leader-Vicsek model for multi-AGV path planning," in *Proceedings of the 2021 20th International Symposium on Communications and Information Technologies (ISCIT)*, Online Conference, Oct. 20–22, 2021, pp. 49–54.

[21] J. Bergmann, D. Gyulai, and J. Váncza, "Adaptive AGV fleet management in a dynamically changing production environment," *Procedia Manufacturing*, vol. 54, pp. 148–153, 2021.

[22] J. D. Hasbach and M. Bennewitz, "The design of self-organizing human–swarm intelligence," *Adaptive Behavior*, vol. 26, no. 4, pp. 361–386, 2022.

[23] T. Higashi, K. Sekiyama, and T. Fukuda, "Self-organizing control of carrier sequence in AGV transportation system," in *Proceedings of the 2000 IEEE International Conference on Systems, Man, and Cybernetics*, vol. 5, pp. 705–711, 2000.

[24] A. Li, Y. Xie, S. Li, F. Tsung, B. Ding, and Y. Li, "Agent-oriented planning in multi-agent systems," arXiv preprint arXiv:2410.02189, 2025. Available: https://arxiv.org/abs/2410.02189

[25] X. Ye, Z. Deng, Y. Shi, and W. Shen, "Toward energy-efficient routing of multiple AGVs with multi-agent reinforcement learning," *Sensors*, vol. 23, no. 12, 5615, 2023.

[26] O. Antons and J. C. Arlinghaus, "A manufacturing scheduling complexity framework and agent-based comparison of centralized and distributed control approaches," *IEEE Journal of Emerging and Selected Topics in Industrial Electronics*, vol. 3, no. 1, pp. 31–44, Jan. 2022.

[27] J. Chen, Y. Wu, S. Huang, and P. Wang, "Multi-objective optimization for AGV energy efficient scheduling problem with customer satisfaction," *AIMS Mathematics*, vol. 8, no. 9, pp. 20097–20124, 2023.

[28] Y. Hu, H. Yang, and Y. Huang, "Conflict-free scheduling of large-scale multi-load AGVs in material transportation network," *Transportation Research Part E: Logistics and Transportation Review*, vol. 158, 102623, 2022.

[29] A. Bozzi, S. Graffione, J.-F. Jimenez, R. Sacile, and E. Zero, "Reliability evaluation of emergent behaviour in a flexible manufacturing problem," in *2023 IEEE 18th International Conference on System of Systems Engineering (SoSE)*, Florence, Italy, Jun. 2023, pp. 1–6.

[30] M. A. Javed, F. U. Muram, S. Punnekkat, and H. Hansson, "Safe and secure platooning of Automated Guided Vehicles in Industry 4.0," *Journal of Systems Architecture*, vol. 121, 102269, 2021.

[31] D. Trentesaux, C. Pach, A. Bekrar, Y. Sallez, T. Berger, T. Bonte, P. Leitão, and J. Barbosa, "Benchmarking flexible job-shop scheduling and control systems," *Control Engineering Practice*, vol. 21, pp. 1204–1225, 2013, doi: 10.1016/j.conengprac.2013.05.004.

Appendix A

Digital Twin – Code excerpt

```
from Agents import Environment, Machine, Product, read_Production_Order
import threading
import queue
import time

from Coordination_Mechanisms import Congestion_avoidance,
↪  Delay_triggered_reassingment, Reservation_board, Virtual_Platooning


def Digital_twin(netlogo_home, model_path, csv_path, RULE, headless=True,
↪  list_sep=';'):

    env = Environment(netlogo_home, model_path, headless=headless)
    mac = Machine(env.nl)

    Production_Order = read_Production_Order(csv_path)
    env.set_scenario(Production_Order)

    plans_by_type = {po['type']: po for po in Production_Order}

    q_in = queue.Queue()
    q_out = queue.Queue()

    prods = {}
    all_prods = {}
```

```
radio_global_values = []


rule1 = None
if RULE == "NONE":
    rule1 = None
elif RULE == "Congestion_avoidance":
    rule1 = threading.Thread(target=Congestion_avoidance, args=(q_in,
    ↪  q_out))
elif RULE == "Delay_triggered_reassingment":
    rule1 = threading.Thread(target=Delay_triggered_reassingment,
    ↪  args=(q_in, q_out))
elif RULE == "Reservation_board":
    rule1 = threading.Thread(target=Reservation_board, args=(q_in,
    ↪  q_out))
elif RULE == "Virtual_Platooning":
    rule1 = threading.Thread(target=Virtual_Platooning, args=(q_in,
    ↪  q_out))


if rule1 is not None:
    rule1.start()

T = 0

while True:
    env.run()


    while not q_out.empty():
        tag, w, cmd = q_out.get_nowait()
        if w in prods:
            prods[w].action(cmd)
    sim_time, alive = env.perception()

    #Costruction of NEW products
    for w in alive:
```

```
        if w not in prods:
            p = Product(env.nl, w)
            plan = plans_by_type.get(p.type)
            p.planned_machines   = plan['machines']
            p.planned_start      = plan['starts']
            p.planned_completion = plan['completions']
            prods[w] = p
            all_prods[w] = p
            p.send_plan_to_netlogo()


    # Call the rule
    t0 = time.perf_counter()

    if alive and sim_time >= T:
        mac.perception()

        if mac.products_radio:
            radio_global_values.append(max(mac.products_radio))

        for w in alive:
            for w_dead in list(prods.keys()):
                if w_dead not in alive:
                    del prods[w_dead]
            p = prods[w]
            p.perception()
            p.sim_time = sim_time
            delay = time.perf_counter() - t0
            if rule1 is not None:
                q_in.put((p, mac, delay))

        T += 5


    if len(alive) == 0 and sim_time > 100:
        break
    if sim_time > 10000:
        break
```

```
    if rule1 is not None:
        q_in.put((None, None, 0))
        rule1.join()


    if radio_global_values:
        avg_radio_global = sum(radio_global_values) /
        ↪  len(radio_global_values)
        max_radio_global = max(radio_global_values)
    else:
        avg_radio_global = 0.0
        max_radio_global = 0.0

    makespan = env.nl.report("Simulated.Time")
    env.nl.kill_workspace()

    return makespan, max_radio_global, avg_radio_global
```

Appendix B

Agents – Code excerpt

```
import pynetlogo
import csv
import json
import re
import random


def read_Production_Order(filename):
    data = []
    with open(filename, 'r', newline='', encoding='utf-8-sig') as f:
        reader = csv.DictReader(f)
        for row in reader:
            data.append({
                'type':        (row.get('type') or '').strip(),
                'machines':    (row.get('machines') or '').strip(),
                'starts':      json.loads(row.get('starts') or '[]'),
                'completions': json.loads(row.get('completions') or '[]'),
            })
    return data


class Environment:

    def __init__(self, netlogo_home, model_path, headless = True):
```

```
        self.nl = pynetlogo.NetLogoLink(gui=not headless,
        ↪  netlogo_home=netlogo_home)
        self.nl.load_model(model_path)
        seed = random.randint(0, 10**9)
        self.nl.command(f"random-seed {seed}")
        self.nl.command("setup")

    def perception(self):

        sim_time = self.nl.report('Simulated.Time')
        prod_alive = [] if self.nl.report('count products with
        ↪  [product.state != "death"]') == 0 else
        ↪  list(self.nl.report('[who] of products with [product.state !=
        ↪  "death"]'))

        return sim_time, [int(w) for w in prod_alive]

    def set_scenario(self, orders):
        scenario = ''.join(row['type'] for row in orders)
        self.nl.command(f'set Scenario.Name "{scenario}"')

    def run(self):
        self.nl.command("go")

class Product:
    def __init__(self, nl, who):
        self.nl = nl
        self.who = int(who)
        self.x = None
        self.y = None
        self.state = None
        self.type = nl.report(f'[ProductType] of turtle {self.who}')
        self.next_op = None
        #self.operations = list(nl.report(f'[ProductOperations] of turtle
        ↪  {self.who}'))
        ops_raw = nl.report(f'[ProductOperations] of turtle {who}')
        ops = list(ops_raw)
        self.operations = [str(x) for x in ops]
```

```
        self.seq_order = None
        self.planned_machines = []
        self.planned_start = []
        self.planned_completion = []
        self.real_start = None
        self.real_completion = None
        self.machine_planned = []
        self.machine_used = None
        self.heading_ws = None

        self.distance_to_heading = None

        self.speed = None


    def perception(self):
        nl = self.nl
        w = self.who

        x, y, seq_order, speed = nl.report(
            f'(list '
            f'[xcor] of turtle {w} '
            f'[ycor] of turtle {w} '
            f'[CurrentSequenceOrder] of turtle {w} '
            f'[speed-factor] of turtle {w})'
        )

        state, next_op, heading_ws = nl.report(
            f'(list '
            f'(word [Product.State] of turtle {w}) '
            f'([Next.Product.Operation] of turtle {w}) '
            f'([Heading.Workstation] of turtle {w}))'
        )

        self.x = x
        self.y = y
```

```
        self.seq_order = int(seq_order) if seq_order is not None else None
        self.speed = float(speed) if speed is not None else None
        self.state = state
        self.next_op = next_op
        self.heading_ws = heading_ws


        n = int(nl.report(f'length [ProductRealStart] of turtle {w}'))
        self.real_start = [] if n == 0 else
        ↪  list(nl.report(f'[ProductRealStart] of turtle {w}'))

        n = int(nl.report(f'length [ProductRealCompletion] of turtle {w}'))
        self.real_completion = [] if n == 0 else
        ↪  list(nl.report(f'[ProductRealCompletion] of turtle {w}'))

        n = int(nl.report(f'length [Machine.Used] of turtle {w}'))
        self.machine_used = [] if n == 0 else
        ↪  list(nl.report(f'[Machine.Used] of turtle {w}'))


        self.distance_to_heading =
        ↪  float(nl.report(f'[my-distance-to-heading-ws] of turtle {w}'))


        return self

    def action (self, cmd):
        self.nl.command(f'ask product {int(self.who)} [ {cmd} ]')


    def send_plan_to_netlogo(self):
        ml = self.planned_machines if self.planned_machines else '[]'

        self.nl.command(
            f'ask product {self.who} [ '
            f'  set Machine.Planned {ml} '
            f']'
        )
```

```
        self.machine_planned = re.findall(r'"([^"]+)"', ml)




class Machine:
    def __init__(self, nl):
        self.nl = nl

        self.who = [int(x) for x in nl.report(
            'map [ m -> [who] of m ] sort-on [who] machines'
        )]

        self.x = [float(nl.report(f'[xcor] of machine {w}')) for w in
        ↪  self.who]
        self.y = [float(nl.report(f'[ycor] of machine {w}')) for w in
        ↪  self.who]

        self.name = [str(nl.report(f'[Machine.Name] of machine {w}')) for w
        ↪  in self.who]
        self.operations_type = [list(nl.report(f'[Machine.Operations.Type]
        ↪  of machine {w}')) for w in self.who]
        self.operations_time = [list(nl.report(f'[Machine.Operations.Time]
        ↪  of machine {w}')) for w in self.who]

        n = len(self.who)
        self.state           = [None] * n
        self.next_completion = [0.0]  * n
        self.products_radio  = [0.0]  * n

    def perception(self):
        self.state = [self.nl.report(f'[Machine.State] of machine {w}') for
        ↪  w in self.who]
        self.next_completion = [float(self.nl.report(f'[Next.Completion] of
        ↪  machine {w}')) for w in self.who]
```

```
        self.products_radio  =
        ↪  [float(self.nl.report(f'[Number.Products.Radio] of machine
        ↪  {w}')) for w in self.who]
        return self
```

Appendix C

Coordination mechanisms – Code excerpts

C.1 Congestion avoidance

```
def Congestion_avoidance(q_in, q_out):
    D_min_M2 = 50.0
    D_min_M3 = 90.0
    D_min_M4 = 45.0
    D_min_M7 = 45.0

    while True:
        p, mac, d = q_in.get()
        if p is None:
            break

        if p.heading_ws == "M1":
            continue

        if p.heading_ws == "M2" and p.distance_to_heading < D_min_M2:
            continue
        if p.heading_ws == "M3" and p.distance_to_heading < D_min_M3:
            continue
        if p.heading_ws == "M4" and p.distance_to_heading < D_min_M4:
            continue
```

```
        if p.heading_ws == "M7" and p.distance_to_heading < D_min_M7:
            continue

        i_h = mac.name.index(p.heading_ws)

        best_name = None
        best_radio  = mac.products_radio[i_h]

        for i, name in enumerate(mac.name):
            if i == i_h:
                continue
            if p.next_op in mac.operations_type[i]:
                r = mac.products_radio[i]
                if r < best_radio:
                    best_radio = r
                    best_name = name

        if best_name is not None:
            time.sleep(d)
            q_out.put(('prod', p.who, f'set Heading.Workstation
            ↪  "{best_name}"'))
```

C.2 Delay triggered reassignment

```
def Delay_triggered_reassignment(q_in, q_out):
    while True:
        p, _, d = q_in.get()
        if p is None:
            break
        if p.heading_ws == "M1":
            continue
        if p.next_op == "O8":
            q_out.put(('prod', p.who, 'set Heading.Workstation "M1"'))
            continue

        i = int(p.seq_order) - 1
```

```
        if i < 0:
            continue
        if i < len(p.real_completion) and i < len(p.planned_completion):
            if p.real_completion[i] > p.planned_completion[i]:
                time.sleep(d)
                q_out.put(('prod', p.who, 'set color red'))
                q_out.put(('prod', p.who, 'set Heading.Workstation "M7"'))
            else:
                q_out.put(('prod', p.who, 'set color 98'))
```

C.3 Reservation board

```
def Reservation_board(q_in, q_out):

    reservations = {m: None for m in ("M1", "M2", "M3", "M4", "M5", "M6",
    ↪  "M7")}

    while True:
        p, mac, d = q_in.get()
        if p is None:
            break

        j = int(p.seq_order) - 1

        completed = len(p.real_completion) if p.real_completion else 0
        for m, info in list(reservations.items()):
            if info and info['who'] == p.who:
                if (completed > info['op_index']):
                    reservations[m] = None

        if p.heading_ws == "M1":
            continue

        m = p.heading_ws

        if m in reservations:
            r = reservations[m]
```

```
            if (r is None) or (r['who'] == p.who):
                reservations[m] = {'who': p.who, 'op_index': j}
                continue

        candidates = []
        for i, nm in enumerate(mac.name):
            if nm == m:
                continue
            if nm not in reservations:
                continue
            if p.next_op in mac.operations_type[i]:
                if reservations[nm] is None:
                    candidates.append(i)

        if candidates:
            best_i = candidates[0]
            nm = mac.name[best_i]
            reservations[nm] = {'who': p.who, 'op_index': j}
            time.sleep(d)
            q_out.put(('prod', p.who, 'set color 98'))
            q_out.put(('prod', p.who, f'set Heading.Workstation "{nm}"'))
            q_out.put(('prod', p.who, 'set priority 1000'))
        else:
            time.sleep(d)
            q_out.put(('prod', p.who, 'set Heading.Workstation ""'))
            q_out.put(('prod', p.who, 'set priority 0'))
            q_out.put(('prod', p.who, 'set color red'))
```

C.4 Virtual platooning

```
def Virtual_Platooning(q_in, q_out, SF_min=0.1, SF_max=1.0):
    groups = {}
    base_by_heading =
    ↪ {"M1":15,"M2":25,"M3":35,"M4":45,"M5":55,"M6":85,"M7":105}
    K = 1.0 / 0.31
```

```
def clamp_sf(x):
    try:
        x = float(x)
    except:
        return SF_min
    if not math.isfinite(x):
        return SF_max if x > 0 else SF_min
    if x < SF_min:
        return SF_min
    if x > SF_max:
        return SF_max
    return x


def t_op_leader(L, mac):
    if L.heading_ws not in mac.name:
        return 0.0
    i_m = mac.name.index(L.heading_ws)

    types = list(map(str, mac.operations_type[i_m]))
    times = [float(t) for t in mac.operations_time[i_m]]
    t_by_code = {t: tm for t, tm in zip(types, times)}

    pos_next = int(L.seq_order) if L.seq_order is not None else 0

    rem_current = 0.0
    if str(mac.state[i_m]) == "Machine.Processing":
        rem_current = max(float(mac.next_completion[i_m]) -
        ↪  float(getattr(L, "sim_time", 0.0)), 0.0)

    total_future = 0.0
    nops = min(len(L.operations), len(L.machine_planned))
    for pos in range(pos_next, nops):
        if L.machine_planned[pos] != L.heading_ws:
            break
        code = str(L.operations[pos])
        total_future += float(t_by_code.get(code, 0.0))
```

```
    return rem_current + total_future


def t_op_here(prod, mac):
    if prod.heading_ws not in mac.name:
        return 0.0
    i_m = mac.name.index(prod.heading_ws)
    types = list(map(str, mac.operations_type[i_m]))
    times = [float(t) for t in mac.operations_time[i_m]]
    t_by_code = {t: tm for t, tm in zip(types, times)}


    pos0 = int(prod.seq_order) if prod.seq_order is not None else 0
    total = 0.0
    for pos in range(pos0, min(len(prod.operations),
    ↪  len(prod.machine_planned))):
        if prod.machine_planned[pos] != prod.heading_ws:
            break
        code = str(prod.operations[pos])
        total += float(t_by_code.get(code, 0.0))
    return total

while True:
    p, mac, d = q_in.get()
    if p is None:
        break
    if p.state == "death":
        continue

    groups.setdefault(p.heading_ws, [])
    if p.who not in [prod.who for prod in groups[p.heading_ws]]:
        groups[p.heading_ws].append(p)

    h = p.heading_ws
    groups[h] = [pr for pr in groups[h] if (pr.heading_ws == h and
    ↪  getattr(pr, "sim_time", None) == getattr(p, "sim_time", None)
    ↪  and str(getattr(pr, "state", "")) != "death")]
```

```
if  (p.heading_ws in mac.name):
    platoon = sorted(groups[p.heading_ws],
                             key=lambda prod:
                             ↪  prod.distance_to_heading)
    for i, prod in enumerate(platoon):
        color_intensity = base_by_heading.get(prod.heading_ws, 95)
        q_out.put(('prod', prod.who, f'set color
        ↪  {color_intensity}'))
        if i == 0:
         q_out.put(('prod', prod.who, f'set speed-factor {SF_max}'))

        if i == 1:
            t_arr_L_ticks = platoon[0].distance_to_heading / (SF_max
            ↪  * 0.3)
            t_op_L_ticks  = (t_op_leader(platoon[0], mac) or 0.0) *
            ↪  K
            denom = 0.19 * (t_arr_L_ticks + t_op_L_ticks)
            SF1 = SF_max if denom <= 0 else
            ↪  (prod.distance_to_heading / denom)
            SF1 = clamp_sf(SF1)
            q_out.put(('prod', prod.who, f'set speed-factor {SF1}'))

        if i == 2:
            t_arr_F1_ticks = (platoon[0].distance_to_heading /
            ↪  (SF_max * 0.3)) \
                             + (t_op_leader(platoon[0], mac) or 0.0)
                             ↪  * K
            t_op_F1_ticks  = (t_op_here(platoon[1], mac)  or 0.0) *
            ↪  K
            denom = 0.19 * (t_arr_F1_ticks + t_op_F1_ticks)
            SF2 = SF_max if denom <= 0 else
            ↪  (prod.distance_to_heading / denom)
            SF2 = clamp_sf(SF2)
            q_out.put(('prod', prod.who, f'set speed-factor {SF2}'))

        if i == 3:
            t_arr_F2_ticks = t_arr_F1_ticks + t_op_F1_ticks
```

```
        t_op_F2_ticks  = (t_op_here(platoon[2], mac) or 0.0) * K
        denom = 0.19 * (t_arr_F2_ticks + t_op_F2_ticks)
        SF3 = SF_max if denom <= 0 else
        ↪  (prod.distance_to_heading / denom)
        SF3 = clamp_sf(SF3)
        q_out.put(('prod', prod.who, f'set speed-factor {SF3}'))

    if i == 4:
        t_arr_F3_ticks = t_arr_F2_ticks + t_op_F2_ticks
        t_op_F3_ticks  = (t_op_here(platoon[3], mac) or 0.0) * K
        denom = 0.19 * (t_arr_F3_ticks + t_op_F3_ticks)
        SF4 = SF_max if denom <= 0 else
        ↪  (prod.distance_to_heading / denom)
        SF4 = clamp_sf(SF4)
        q_out.put(('prod', prod.who, f'set speed-factor {SF4}'))

    if i == 5:
        t_arr_F4_ticks = t_arr_F3_ticks + t_op_F3_ticks
        t_op_F4_ticks  = (t_op_here(platoon[4], mac) or 0.0) * K
        denom = 0.19 * (t_arr_F4_ticks + t_op_F4_ticks)
        SF5 = SF_max if denom <= 0 else
        ↪  (prod.distance_to_heading / denom)
        SF5 = clamp_sf(SF5)
        q_out.put(('prod', prod.who, f'set speed-factor {SF5}'))

    if i == 6:
        t_arr_F5_ticks = t_arr_F4_ticks + t_op_F4_ticks
        t_op_F5_ticks  = (t_op_here(platoon[5], mac) or 0.0) * K
        denom = 0.19 * (t_arr_F5_ticks + t_op_F5_ticks)
        SF6 = SF_max if denom <= 0 else
        ↪  (prod.distance_to_heading / denom)
        SF6 = clamp_sf(SF6)
        q_out.put(('prod', prod.who, f'set speed-factor {SF6}'))

    if i == 7:
        t_arr_F6_ticks = t_arr_F5_ticks + t_op_F5_ticks
        t_op_F6_ticks  = (t_op_here(platoon[6], mac) or 0.0) * K
        denom = 0.19 * (t_arr_F6_ticks + t_op_F6_ticks)
```

```
                SF7 = SF_max if denom <= 0 else
                ↪  (prod.distance_to_heading / denom)
                SF7 = clamp_sf(SF7)
                q_out.put(('prod', prod.who, f'set speed-factor {SF7}'))

            if i == 8:
                t_arr_F7_ticks = t_arr_F6_ticks + t_op_F6_ticks
                t_op_F7_ticks  = (t_op_here(platoon[7], mac) or 0.0) * K
                denom = 0.19 * (t_arr_F7_ticks + t_op_F7_ticks)
                SF8 = SF_max if denom <= 0 else
                ↪  (prod.distance_to_heading / denom)
                SF8 = clamp_sf(SF8)
                q_out.put(('prod', prod.who, f'set speed-factor {SF8}'))

            if i == 9:
                t_arr_F8_ticks = t_arr_F7_ticks + t_op_F7_ticks
                t_op_F8_ticks  = (t_op_here(platoon[8], mac) or 0.0) * K
                denom = 0.19 * (t_arr_F8_ticks + t_op_F8_ticks)
                SF9 = SF_max if denom <= 0 else
                ↪  (prod.distance_to_heading / denom)
                SF9 = clamp_sf(SF9)
                q_out.put(('prod', prod.who, f'set speed-factor {SF9}'))

            if i == 10:
                t_arr_F9_ticks = t_arr_F8_ticks + t_op_F8_ticks
                t_op_F9_ticks  = (t_op_here(platoon[9], mac) or 0.0) * K
                denom = 0.19 * (t_arr_F9_ticks + t_op_F9_ticks)
                SF10 = SF_max if denom <= 0 else
                ↪  (prod.distance_to_heading / denom)
                SF10 = clamp_sf(SF10)
                q_out.put(('prod', prod.who, f'set speed-factor
                ↪  {SF10}'))
```