Heisenpub

Multi-Agent Reinforcement Learning Accelerated by Swarm Heuristics for Distributed Control in Smart Manufacturing Lines

Samir Benyahia[◇,]

[◇] Université Saoura de Technologies, Department of Computer Science and Engineering, Boulevard El-Moudjahidine, Béchar 08000, Algeria

Yacine Kherouf[,]

Institut Maghrébin dInformatique Avancée, Department of Computer Science and Engineering, Avenue Abdelkader Benkhedda, Tlemcen 13000, Algeria

Abstract. Smart manufacturing lines integrate heterogeneous machines, conveyors, buffers, and transport systems that must operate under stringent productivity, quality, and energy constraints. Conventional centralized control architectures encounter scalability limitations when the line grows in size or configuration changes become frequent. Distributed decision making based on multi-agent reinforcement learning is one possible direction, but pure data-driven learning often suffers from slow convergence, unstable exploration, and difficulty in exploiting structural properties of manufacturing processes. This work discusses a distributed control framework where local controllers are modeled as learning agents whose policies are optimized through multi-agent reinforcement learning while their exploration is guided by swarm-based metaheuristics. The swarm layer searches over policy parameters, coordination signals, and shaping rewards, and injects structured perturbations into the learning process in order to accelerate the discovery of high-performing joint behaviors. The manufacturing line is represented as a network of stations, buffers, and routing elements with local observations and shared global performance indicators. The study outlines a linear state-space abstraction of the line dynamics, formulates the agents' interaction as a cooperative game, and integrates swarm heuristics with policy gradient and value-based methods. Numerical experiments on synthetic smart line configurations illustrate how the combined scheme may affect throughput, work-in-process, tardiness, and energy usage under varying demand and disturbance patterns. The discussion highlights practical design choices and limitations when embedding swarm heuristics into multi-agent reinforcement learning for distributed control in reconfigurable manufacturing environments.

1. Introduction

Smart manufacturing lines combine sensing, computation, and actuation to support flexible, demand-driven production with short changeover times and fine-grained traceability [1]. Compared with traditional lines, these systems typically feature higher variability in

product mix, smaller batch sizes, and richer possibilities for routing and resource sharing. Decision making in such environments is distributed over multiple levels, including low-level motion control, cell-level dispatching, buffer management, maintenance scheduling, and energy-aware operation. When the number of controllable assets grows, centralized control schemes tend to suffer from combinatorial explosion, communication bottlenecks, and single points of failure. Distributed or semi-distributed control architectures, where local controllers coordinate via limited information exchange, are therefore considered in many implementations of reconfigurable manufacturing lines [2] [3].

Multi-agent reinforcement learning offers a way to derive control policies from interaction data without explicit enumeration of all scenarios. In this setting, each controller or group of machines can be modeled as an agent that learns a policy mapping observed states to actions based on reward signals derived from line performance metrics. Agents interact in a common environment, and their actions collectively influence throughput, work-in-process levels, lead times, and equipment utilization. However, multi-agent reinforcement learning in high-dimensional discrete or hybrid action spaces encounters challenges such as non-stationarity, sparse or delayed rewards, large state spaces, and sensitivity to hyperparameters. These issues can lead to slow convergence, oscillatory behavior, or policies that overfit to specific demand profiles or disturbance realizations [4] [5].

Swarm heuristics such as particle swarm optimization, ant-colony style path search, or population-based search algorithms have been explored in various scheduling and routing problems because they provide simple mechanisms to explore large combinatorial spaces using distributed local rules and information sharing. In many such heuristics, a population of candidate solutions evolves by combining stochastic search with collective memory or communication. These methods are often straightforward to adapt to changing environments, and their explicit search behavior can be monitored and constrained. However, swarm heuristics typically operate on static or episodic problem formulations, and they do not learn state-dependent feedback policies in the same way as reinforcement learning methods [6].

The combination of multi-agent reinforcement learning with swarm heuristics aims to exploit complementary properties of both paradigms. Reinforcement learning provides a framework to learn state-feedback policies, while swarm heuristics can be used as an additional search process over policies, exploration strategies, or coordination signals. In a smart manufacturing line, the two layers can be integrated so that swarm-based search proposes promising regions in the policy parameter space or in the space of structured joint actions, while agents refine these suggestions through gradient-based or temporal-difference learning. The coordination between the two layers must be designed carefully to avoid destabilizing the learning process or violating operational constraints.

In this work, smart manufacturing lines are represented as networks of stations, buffers, and transport links with local resource constraints and global objectives defined over throughput, lead times, and energy usage [7]. Each decision-making entity is modeled

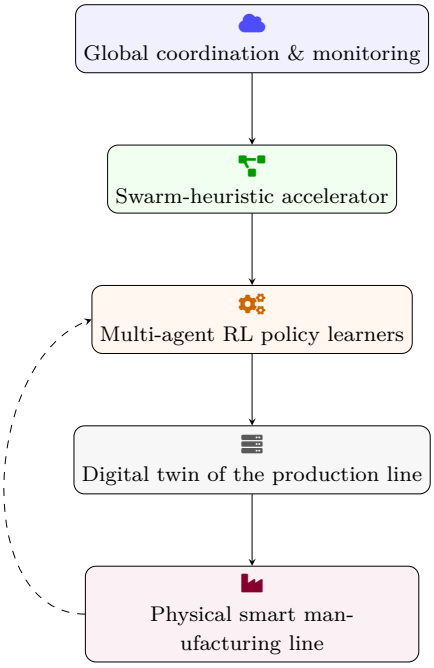

Figure 1. High-level architecture of swarm-accelerated multi-agent reinforcement learning for distributed control in a smart manufacturing line. Global coordination services configure the swarm-heuristic layer, which shapes multi-agent policy learning over a digital twin of the line before deployment to the physical production system. Dashed feedback closes the loop with measurements from the shop floor.

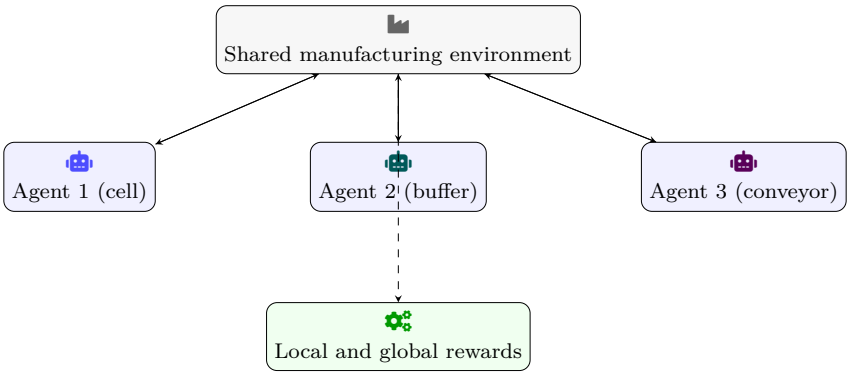

Figure 2. Multi-agent reinforcement learning loop for a smart manufacturing line. A shared environment broadcasts compact observations to distributed controllers at cells, buffers, and conveyors, each implemented as an autonomous agent. Agents return control actions, and the environment produces local and global rewards, forming the underlying multi-agent Markov decision process.

as an agent that receives partial observations about local queues, machine states, and upstream or downstream congestion indicators. The agents' goal is to collectively optimize a global performance criterion while respecting local operational constraints and safety limits. To structure the analysis, the line dynamics are approximated by linear state-space

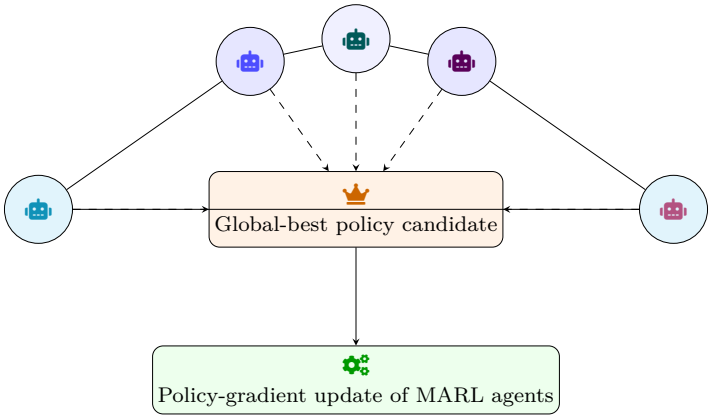

Figure 3. Swarm-heuristic acceleration of multi-agent learning. A population of candidate policy configurations (particles) explores the parameter space while exchanging neighborhood information. The current global-best candidate guides updates in the underlying multi-agent policy-gradient algorithm, enabling faster convergence in high-dimensional distributed control problems.

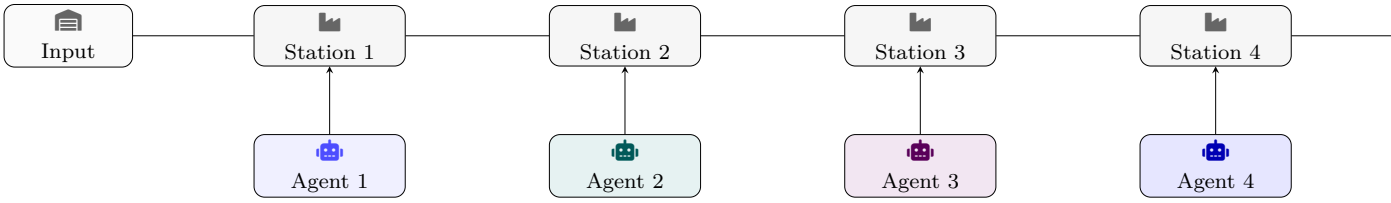

Figure 4. Distributed control over a smart manufacturing line. Each workstation along the line is paired with a local reinforcement learning agent that selects actions such as processing set-points, routing, or buffering decisions. Material flow between stations forms the underlying coupling between agents, while controllers operate locally and asynchronously.

models that capture buffer levels, machine utilization states, and flow conservation relationships. These linear models provide a compact representation that can be used for both control design and analysis of the learning process [8].

The proposed framework introduces a swarm layer that operates over the space of parameterized policies of the agents. Each particle in the swarm corresponds to a joint configuration of policy parameters, temperature parameters for exploration, or weights for shaping rewards. During training, the swarm evaluates candidate configurations over multiple episodes in a simulated manufacturing environment. Performance metrics from these evaluations influence the movement of particles in the search space, while agents continuously adapt their parameters following reinforcement learning updates. The swarm layer thus provides macro-level guidance, while the reinforcement learning layer implements micro-level adaptation to local state information [9]. This interaction is expected to shape the exploration behavior of agents and to reduce the reliance on random noise for discovering useful joint behaviors.

The study focuses on distributed control for smart manufacturing lines under the assumption of reliable communication with limited bandwidth and relatively accurate local

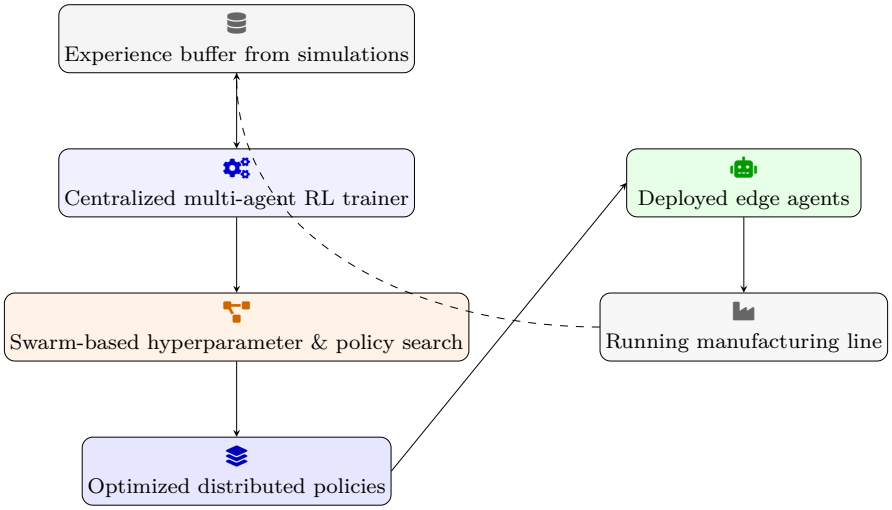

Figure 5. Training and deployment pipeline. Offline, a digital twin generates experience for centralized multi-agent reinforcement learning, while a swarm-based search explores policy and hyperparameter configurations. Optimized policies are then distributed to edge agents deployed on the real manufacturing line, which in turn stream operational traces back to the training pipeline.

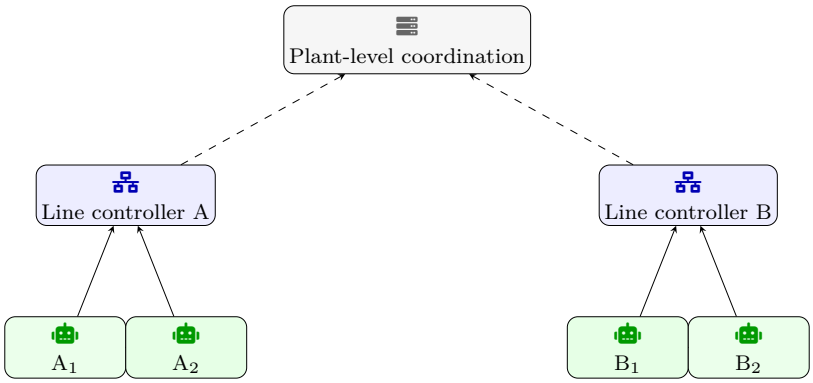

Figure 6. Hierarchical communication topology for distributed control. Local agents coordinate within a line via short-range links, while lightweight summaries flow upward to line-level coordinators and a plant-level orchestrator. This hierarchy constrains communication while still enabling swarm-informed, multi-agent decisions across the production facility.

sensing. The lines considered include serial, parallel, and split-merge topologies, possibly with re-entrant flows, sequence-dependent setup times, and time-varying demand patterns. While the numerical experiments are based on simulated environments, the modeling approach is designed so that it can accommodate digital twin representations of real production lines [10]. The remainder of the paper describes the mathematical formulation of the control problem, the multi-agent reinforcement learning architecture, the integration of swarm heuristics, and the evaluation of the combined scheme on representative smart manufacturing scenarios.

2. Problem Formulation for Distributed Control in Smart Manufacturing Lines

A smart manufacturing line is modeled as a directed graph with nodes corresponding to processing stations and buffers, and edges representing conveyor segments or transport links. Let the line consist of N controllable stations, where each station can represent a single machine, a machine group, or a local cell including a buffer and processing unit. The state of the line at discrete decision time t is represented by a vector x_t that aggregates buffer levels, machine states, and possibly additional process indicators. A compact linear state representation can be written as [11]

$$x_t \in \mathbb{R}^{n_x}.$$

The action vector u_t collects control decisions issued simultaneously by all station-level agents. These actions can encode dispatching decisions, routing choices, machine speed setpoints, or other local control variables. The collective action is represented as

$$u_t \in \mathbb{R}^{n_u}.$$

The dynamics of the line are approximated by a linear time-varying or time-invariant model, depending on the configuration [12]. For a given configuration, a linear discrete-time model is adopted,

$$x_{t+1} = Ax_t + Bu_t + w_t,$$

where A is an $n_x \times n_x$ matrix capturing the propagation of work-in-process and machine state evolution, B is an $n_x \times n_u$ input matrix capturing the effect of control actions, and w_t denotes exogenous disturbances such as demand fluctuations or stochastic processing time variations. In many manufacturing lines, buffer dynamics can be represented through flow conservation relationships. For a buffer level vector q_t and departure vector d_t, a simple approximation is

$$q_{t+1} = q_t + a_t - d_t,$$

where a_t aggregates arrivals from upstream stations and external arrivals, while departures depend on local machine availability and control actions [13].

Control actions must satisfy operational constraints related to capacity limits, safety rules, and discrete decision variables. A linear inequality representation is adopted,

$$Eu_t \leq f,$$

where matrix E and vector f encode constraints such as non-negativity, capacity limits for simultaneous operations, and restrictions on routing decisions [14]. When discrete decisions are required, their relaxation to continuous variables in a bounded interval can be used for the purpose of defining the learning problem, while practical implementation recovers discrete choices by thresholding or sampling.

The performance of the line is measured through a scalar reward function that combines multiple objectives. At time t, the instantaneous reward r_t is expressed as a linear function

of states and actions,

$$r_t = c^\top x_t + d^\top u_t,$$

where vector c assigns weights to state components such as buffer levels, tardiness indicators, and machine utilization, while vector d assigns weights to control signals such as energy consumption or changeover rates [15]. In some cases, a quadratic term in the actions is added to penalize aggressive control moves, but for analytical convenience a linear reward is considered here.

The distributed nature of the control problem is captured by partitioning the line into M local control regions, each associated with an agent. Let $x_t^{(i)}$ denote the local state observed by agent i, possibly including a subset of neighbors' states, and let $u_t^{(i)}$ denote the local action vector. The global state and action vectors can be written as stacked forms,

$$x_t = \begin{bmatrix} x_t^{(1)} \\ \vdots \\ x_t^{(M)} \end{bmatrix}, \quad u_t = \begin{bmatrix} u_t^{(1)} \\ \vdots \\ [16]u_t^{(M)} \end{bmatrix}.$$

The mapping from global actions to state evolution remains linear, but each agent only has access to a subset of state components. Observations $o_t^{(i)}$ for agent i can be represented as

$$o_t^{(i)} = H_i x_t,$$

where matrix H_i selects or aggregates the relevant state components. In partially observable settings, agents may also include features derived from short histories of observations and actions.

The multi-agent reinforcement learning problem is formulated as a cooperative game in which the agents aim to maximize a common expected discounted return [17]. The global return from time 0 is

$$J = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t\right],$$

where $0 < \gamma < 1$ is the discount factor. The expectation is taken over the stochastic disturbances w_t and any randomness in the agents' policies. Operational constraints must hold for all time steps and all disturbance realizations within the modeling assumptions [18]. In some cases, chance constraints can be considered, but the present formulation emphasizes deterministic linear constraints for clarity.

The distributed control problem can be viewed as the search for a joint policy π that maps each agent's observation history to a distribution over local actions. To facilitate analysis and implementation, policies are often parameterized by finite-dimensional vectors. For agent i, a parameter vector θ_i is introduced, and the local policy is written as [19]

$$\pi_i(u^{(i)} \mid o^{(i)}; \theta_i).$$

The joint policy over all agents is then characterized by

$$\theta = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_M[20] \end{bmatrix}.$$

The control design problem can be restated as the maximization of J with respect to θ, subject to the linear state dynamics, constraints, and the implicit distribution over trajectories induced by the policies. The search space is typically high dimensional, and the effect of one agent's parameters depends on those of others, leading to non-convex optimization landscapes.

To analyze the structural properties of the line and the coupling between agents, a compact linear representation can be used. For example, a serial line with buffers between stations may be captured by a tri-diagonal matrix A that encodes the flow of items between neighbors, while parallel and re-entrant flows can be represented by block structures [21]. Although the full dynamics are influenced by stochastic processing times and machine failures, the linear representation serves as a basis for designing reward functions, shaping signals, and communication structures that respect the physical dependencies in the manufacturing line.

3. Multi-Agent Reinforcement Learning Architecture

Table 1. Smart manufacturing line configurations used in the study.

Scenario	Stations	Buffers / link	Disturbance rate (%)
Baseline	8	5	1.0
High-variability	8	5	5.0
Long-line	16	5	1.0
Energy-constrained	8	5	1.0
Reconfigurable	12	4	3.0

Table 2. Agent types and their observation and action spaces.

Agent role	Observation dim.	Action dim.	Control period (s)
Machine controller	18	4	1.0
Buffer controller	10	3	0.5
AGV dispatcher	24	5	2.0
Quality inspector	12	2	5.0
Maintenance planner	20	3	10.0

The control scheme is based on a multi-agent reinforcement learning architecture in which each station-level controller is represented by a learning agent. Agents interact with

Table 3. Control approaches compared in the experiments.

Method	Coordination type	Swarm component
Rule-based	None	None
Centralized MARL	Parameter sharing	None
Independent Q-learning	Decentralized	None
MARL + PSO	Decentralized	Particle swarm
MARL + ACO	Decentralized	Ant colony
Hybrid swarm MARL	Hierarchical	PSO + ACO

Table 4. Swarm heuristic hyperparameters used for policy acceleration.

Parameter	PSO value	ACO value
Population size	40	60
Max iterations per update	15	20
Inertia weight	0.7	–
Cognitive coefficient	1.4	–
Social coefficient	1.4	–
Pheromone evaporation rate	–	0.3
Exploration bias	–	0.6

Table 5. Training configuration for the multi-agent reinforcement learning setup.

Setting	Value	Notes
Episodes per run	5,000	All scenarios
Steps per episode	1,000	Discrete time
Discount factor γ	0.99	Long-term reward
Learning rate α	3×10^{-4}	Adam optimizer
Replay buffer size	1×10^{6}	Shared across agents
Target network update	1,000	Environment steps
Batch size	256	Uniform sampling

a simulation or digital twin of the manufacturing line, which provides state transitions and reward signals according to the linear model and additional stochastic effects. The architecture can be designed to use either value-based or policy-gradient methods, or a combination of both, depending on the properties of the action space and the desired form of the learned policies.

For linear state representations, a natural choice is to approximate value functions by linear combinations of features [22]. Consider the global state value function $V(x;\omega)$ parameterized by a vector ω. A linear function approximation takes the form

$$V(x;\omega) = \phi(x)^{\top}\omega,$$

Table 6. Steady-state performance on the baseline manufacturing scenario.

Method	Norm. throughput ↑	Avg. WIP ↓	Energy / part ↓
Rule-based	1.00	1.00	1.00
Centralized MARL	1.14	0.93	0.97
Independent Q-learning	1.09	0.96	0.99
MARL + PSO	1.22	0.88	0.94
MARL + ACO	1.20	0.89	0.95
Hybrid swarm MARL	1.26	0.85	0.92

Table 7. Effect of swarm-based acceleration on learning efficiency.

Variant	Convergence episodes	Final mean reward	Speedup vs. no-swarm
No-swarm MARL	3,800	1.00	1.0×
PSO-only	2,300	1.03	1.7×
ACO-only	2,500	1.02	1.5×
Hybrid swarm MARL	1,900	1.05	2.0×

Table 8. Communication overhead and latency as the number of agents increases.

Agents	Method	Messages / step	Mean latency (ms)
10	Hybrid swarm MARL	45	2.1
20	Hybrid swarm MARL	96	3.0
40	Hybrid swarm MARL	210	4.6
80	Hybrid swarm MARL	460	7.9

where $\phi(x)$ is a feature vector derived from the state [23]. Features may include raw state components, aggregated buffer levels, machine utilization indicators, or other linear transformations derived from the matrix A. Similarly, state-action value functions $Q(x, u; \eta)$ can be approximated as

$$Q(x, u; \eta) = \psi(x, u)^\top \eta,$$

where $\psi(x, u)$ collects features of states and actions. The use of linear value function approximation simplifies theoretical analysis and can reduce the risk of instability compared with high-capacity nonlinear approximators [24].

In a distributed setting, it is often beneficial to decompose value functions into local contributions associated with each agent. For a cooperative line with shared reward, one may consider an additive decomposition,

$$Q(x, u; \eta) = \sum_{i=1}^{M} Q_i(x^{(i)}, u^{(i)}; \eta_i),$$

where Q_i depends on the local state and action of agent i [25]. This factorization is approximate in general, since the true value function depends on all interactions, but it can

significantly reduce the complexity of learning. Each agent then maintains local parameters η_i and updates them based on local information and a limited set of coordination signals derived from global measurements.

Policy representations must accommodate local observations and possibly continuous, discrete, or hybrid actions. For continuous actions such as speed setpoints, Gaussian policies with linear means can be used. A simple parameterization is [26]

$$\mu_i(o^{(i)}; \theta_i) = W_i o^{(i)},$$

where W_i is a parameter matrix reshaped into the vector θ_i. The actual action is sampled from

$$u^{(i)} \sim \mathcal{N}\big(\mu_i(o^{(i)}; \theta_i), \Sigma_i\big), [27]$$

with fixed or learned covariance Σ_i. For discrete decisions such as routing choices, a softmax parameterization can be considered, where preference scores are linear in observations and transformed into probabilities.

Learning updates for policy parameters can be based on gradient estimates obtained from sampled trajectories. In a centralized training, decentralized execution paradigm, a central learner collects experience tuples from all agents and computes joint gradients. The gradient of the expected return with respect to θ_i can be written as [28]

$$\nabla_{\theta_i} J = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \nabla_{\theta_i} \log \pi_i(u_t^{(i)} \mid o_t^{(i)}; \theta_i) G_t\right],$$

where G_t denotes an estimate of the return-to-go or an advantage estimate. In practice, these expectations are approximated by empirical averages over episodes or mini-batches [29].

Temporal-difference learning is used to update value function parameters. For example, the parameter vector ω of a linear state value function can be updated according to

$$\omega_{k+1} = \omega_k + \alpha_k \delta_k \phi(x_k), [30]$$

where δ_k is the temporal-difference error. For each transition (x_k, u_k, r_k, x_{k+1}), the error is computed as

$$\delta_k = r_k + \gamma V(x_{k+1}; \omega_k) - V(x_k; \omega_k).$$

In the multi-agent case, the temporal-difference error can be computed using either global or local rewards, depending on the chosen credit assignment scheme.

Since the environment is non-stationary from the perspective of each agent due to the learning of others, stabilization techniques are considered [31]. These may include slow target networks, experience replay with prioritized sampling, or periodic synchronization of policy parameters across agents. In the linear approximation context, one can also analyze the relation between the spectral properties of the transition matrix A, the discount factor γ, and the convergence of temporal-difference updates. For instance, when the spectral radius of γA is less than one, the linear system exhibits contractive behavior that can support stable value iteration for a fixed policy.

Coordination among agents is facilitated through low-dimensional messages that summarize local congestion or forecasted arrival rates [32]. For example, an agent may receive an estimate of the expected inflow from upstream stations, computed as a linear function of their buffer states. This can be represented as

$$\hat{a}_t^{(i)} = L_i x_t,$$

where L_i is a sparse matrix capturing the relevant upstream connections. These coordination signals can be treated as part of the observation vector for each agent. The reinforcement learning architecture thus operates on an augmented observation space that encodes both local state and limited global information [33].

During training, episodes are generated by simulating the manufacturing line under a variety of demand profiles and disturbance patterns. The linearly modeled dynamics determine the propagation of work-in-process, while stochastic arrivals and failures introduce variability. Agents explore their action spaces according to the current policies, and their parameters are updated after each batch of episodes. The training process is computationally demanding due to the dimensionality of the joint state-action space and the need for averaging over many stochastic realizations [34]. Swarm heuristics are introduced in the next section as an additional mechanism to structure and accelerate the exploration in this space.

4. Swarm-Heuristic Acceleration of Multi-Agent Reinforcement Learning

Swarm heuristics operate on a population of candidate solutions that interact through simple update rules inspired by collective behavior. In the context of multi-agent reinforcement learning for smart manufacturing, swarm heuristics are used to search over the space of joint policy parameters or over additional hyperparameters influencing exploration and coordination. The central idea is to embed a swarm layer that periodically proposes new configurations of parameters, evaluates them using the reinforcement learning environment, and adjusts the population based on observed performance.

Consider a swarm of K particles, where each particle k is associated with a vector $z^{(k)}$ representing a candidate joint parameter configuration. A simple particle swarm update rule uses particle velocities $v^{(k)}$, personal best positions $p^{(k)}$, and a global best position g. The velocity update can be expressed as [35]

$$v_{j+1}^{(k)} = \chi v_j^{(k)} + \varphi_1 r_{1,j}^{(k)} \big(p_j^{(k)} - z_j^{(k)}\big) + \varphi_2 r_{2,j}^{(k)} \big(g_j - z_j^{(k)}\big),$$

where χ is an inertia coefficient, φ_1 and φ_2 are acceleration coefficients, and $r_{1,j}^{(k)}$ and $r_{2,j}^{(k)}$ are random factors. The position update is

$$z_{j+1}^{(k)} = z_j^{(k)} + v_{j+1}^{(k)}.$$

In this setting, the vector $z^{(k)}$ can include concatenated policy parameters of all agents, exploration noise scales, or weights for combining different reward components.

To connect the swarm layer with the reinforcement learning process, each particle is evaluated by running episodes with policy parameters instantiated from its position vector. Let $J(z^{(k)})$ denote the estimated expected return obtained by running multiple episodes with configuration $z^{(k)}$. After evaluation, the personal best position $p^{(k)}$ and the global best position g are updated according to comparisons of the scalar performance metric. These updates define a stochastic approximation of the gradient of performance with respect to parameters, but they do not rely on differentiability of the environment or policies [36].

The swarm layer interacts with the learning agents in several ways. One option is to use the swarm to propose initializations for policy parameters before each training phase. In this scheme, the particle positions are mapped to θ vectors, and the reinforcement learning algorithm is run for a fixed number of iterations starting from these initializations. The resulting performance feedback is returned to the swarm, which then updates particle positions. Another option is to run the swarm and reinforcement learning concurrently, where the swarm modifies policy parameters on a slower time scale while agents perform gradient-based updates within each episode [37].

The distributed nature of the control problem suggests that particles may also represent structured variations across agents. For example, instead of encoding all policy parameters, a particle could encode a lower-dimensional coordination vector λ that modifies local reward functions. A simple linear shaping scheme can be written as

$$r_t^{(i)} = r_t + \lambda^\top h_t^{(i)},$$

where $h_t^{(i)}$ collects local shaping features such as buffer deviations from targets or synchronization signals with neighboring stations. In this case, the swarm searches over the space of λ vectors, while agents adapt their policies under the shaped rewards [38]. This separation of roles allows the swarm to focus on a small parameter space, which can improve its efficiency and reduce interaction complexity.

The linear state-space representation facilitates the design of swarm-based search directions. For example, sensitivity measures of the expected return with respect to certain linear combinations of state components can be approximated using perturbation experiments. Consider a direction vector d in the space of policy parameters. A finite-difference estimate of the directional derivative can be written as [39]

$$\Delta J(d) = J(\theta + \epsilon d) - J(\theta),$$

for a small scalar ϵ [40]. The swarm can be biased to move along directions with positive estimated improvements by incorporating such finite-difference information into velocity updates, while still maintaining stochastic exploration.

Swarm heuristics can also be used to adapt learning rates and exploration magnitudes. For each agent i, a learning rate α_i and exploration scale σ_i may be included in the swarm parameter vector. The agents' update rules depend linearly on these quantities, for instance in temporal-difference updates. The swarm thus acts as an outer-loop tuner that seeks combinations of learning rates and exploration scales that yield better performance under the stochastic dynamics of training [41].

The convergence behavior of the combined swarm and reinforcement learning process is influenced by the time-scale separation between the two layers. If the swarm operates too quickly relative to the reinforcement learning updates, the agents may not fully exploit the information in each proposed configuration, leading to noisy performance evaluations. Conversely, if the swarm operates too slowly, its influence on exploration may be limited. A practical approach is to choose an intermediate time scale, where each particle position is evaluated by a moderate number of episodes, and reinforcement learning is allowed to adjust local parameters significantly between swarm updates.

To avoid violating operational constraints during exploration, particles can be restricted to feasible regions of the parameter space, and policy parameterizations can be chosen to ensure that actions satisfy constraints [42]. For linear constraints of the form $Eu_t \leq f$, one can construct policy outputs as projections onto the feasible set. If a raw action proposal $\tilde{u}_t$ is generated by the policy, the actual applied action is

$$u_t = \Pi_{\mathcal{U}}(\tilde{u}_t),$$

where $\Pi_{\mathcal{U}}$ denotes the projection onto the polyhedral set $\mathcal{U} = \{u : Eu \leq f\}$. This projection can be formulated as a quadratic program with linear constraints, and in some cases admits closed-form expressions for simple box constraints.

The integration of swarm heuristics and reinforcement learning introduces additional stochastic elements into the training process. Analysis of sample complexity and convergence properties becomes more intricate because the noise arises from both environmental stochasticity and the randomness of swarm updates [43]. However, the linear dynamics and reward functions permit certain bounding arguments. For example, if the state vector remains bounded in expectation under all considered policy configurations, and if reward functions are linear, one can derive upper bounds on the variance of return estimates used in swarm evaluation. These bounds can inform the choice of the number of episodes per particle evaluation.

5. Application to Smart Manufacturing Lines

To illustrate the modeling and algorithmic components, consider a smart manufacturing line consisting of multiple processing stages with intermediate buffers and alternative routing options. Each stage may contain one or several parallel machines, and products may follow different paths depending on their type [44]. The line is equipped with sensors providing real-time data on buffer levels, machine status, and energy consumption. The digital twin used for training incorporates a linear approximation of the main flow dynamics, combined with stochastic models for processing times, failure events, and maintenance interventions.

The state vector x_t is constructed by stacking buffer levels, binary or continuous indicators of machine availability, and auxiliary variables representing aggregated demand and energy prices. Let q_t denote the vector of buffer levels across all intermediate buffers, and

let s_t represent machine states [45]. The state vector is

$$x_t = \begin{bmatrix} q_t \\ s_t \end{bmatrix}.$$

The dynamics of buffer levels are approximated using deterministic flow equations with added stochastic disturbances. For a buffer between station i and station j, the level evolves according to

$$q_{t+1}^{(i,j)} = q_t^{(i,j)} + a_t^{(i,j)} - d_t^{(i,j)},$$

where arrivals and departures depend on local actions and upstream completion times [46]. Machine states evolve according to simple models that capture busy, idle, and failure modes.

Agents are assigned to logical segments of the line, typically to stations or groups of nearby stations that share buffers or transport resources. Agent i observes local buffers, machine states, and a small set of aggregated indicators from neighbors. Its observation vector is thus written as [47]

$$o_t^{(i)} = H_i x_t,$$

with H_i constructed to include local and near-neighbor states. Actions for each agent include dispatching decisions that determine which job to process next, routing choices for jobs that can be sent to multiple downstream stations, and possibly adjustments of processing speeds or energy modes. These actions are encoded in a continuous or mixed representation, relaxed for training purposes. The joint action vector u_t is subject to constraints capturing machine capacity and mutual exclusivity of routing decisions [48].

The reward signal is designed to reflect throughput, work-in-process levels, tardiness, and energy usage. A simple linear reward function combines these criteria as

$$r_t = \beta_T T_t - \beta_W W_t [49] - \beta_L L_t - \beta_E E_t,$$

where T_t denotes throughput, W_t work-in-process, L_t tardiness-related penalties, and E_t energy consumption, with nonnegative weighting coefficients β_T, β_W, β_L, and β_E. Throughput and work-in-process can be computed from buffer and completion data in the state vector, while tardiness is derived from job completion times relative to due dates. Energy consumption is estimated from machine states and speed settings [50].

The linear approximation of dynamics is calibrated using either physical models or identification from data generated by more detailed simulations. For example, average processing times at each station and routing probabilities are used to derive nominal flow rates, which yield coefficients in the matrix A. Similarly, the influence of speed setpoints on throughput can be linearized around typical operating points to obtain entries in the matrix B. Although this linear approximation cannot capture all nonlinearities and stochastic effects, it provides a compact structure for designing control policies and coordination mechanisms [51].

During training, various demand scenarios are considered, including time-varying arrival rates and changes in product mix. Disturbance realizations include randomly generated

failure times and repair durations. The multi-agent reinforcement learning algorithm, augmented with the swarm layer, is run over many training episodes. At the beginning of each training epoch, the swarm layer proposes a set of configurations of policy parameters or shaping weights. For each configuration, a subset of agents may be initialized accordingly, and episodes are simulated with these policies [52]. The performance of each configuration is measured using empirical averages of return and additional metrics such as average throughput and average work-in-process.

The distributed nature of the control is reflected in the communication patterns among agents. Communication is restricted to local neighborhoods, meaning that agents can exchange congestion indicators or simple summary statistics only with directly connected neighbors. The reinforcement learning policies and swarm-generated parameters must operate within this communication graph [53]. In practice, this restriction induces a form of locality in the joint policy space, which can be exploited by designing swarm particles that modify only parameters of agents within a limited radius around identified bottlenecks.

Resource-sharing scenarios highlight the interplay between local autonomy and global performance. For example, a set of parallel machines may serve multiple product families. Agents controlling these machines must decide on job sequencing and routing of jobs from upstream stations. The reward function penalizes excessive waiting times and imbalance of utilization across machines [54]. Under the swarm-accelerated reinforcement learning scheme, particles can encode different bias patterns in routing policies, such as preferring specific machines for certain products or distributing jobs more evenly. Reinforcement learning agents then adapt lower-level decisions within these patterns, using observations of local queues and downstream congestion.

Energy-aware operation introduces additional trade-offs. Machines can be operated at different speed or energy modes, with higher speed often implying higher energy usage. The linear model can incorporate these effects by including energy-related state components and by structuring the input matrix B so that control variables influence both throughput and energy consumption [55]. The reward weights β_E and β_T determine the balance between productivity and energy usage. The swarm layer can explore different combinations of these weights or different functional forms for energy penalties, allowing the reinforcement learning agents to adapt their policies to various cost structures.

6. Experimental Evaluation and Discussion

The performance of the proposed multi-agent reinforcement learning architecture with swarm acceleration is evaluated using simulated smart manufacturing lines with different topologies and operating conditions. A typical experimental setup involves a serial line with re-entrant flows and several branches, comprising ten to twenty stations with associated buffers [56]. The digital twin simulates processing times as random variables with specified means and variances, and includes stochastic failure and repair processes for selected machines. Demand patterns are generated with diurnal variations and occasional demand spikes.

Agents are initialized with random linear policies and value function parameters. The swarm is initialized with particles distributed around these initializations, with moderate variation in parameter values to provide diversity. For each training run, both a baseline multi-agent reinforcement learning scheme without swarm assistance and the swarm-accelerated scheme are executed under identical random seeds for arrival and failure processes, to the extent feasible [57]. Performance is measured using discounted return, average throughput, average work-in-process, average tardiness, and average energy consumption over evaluation episodes held out from training.

In the baseline configuration, learning may proceed slowly due to the high dimensionality of the joint action space and the sparse nature of global rewards, especially in scenarios where throughput and tardiness become meaningful only after several time steps. Agents rely on random exploration noise to discover beneficial joint behaviors, which can lead to significant variability between runs. In contrast, the swarm-accelerated scheme augments exploration by proposing coordinated parameter updates that affect multiple agents simultaneously [58]. When particles correspond to joint policy configurations, the swarm tends to bias exploration toward regions where coordinated behavior is more likely to improve throughput or reduce congestion.

Empirical results from repeated simulation runs suggest that swarm-accelerated training can reach certain performance levels with fewer episodes than the baseline in many instances, although the magnitude of this effect depends on the line topology and disturbance characteristics. For example, in a line with a pronounced bottleneck station, particles that adjust policies near the bottleneck can quickly exploit information about upstream and downstream congestion, leading to more stable queue lengths and reduced variability in throughput. In systems with more distributed bottlenecks, the advantage of swarm-based parameter search may be less pronounced because beneficial configurations are more scattered in the parameter space.

The energy-performance trade-off is analyzed by varying reward weights associated with energy consumption [59]. For each set of weights, the swarm-accelerated scheme and the baseline scheme are trained separately. The resulting policies are then evaluated along a Pareto-like curve relating throughput and energy usage. In several scenarios, the swarm-accelerated approach identifies policy configurations that achieve similar throughput with lower energy consumption compared to those discovered by the baseline within the same number of training episodes. However, when training is extended for a much larger number of episodes, the performance gap narrows, indicating that swarm acceleration primarily affects the transient learning phase rather than the asymptotic achievable performance, under the considered configurations [60].

An aspect of interest is the robustness of learned policies to changes in demand profiles and failure rates. To assess this, policies obtained under nominal conditions are tested under perturbed scenarios where arrival rates are increased or decreased, and failure intensities are varied. Both baseline and swarm-accelerated policies are evaluated without further training. The linear structure of the model and the policies tends to facilitate

generalization to moderate changes, but large deviations in demand or failure patterns can degrade performance. In some cases, policies obtained through swarm-accelerated training exhibit slightly more robust behavior, possibly because the swarm explores a wider region of the parameter space during training [61]. Nonetheless, the dominant factor in robustness appears to be the diversity of scenarios included during training rather than the presence or absence of swarm acceleration.

The computational cost of the combined scheme is also examined. Swarm evaluation requires running multiple episodes for each particle, which increases simulation load. However, this additional cost can be offset by the reduction in the number of training epochs needed to reach a given performance threshold [62]. The actual trade-off depends on implementation details such as parallelization of simulations and the relative cost of environment simulation versus learning updates. In high-fidelity digital twin environments, where simulation is computationally expensive, careful tuning of the number of particles and the number of episodes per particle is important to avoid excessive overhead.

From a control perspective, the learned policies are examined for interpretability. Because the policies are linear functions of observations in the considered implementation, they can be inspected by analyzing the parameter matrices W_i associated with each agent. These matrices reveal how agents weight buffer levels, machine states, and neighbor indicators when selecting actions [63]. In some trained configurations, agents display behavior reminiscent of classical dispatching rules, such as prioritizing shorter processing times or balancing loads across parallel machines, but adjusted to account for energy penalties and downstream congestion. The presence of the swarm layer appears to steer policies toward sets of parameters where such patterns emerge more consistently across agents.

7. Conclusion

This work has considered a distributed control framework for smart manufacturing lines based on multi-agent reinforcement learning accelerated by swarm heuristics. Smart manufacturing environments are characterized by complex interactions among machines, buffers, and transport systems, along with time-varying demand and disturbance patterns [64]. A linear state-space representation was adopted to model the evolution of buffer levels and machine states under the influence of control actions, while reinforcement learning agents were tasked with learning policies that map local observations to actions in a cooperative setting.

The multi-agent reinforcement learning architecture used linear function approximation for value functions and linear policies for agents, which facilitated analysis and inspection of learned behaviors. A swarm-based metaheuristic layer was introduced to search over joint policy parameters, exploration magnitudes, and reward shaping weights. This layer proposed candidate configurations, evaluated them through episodes in a simulated digital twin of the manufacturing line, and updated a population of particles using simple velocity and position updates. The swarm thus provided an additional exploration mechanism that

could adapt to the observed performance landscape without relying solely on gradient-based updates [65].

Applications to simulated smart manufacturing lines with various topologies illustrated how the combined scheme may influence learning speed and policy characteristics. In several scenarios, swarm acceleration reduced the number of training episodes required to reach specified performance levels in terms of throughput, work-in-process, tardiness, and energy usage. The magnitude of these effects depended on the structure of the manufacturing line, the nature of bottlenecks, and the diversity of demand and disturbance patterns. When training was extended for longer periods, differences between baseline and swarm-accelerated schemes tended to diminish, suggesting that the primary impact of swarm heuristics is on the transient learning phase [66].

The distributed nature of the control and the linear policy structures allowed for examination of learned policies in terms of their sensitivities to buffer levels and machine states. In some cases, the resulting policies resembled traditional dispatching rules adapted to energy-aware objectives and local congestion information. The swarm layer contributed by guiding exploration toward parameter regions where such rules emerged in a coordinated way across agents, while reinforcement learning provided fine-tuning based on state-dependent feedback.

Several limitations of the present study deserve attention. The linear dynamic models used for control design and training do not capture all nonlinearities and stochastic aspects of real manufacturing lines, and the performance of learned policies in real systems would depend on the fidelity of the digital twin [67]. The use of linear policies and value function approximations simplifies analysis but may restrict achievable performance in highly nonlinear settings. In addition, the evaluation of swarm particles incurs computational cost that must be balanced against potential gains in learning speed, especially when high-fidelity simulations are involved.

Future work may consider extensions to richer function approximators while maintaining some of the structure provided by linear models, as well as adaptive mechanisms that adjust the intensity of swarm exploration based on indicators of learning progress. The integration of safety constraints and formal guarantees on constraint satisfaction during exploration is another area of interest, particularly in contexts where physical experiments complement or replace simulations. Overall, the combination of multi-agent reinforcement learning with swarm heuristics represents one possible approach for addressing the distributed control challenges of smart manufacturing lines under uncertainty and reconfigurability, and further investigations may clarify in which settings such combinations provide the most tangible benefits [68].

References

[1] J. Terán, J. Aguilar, and M. Cerrada, "Collective learning in multi-agent systems based on cultural algorithms," *CLEI Electronic Journal*, vol. 17, no. 2, pp. 8–8, Aug. 1, 2014. DOI: 10.19153/cleiej.17.2.7

[2] J. Wang, X. Nian, and H.-b. Wang, "Consensus and formation control of discrete-time multi-agent systems," *Journal of Central South University of Technology*, vol. 18, no. 4, pp. 1161–1168, Jul. 10, 2011. DOI: 10.1007/s11771-011-0818-z

[3] W. Alrawagfeh, E. Brown, and M. Mata-Montero, "Norms of behaviour and their identification and verification in open multi-agent societies," in IGI Global, Apr. 12, 2012, pp. 129–145. DOI: 10.4018/978-1-4666-1565-6.ch009

[4] A. Yunianta et al., "The ontology-based methodology phases to develop multi-agent system (ommas)," *Proceeding of the Electrical Engineering Computer Science and Informatics*, vol. 4, no. 1, pp. 300–305, Oct. 1, 2017. DOI: 10.11591/eecsi.v4.1027

[5] W. Yu, G. Wen, and Y. Li, "Application of fractional-order calculus in a class of multi-agent systems," in Germany: Springer Berlin Heidelberg, Aug. 15, 2015, pp. 229–261. DOI: 10.1007/978-3-662-47824-0_9

[6] R. Chandrasekar and S. Misra, "Using zonal agent distribution effectively for routing in mobile ad hoc networks," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 3, no. 2, pp. 82–89, 2008.

[7] W. Zemzem and M. Tagina, "Mdai - cooperative multi-agent learning in a large dynamic environment," in Germany: Springer International Publishing, Sep. 1, 2015, pp. 155–166. DOI: 10.1007/978-3-319-23240-9_13

[8] K. Matsumoto, A. Tanimoto, and N. Mori, "A dependable multi-agent system with self-diagnosable function," in InTech, Apr. 1, 2011. DOI: 10.5772/14455

[9] Y. Wei, S. Zhang, and J. Cao, "Edcis - coordination among multi-agents using process calculus and eca rule," in Germany: Springer Berlin Heidelberg, Sep. 2, 2002, pp. 456–465. DOI: 10.1007/3-540-45785-2_36

[10] D. Kazakov and D. Kudenko, "Easss - machine learning and inductive logic programming for multi-agent systems," in Germany: Springer Berlin Heidelberg, Jun. 28, 2001, pp. 246–272. DOI: 10.1007/3-540-47745-4_11

[11] W. Xiong, D. W. C. Ho, and J. Cao, "Impulsive consensus of multi-agent directed networks with nonlinear perturbations," *International Journal of Robust and Nonlinear Control*, vol. 22, no. 14, pp. 1571–1582, Jul. 19, 2011. DOI: 10.1002/rnc.1768

[12] K. Arkoudas and S. Bringsjord, *CLIMA - Metareasoning for multi-agent epistemic logics*. Germany: Springer Berlin Heidelberg, Aug. 19, 2005. DOI: 10.1007/11533092_7

[13] S. Li, G. Feng, X. Luo, and X. Guan, "Output consensus of heterogeneous linear multi-agent systems subject to different disturbances," *Asian Journal of Control*, vol. 18, no. 2, pp. 757–762, Dec. 19, 2014. DOI: 10.1002/asjc.1078

[14] A. Komenda, P. Novák, and M. Pchouek, *Decentralized multi-agent plan repair in dynamic environments*, Jan. 1, 2012. DOI: 10.48550/arxiv.1202.2773

[15] V. Vijaykumar, R. Chandrasekar, and T. Srinivasan, "An ant odor analysis approach to the ant colony optimization algorithm for data-aggregation in wireless sensor networks," in *2006 International Conference on Wireless Communications, Networking and Mobile Computing*, IEEE, 2006, pp. 1–4.

[16] Z. Gao, T. Kawasoe, A. Yamamoto, and T. Ishida, "Prima - meta-level architecture for executing multi-agent scenarios," in Germany: Springer Berlin Heidelberg, Jul. 17, 2002, pp. 163–177. DOI: 10.1007/3-540-45680-5_12

[17] I. Stengel, *Optimization in multi-agent systems : Structures and procedures*, Jan. 1, 2008. DOI: 10.34719/yzgv9817

[18] null MinskyNaftaly, "Law-governed multi-agent systems," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1–1, May 15, 2005. DOI: 10.1145/1082983.1082965

[19] M. Egerstedt and null Xiaoming Hu, "Formation constrained multi-agent control," in *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)*, vol. 4, IEEE, Nov. 13, 2002, pp. 3961–3966. DOI: 10.1109/robot.2001.933235

[20] H. J. Kim, S. Yoon, and T. Chung, "A research on non-interactive multi agents by acs & direction vector algorithm," *Journal of the Korea Society of Computer and Information*, vol. 15, no. 12, pp. 11–18, Dec. 31, 2010. DOI: 10.9708/jksci.2010.15.12.011

[21] Y. Jung, M. Kim, A. Masoumzadeh, and J. Joshi, "A survey of security issue in multi-agent systems," *Artificial Intelligence Review*, vol. 37, no. 3, pp. 239–260, Jun. 4, 2011. DOI: 10.1007/s10462-011-9228-8

[22] J. Zhang, *The technical foundation of a multiagent system*, Apr. 26, 2017. DOI: 10.1002/9781118890073.ch2

[23] W.-S. Wong and J. Baillieul, "Information-based multi-agent systems," in Springer London, Jul. 21, 2015, pp. 571–575. DOI: 10.1007/978-1-4471-5058-9_153

[24] M. A. Rigi and F. Khoshalhan, "Eliciting user preferences in multi-agent meeting scheduling problem," in IGI Global, Oct. 1, 2012, pp. 108–124. DOI: 10.4018/978-1-4666-2047-6.ch007

[25] T. Srinivasan, R. Chandrasekar, V. Vijaykumar, V. Mahadevan, A. Meyyappan, and M. Nivedita, "Exploring the synergism of a multiple auction-based task allocation scheme for power-aware intrusion detection in wireless ad-hoc networks," in *2006 10th IEEE Singapore International Conference on Communication Systems*, IEEE, 2006, pp. 1–5.

[26] S. Yang, X. Liao, Y. Liu, and X. Chen, "Consensus of delayed multi-agent dynamical systems with stochastic perturbation via impulsive approach," *Neural Computing and Applications*, vol. 28, no. 1, pp. 647–657, Jun. 6, 2016. DOI: 10.1007/s00521-016-2393-6

[27] B. Heinemann, "Lfcs - augmenting subset spaces to cope with multi-agent knowledge," in Germany: Springer International Publishing, Dec. 10, 2015, pp. 130–145. DOI: 10.1007/978-3-319-27683-0_10

[28] A. Verma and K. K. Pattanaik, "Multi-agent communication-based train control system for indian railways: The behavioural analysis," *Journal of Modern Transportation*, vol. 23, no. 4, pp. 272–286, Aug. 1, 2015. DOI: 10.1007/s40534-015-0083-1

[29] C. Oechslein, F. Klügl, R. Herrler, and F. Puppe, *CEEMAS - UML for Behavior-Oriented Multi-agent Simulations*. Germany: Springer Berlin Heidelberg, Mar. 14, 2002. DOI: 10.1007/3-540-45941-3_23

[30] S. Yang, J. Cao, and J. Lu, "A new protocol for finite-time consensus of detail-balanced multi-agent networks.," *Chaos (Woodbury, N.Y.)*, vol. 22, no. 4, pp. 043 134–, Dec. 1, 2012. DOI: 10.1063/1.4768662

[31] D. D. Vecchio and R. M. Murray, *Cooperative Control of Distributed MultiAgent Systems - Complexity Management in the State Estimation of MultiAgent Systems*. Wiley, Nov. 30, 2007. DOI: 10.1002/9780470724200.ch16

[32] E. Herrera-Viedma, C. Porcel, F. Herrera, L. Martínez, and A. Lopez-Herrera, "Techniques to improve multi-agent systems for searching and mining the web," in Germany: Springer Berlin Heidelberg, Jul. 25, 2005, pp. 463–486. DOI: 10.1007/11004011_23

[33] M. Thimm, "Strategic argumentation in multi-agent systems," *KI - Künstliche Intelligenz*, vol. 28, no. 3, pp. 159–168, Jun. 17, 2014. DOI: 10.1007/s13218-014-0307-2

[34] M. Dastani, "Programming multi-agent systems," *The Knowledge Engineering Review*, vol. 30, no. 4, pp. 394–418, Sep. 3, 2015. DOI: 10.1017/s0269888915000077

[35] C. Ramachandran, R. Malik, X. Jin, J. Gao, K. Nahrstedt, and J. Han, "Videomule: A consensus learning approach to multi-label classification from noisy user-generated videos," in *Proceedings of the 17th ACM international conference on Multimedia*, 2009, pp. 721–724.

[36] J.-H. Kim, H.-J. Kwon, and K.-S. Hong, "Location awareness-based intelligent multi-agent technology," *Multimedia Systems*, vol. 16, no. 4, pp. 275–292, May 15, 2010. DOI: 10.1007/s00530-010-0194-9

[37] M. Dastani, C. M. Jonker, and J. Treur, *AOSE - A Requirement Specification Language for Configuration Dynamics of Multi-agent Systems*. Germany: Springer Berlin Heidelberg, Feb. 21, 2002. DOI: 10.1007/3-540-70657-7_12

[38] A. Agnetis, D. Pacciarelli, and A. Pacifici, "Multi-agent single machine scheduling," *Annals of Operations Research*, vol. 150, no. 1, pp. 3–15, Jan. 10, 2007. DOI: 10.1007/s10479-006-0164-y

[39] A. Machado, G. Ramalho, J.-D. Zucker, and A. Drogoul, "Mabs - multi-agent patrolling: An empirical analysis of alternative architectures," in Germany: Springer Berlin Heidelberg, Apr. 15, 2003, vol. 2581, pp. 155–170. DOI: 10.1007/3-540-36483-8_11

[40] P. Guyot, A. Drogoul, and C. Lemaître, "Aamas - using emergence in participatory simulations to design multi-agent systems," in *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, ACM, Jul. 25, 2005, pp. 199–203. DOI: 10.1145/1082473.1082503

[41] W. Zhu and M. Chen, "Ccta (3) - the architecture of information fusion system ingreenhouse wireless sensor network based on multi-agent," in Germany: Springer US, Apr. 11, 2009, vol. 3, pp. 2057–2066. DOI: 10.1007/978-1-4419-0213-9_56

[42] H. Liang, F. Kang, and H. Li, "Uuv formation system modeling and simulation research based on multi-agent interaction chain," *International Journal of Modeling, Simulation, and Scientific Computing*, vol. 06, no. 02, pp. 1 550 019–, May 29, 2015. DOI: 10.1142/s1793962315500191

[43] B. Banerjee, L. Kraemer, and J. Lyle, "Multi-agent plan recognition: Formalization and algorithms," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 24, no. 1, pp. 1059–1064, Jul. 4, 2010. DOI: 10.1609/aaai.v24i1.7746

[44] H. R. Choi, H. S. Kim, Y. S. Park, and B. J. Park, "A multi-agent system for optimal supply chain management," in IGI Global, May 24, 2011, pp. 794–817. DOI: 10.4018/978-1-60566-677-8.ch052

[45] R. Chandrasekar and T. Srinivasan, "An improved probabilistic ant based clustering for distributed databases," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI*, 2007, pp. 2701–2706.

[46] N. Y. Mutovkina, "Fuzzy complex assessment of activities of the agent in multi-agent system," in Springer International Publishing, Aug. 20, 2017, pp. 303–313. DOI: 10.1007/978-3-319-67349-3_29

[47] X. Chen, G. Chen, W. Cao, and M. Wu, "Cooperative learning with joint state value approximation for multi-agent systems," *Journal of Control Theory and Applications*, vol. 11, no. 2, pp. 149–155, May 5, 2013. DOI: 10.1007/s11768-013-1141-z

[48] V. Samoylov and V. Gorodetsky, "Ais-adm - ontology issue in multi-agent distributed learning," in Germany: Springer Berlin Heidelberg, May 20, 2005, pp. 215–230. DOI: 10.1007/11492870_18

[49] H. Achten and A. Jessurun, "A multi-agent mah jong playing system : Towards real-time recognition of graphic units in graphic representations," *Acta Polytechnica*, vol. 43, no. 2, pp. 28–33, Jan. 2, 2003. DOI: 10.14311/416

[50] S. Ahmed, M. N. Karsiti, and R. N. K. Loh, "Control analysis and feedback techniques for multi agent robots," in I-Tech Education and Publishing, Jan. 1, 2009, vol. 2009. DOI: 10.5772/6597

[51] P. Huang and K. Sycara, "Aamas - multi-agent learning in extensive games with complete information," in *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, ACM, Jul. 14, 2003, pp. 701–708. DOI: 10.1145/860575.860688

[52] Z. Shanying, C. Cailian, and G. Xinping, "Consensus protocol for heterogeneous multi-agent systems: A markov chain approach," *Chinese Physics B*, vol. 22, no. 1, pp. 018 901–, Jan. 31, 2013. DOI: 10.1088/1674-1056/22/1/018901

[53] J. A. A. Araúzo, R. del Olmo, and J. J. Laviós, "Gaia analysis and design of a multi-agent-based shop floor control system," *Dirección y Organización*, no. 54, pp. 26–35, Dec. 1, 2014. DOI: 10.37610/dyo.v0i54.457

[54] M. A. Rigi and F. Khoshalhan, "Eliciting user preferences in multi-agent meeting scheduling problem," *International Journal of Intelligent Information Technologies*, vol. 7, no. 2, pp. 45–62, Apr. 1, 2011. DOI: 10.4018/jiit.2011040103

[55] R. Chandrasekar and S. Misra, "Introducing an aco based paradigm for detecting wildfires using wireless sensor networks," in *2006 International Symposium on Ad Hoc and Ubiquitous Computing*, IEEE, 2006, pp. 112–117.

[56] A. Vollebregt, D. P. Hannessen, H. Hesselink, and J. Beetstra, "Iea/aie - modelling crew assistants with multi-agent systems in fighter aircraft," in Germany: Springer Berlin Heidelberg, Jun. 21, 2002, pp. 129–135. DOI: 10.1007/3-540-48035-8_13

[57] C.-N. Bodea and R.-I. Mogos, "A multi-agent system for resource allocation to education programmes in higher education institutions," *Journal of e-Learning & Higher Education*, pp. 1–9, Jan. 2, 2013. DOI: 10.5171/2013.261432

[58] L. Xiao, X. Shi, and H. Li, "Leader-following consensus of second-order multi-agent systems with switching topologies," in Springer International Publishing, Oct. 15, 2014, pp. 387–396. DOI: 10.1007/978-3-319-08377-3_38

[59] Z. Feng, G. Hu, and G. Wen, "Distributed consensus tracking for multiagent systems under two types of attacks," *International Journal of Robust and Nonlinear Control*, vol. 26, no. 5, pp. 896–918, Apr. 14, 2015. DOI: 10.1002/rnc.3342

[60] A. Gwiazda, A. Skala, and W. Bana, "Modeling of a production system using the multi-agent approach," *IOP Conference Series: Materials Science and Engineering*, vol. 227, no. 1, pp. 012 052–, Aug. 5, 2017. DOI: 10.1088/1757-899x/227/1/012052

[61] S. C. Silva, T. Carneiro, J. de Castro Lima, and R. R. Pereira, "Sigsim-pads - terrame hpa: Parallel simulation of multi-agent systems over smps," in *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, ACM, May 19, 2013, pp. 361–366. DOI: 10.1145/2486092.2486141

[62] G. S. Basheer, M. S. Ahmad, and A. T. Y. Chong, "A conflict classification and resolution modeling in multi-agent systems," in IGI Global, Jul. 31, 2014, pp. 6956–6965. DOI: 10.4018/978-1-4666-5888-2.ch685

[63] F.-x. Meng, Q.-l. Cai, and W. Zhang, "A multi-agent simulation system considering psychological stress for fire evacuation," in Springer Berlin Heidelberg, Jun. 14, 2013, pp. 1047–1054. DOI: 10.1007/978-3-642-38391-5_111

[64] P. Caillou, S. Aknine, and S. Pinson, "Searching pareto optimal solutions for the problem of forming and restructuring coalitions in multi-agent systems," *Group Decision and Negotiation*, vol. 19, no. 1, pp. 7–37, Nov. 12, 2009. DOI: 10.1007/s10726-009-9183-9

[65] R. Chandrasekar, V. Vijaykumar, and T. Srinivasan, "Probabilistic ant based clustering for distributed databases," in *2006 3rd International IEEE Conference Intelligent Systems*, IEEE, 2006, pp. 538–545.

[66] E. Alonso, "Foundations and applications of multi-agent systems - rights for multi-agent systems," *Lecture Notes in Computer Science*, pp. 59–72, Jul. 12, 2002. DOI: 10.1007/3-540-45634-1_5

[67] G. Ali, "Formal modeling towards a dynamic organization of multi-agent systems using communicating x-machine and z-notation," *Indian Journal of Science and Technology*, vol. 5, no. 7, pp. 1–6, Jul. 20, 2012. DOI: 10.17485/ijst/2012/v5i7.13

[68] K. Tuyls and A. Nowé, "Evolutionary game theory and multi-agent reinforcement learning," *The Knowledge Engineering Review*, vol. 20, no. 1, pp. 63–90, Dec. 1, 2005. DOI: 10.1017/s026988890500041x