sustainability

MDPI

Article

Advancing Sustainable Manufacturing: Reinforcement Learning with Adaptive Reward Machine Using an Ontology-Based Approach

Fatemeh Golpayegani [1], Saeedeh Ghanadbashi [1,*] and Akram Zarchini [2]

[1] School of Computer Science, University College Dublin, D04 V1W8 Dublin, Ireland; fatemeh.golpayegani@ucd.ie
[2] Department of Computer Engineering, Sharif University of Technology, Tehran 11155-9517, Iran; zarchiniakram1986@gmail.com
* Correspondence: saeedeh.ghanadbashi@ucd.ie

Abstract: Sustainable manufacturing practices are crucial in job shop scheduling (JSS) to enhance the resilience of production systems against resource shortages and regulatory changes, contributing to long-term operational stability and environmental care. JSS involves rapidly changing conditions and unforeseen disruptions that can lead to inefficient resource use and increased waste. However, by addressing these uncertainties, we can promote more sustainable operations. Reinforcement learning-based job shop scheduler agents learn through trial and error by receiving scheduling decisions feedback in the form of a reward function (e.g., maximizing machines working time) from the environment, with their primary challenge being the handling of dynamic reward functions and navigating uncertain environments. Recently, Reward Machines (RMs) have been introduced to specify and expose reward function structures through a finite-state machine. With RMs, it is possible to define multiple reward functions for different states and switch between them dynamically. RMs can be extended to incorporate domain-specific prior knowledge, such as task-specific objectives. However, designing RMs becomes cumbersome as task complexity increases and agents must react to unforeseen events in dynamic and partially observable environments. Our proposed Ontology-based Adaptive Reward Machine (ONTOADAPT-REWARD) model addresses these challenges by dynamically creating and modifying RMs based on domain ontologies. This adaptability allows the model to outperform a state-of-the-art baseline algorithm in resource utilization, processed orders, average waiting time, and failed orders, highlighting its potential for sustainable manufacturing by optimizing resource usage and reducing idle times.

Keywords: adaptive reward function; dynamic environments; multi-objective; ontology; partially observable environments; reinforcement learning; reward machines; sustainable manufacturing; job shop scheduling

check for updates

Citation: Golpayegani, F.; Ghanadbashi, S.; Zarchini, A. Advancing Sustainable Manufacturing: Reinforcement Learning with Adaptive Reward Machine Using an Ontology-Based Approach. *Sustainability* **2024**, *16*, 5873. https://doi.org/10.3390/su16145873

Academic Editors: Kumaran Kadirgama and Navid Aslfattahi

Received: 10 June 2024
Revised: 2 July 2024
Accepted: 5 July 2024
Published: 10 July 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

Nowadays, energy consumption has become an important issue due to its significant environmental impact, attracting much attention. According to studies [1], China's and India's current energy consumption will double by 2040. The industrial sector accounted for 37% of global energy use in 2022, resulting in the emission of 9.0 Gt (Gigatonne) of CO_2, which constitutes a quarter of total CO_2 emissions from the global energy system [2]. Thus, manufacturing enterprises urgently need sustainable practices to achieve economic, ecological, and social goals. For instance, Ref. [3] proposed a low-carbon scheduling model to optimize productivity and energy efficiency in job-shop environments, while Ref. [4] discussed the importance of sustainability assessment in manufacturing organizations. Additionally, Refs. [5,6] highlighted the use of advanced techniques such as multi-agent learning and smart manufacturing technologies to enhance sustainability. Job

Shop Scheduling (JSS) plays a crucial role in manufacturing by optimizing production processes, ensuring efficient resource utilization, and meeting delivery deadlines. It helps to minimize production costs, reduce lead times, and improve overall productivity, ultimately contributing to the competitiveness and success of manufacturing operations. Effective JSS ensures that resources, such as machines, are allocated optimally to meet the manufacturing environment's diverse and dynamic demands [7]. Effective job scheduling strategies have demonstrated the potential to decrease energy consumption and emissions through optimized production sequences, reduced idle times, and enhanced resource utilization [8,9]. They maximize resource utilization by intelligently allocating jobs to machines based on current workloads and machine availability, thereby reducing bottlenecks and idle times. This efficient allocation not only maximizes the use of available resources but also minimizes energy consumption and operational costs by ensuring that machines are operating at optimal capacity.

In dynamic and partially observable JSS environments, facing unforeseen situations is inevitable. For example, an urgent customer order with a tight deadline arrives suddenly, necessitating a reorganization of the current schedule to accommodate this priority job. Also, the observations made by the job shop scheduler agent are frequently derived from incomplete, ambiguous, and noisy data. For example, sensors tracking machine status could provide erroneous information about machine conditions. In such environments, the agent needs to be adaptive to emerging requirements or changes in the availability of the required resources. Reinforcement Learning (RL) methods [10–12] have gained attention for their adaptability to learning and optimizing schedules in dynamic scenarios. In RL, the agent learns optimal behavior through trial-and-error interactions with its environment, with the reward function, defined by the problem designer, serving as a crucial guide to incentivize desired actions while avoiding unintended outcomes. RL agents can continuously update schedules based on feedback from the environment. However, RL algorithms struggle with partial observability, as scheduling agents often operate with incomplete information about the factory's state [13,14]. Furthermore, the complexity of the state space in JSS can hinder efficient exploration and learning, especially when unforeseen events introduce new states or transitions [15,16]. Specifying appropriate reward functions to guide scheduling decisions in the face of unforeseen events is challenging, and maintaining a balance between exploration and exploitation becomes intricate in the presence of uncertainty. Additionally, generalization from past experiences to novel, unexpected scenarios is challenging. Furthermore, integrating human expertise in the decision-making process adds further complexity to managing unforeseen events in JSS within an RL framework [17,18]. On the other hand, non-RL approaches often rely on various techniques such as heuristics, simulation, and optimization to address the challenges of unforeseen events. For instance, Ref. [19] proposed an adaptive scheduling algorithm for minimizing makespan in dynamic JSS, while Ref. [16] introduced a heuristic algorithm for similar problems. Additionally, Ref. [20] presented a linear programming-based method for JSS, and Ref. [21] suggested a dynamic scheduling approach based on a data-driven genetic algorithm. However, these approaches may struggle to effectively model and optimize for dynamic changes or unforeseen disruptions and variations (face of varying task duration, resource availability, and unexpected factors), as they often rely on static predefined strategies and lack the adaptability that RL techniques offer in response to evolving conditions. For example, traditional methods frequently utilize neighborhood-based approaches, genetic programming, and fuzzy logic, but these can be limited in handling dynamic JSS problems [22,23]. Moreover, despite advances in self-adaptive multi-objective algorithms, challenges remain in optimizing for both performance and sustainability under uncertain conditions [24,25].

Creating reward functions often demands domain expertise and can be time-consuming, leaving room for errors and limited coverage of all scenarios. Moreover, these functions struggle to generalize across different scheduling environments and cope with the inherent uncertainty present in JSS, such as unforeseen events. The literature proposes various techniques to handle these challenges. To provide more frequent feedback on

appropriate behaviors, agents can be incentivized with intrinsic rewards [26]. Additionally, inner rewards can be generated using various reward shaping approaches, such as adaptive inner-reward shaping [27], bi-level optimization for shaping rewards [28], and context-sensitive reward shaping in multi-agent systems [29]. Dynamic reward shaping with user feedback is another effective method [30]. However, these techniques are challenging in dynamic and partially observable environments, especially when faced with unexpected events. Inverse reward design and reward learning from demonstration [31,32] infer the intents or goals of an expert demonstrating a task; however, these approaches only work when the agent has access to expert demonstrations.

Reward Machine (RM) is defined as a Finite-State Automata (FSA). It is a way to expose the reward function structure to the agent (i.e., the way it assigns rewards to different states and actions) to use it to decompose the problem and speed up learning [33]. The intuition is that the agent will be rewarded with different reward functions at different times, depending on the changes made in the RM. Partially observable and dynamic environments can present challenges for RM design, mainly when complex tasks and unexpected events occur [34]. Although various variations of RM have been proposed, such as learning RM in partially observable environments [35], hierarchies of RMs [36], and symbolic RMs [37], to define reward functions for complex tasks, they cannot still define reward functions adaptively and require manual reward assignment.

We address this gap by integrating ontology with RM, aiming to enhance the JSS and sustainability in manufacturing. Ontologies explicitly represent human knowledge about domains of interest and serve as critical components in knowledge management applications [38]. The hierarchical structure of ontologies, with subclasses inheriting the properties from their ascendant classes, facilitates inferring the environment's properties. As the environment changes, new concepts (entities), properties, and relations may become apparent. Each concept can be associated with positive or negative properties, dependent on the domain being modeled by the ontology. Positive properties are desirable or valuable, whereas negative properties are undesirable or harmful or lead to negative outcomes. For example, in the domain of JSS, positive properties might represent a low order waiting time or high machine working time, while negative properties might represent a high machine idle time or high machine failure rate. In ontology, the properties are structured and related to other concepts within the domain, making it possible to reason about them and draw conclusions based on their relations. In our previous work [39], we introduced a new Ontology-based Adaptive Reward Function (OARF) method, which dynamically creates new reward functions using positive or negative properties to adapt the agent's behavior based on the changing dynamics of the environment. The OARF method is extensively extended in this work by proposing an ONTOlogy-based ADAPTive REWARD Machine (ONTOADAPT-REWARD) model to dynamically/adaptively create/modify RMs. First, the RL agent models its observation as an ontology-based schema that maps low-level sensor data to high-level concepts, relations, and properties. This schema allows the agent to access the structure of ontological knowledge of the observation and to conduct logical reasoning based on this knowledge. Based on the new concepts and properties of the agent's observation, the agent identifies unforeseen events by extracting subsumers and creating new transitions and reward functions for its RM. Finally, the agent can use multiple reinforcement learners for different reward functions.

The rest of this paper is organized as follows. Section 2 examines related works. Section 3 provides the required background information, and Section 4 describes JSS as a partially observable and dynamic environment. Section 5 describes our model and its different stages. In Section 6, the scenarios are defined, and the results are analyzed. Section 7 concludes with our findings and outlines future work.

2. Related Work

Static scheduling in JSS involves the creation of a fixed production schedule that remains unchanged throughout the manufacturing process. This approach assumes a stable

and predictable manufacturing environment where all parameters are known in advance. Each job/order or operation is assigned to a specific machine and a predefined processing sequence based on factors like job priority, due dates, and setup times. Although static scheduling methods are relatively straightforward to implement and require less computational effort, they lack adaptability [40]. They need to account for real-time changes, such as machine breakdowns or shifts in job priorities, which can lead to suboptimal resource utilization and missed delivery deadlines. On the contrary, dynamic scheduling allows for real-time adjustments to the production schedule in response to changing conditions. It accommodates uncertainty and is well suited for modern manufacturing environments characterized by fluctuations in demand, machine failures, and evolving priorities. Dynamic scheduling systems continuously monitor the shop floor, collecting data on job progress, machine status, and unforeseen disruptions. Using this real-time information, dynamic scheduling algorithms can adapt the schedule to optimize resource utilization, minimize delays, and meet customer deadlines. This flexibility comes at the cost of increased complexity and computational requirements but it is essential for achieving agility and responsiveness in dynamic manufacturing settings [41,42]. Dynamic scheduling in job shops can be approached as an RL problem. The scheduler (agent) takes actions (schedule adjustments) to maximize a cumulative reward signal (e.g., minimizing production time, meeting deadlines) [15,43].

Adaptive reward functions are crucial in JSS due to the dynamic and complex nature of the environment. They allow scheduling algorithms to respond effectively to changing conditions, such as machine breakdowns or shifting job priorities, ensuring optimal scheduling decisions. Additionally, these functions enable the algorithm to balance multiple objectives, like minimizing production time and reducing costs, by dynamically adjusting the importance of each objective based on the current context, ultimately improving scheduling performance. Substantial efforts have been made to enrich the information in reward functions. RMs from [44] directly represent reward functions as FSAs. There have been efforts to learn the so-called perfect RM in [35] from the experience of an RL agent in a partially observable environment. They formulated finding a perfect RM problem as an optimization problem. Their method focuses on the RM that predicts what is possible and impossible in the environment at the abstract level. However, their method still needs to know a set of high-level propositional symbols and a labeling function that can detect them prior. So it does not work when there is unforeseen observation for which there is no experience in the training set. Learning Finite State Controllers (FSCs), [45] suggested looking for the smallest RM that correctly mimics the external reward signal given by the environment and whose optimal policy receives the most reward. It is a desirable property for RMs, but it requires computing optimal policies to compare their relative quality, which seems prohibitively expensive. With Deterministic Markov Models (DMMs), [46] proposed learning the RM that remembers sufficient information about history to make accurate predictions about the next observation. However, their approach needs to keep track of details that will not necessarily result in better predictions, especially in dynamic and noisy environments. In [36], the authors defined Hierarchies of RMs (HRMs), in which RM and its associated policies are reusable across several RMs. Ref. [37] proposed Symbolic RMs (SRMs) for incorporating high-level task knowledge when specifying the reward functions. It facilitates the expression of complex task scenarios and the specification of symbolic constraints. Ref. [37] developed a hierarchical Bayesian approach that can concretize an SRM by inferring appropriate reward assignments from expert demonstrations. Also, the Inverse Reward Design (IRD) proposed in [32] aims at inferring a reward function from an expert demonstration. However, these approaches only work when the agent can access expert demonstrations.

Knowledge-based RL seeks to guide agents in their exploration using prior knowledge. In [47], we proposed a new automatic goal generation model to evolve/generate new goals to address emerging requirements in unforeseen situations. However, a new goal could only be defined as returning the environment to a previous normal state. Also, we

have proposed a new Ontology-based Adaptive Reward Function (OARF) method, which dynamically creates new reward functions based on domain ontologies [39]. However, we do not handle the design of RMs when dealing with increasingly complex tasks and unforeseen events in dynamic and partially observable environments. Reward shaping proposed by [48] added state-based potentials to the reward in each state. Exploration-driven approaches such as [26] incentivize agents with intrinsic rewards. Ref. [49] applied reward shaping approaches to multi-objective problems. However, it is not always apparent how general concepts describing intended behaviors can be translated into meaningful numeric rewards [50]. Ref. [27] proposed learning-based reward shaping to allow the agent to generate inner rewards to guide itself. However, their learning approach has challenges with a limited number of training steps and uncertain environments. Ref. [51] proposed a meta-learning framework to learn efficient reward shaping on newly sampled tasks automatically. However, they assumed that the state spaces of the task distribution are shared. Ref. [52] used genetic programming to find novel reward functions that improve learning system performance. Ref. [28] considered how to adaptively utilize a given parameterized shaping reward function, utilizing the beneficial part of the given shaping reward function as much as possible and meanwhile ignoring the non-beneficial shaping rewards. Ref. [29] described the use of context-aware potential functions in which different shaping functions should be used for different contexts to address different subgoals. However, the agent must autonomously learn the mapping from context to shaping function. Ref. [30] introduced a dynamic reward shaping approach in which a human provides verbal feedback translated into additional rewards. In [53], demonstrations from a teacher were used to shape a potential reward function by training a deep convolutional neural network. However, humans sometimes give the wrong reward, give feedback out of time, or the agent's speech recognition system misinterprets the command.

3. Background

3.1. Markov Decision Process

By providing a discrete-time stochastic control process, a Markov Decision Process (MDP) can mathematically describe an environment in RL [54]. In the standard formulation of MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, r, p, \gamma)$, at time step $t \geq 0$, an agent is in state $s^t \in \mathcal{S}$, takes an action $a^t \in \mathcal{A}$, receives an immediate reward $r = r(s^t, a^t) \in \mathbb{R}$, and transitions to a next state $s^{t+1} \sim p(\cdot \mid s^t, a^t) \in \mathcal{S}$. Let $\pi : \mathcal{S} \mapsto P(\mathcal{A})$ be a policy, where $P(\mathcal{A})$ is the set of distributions over the action space $\mathcal{A}$. The discounted cumulative reward under policy π is $J(\pi) = \mathbb{E}_\pi\left[\sum_{t=0}^{\infty} \gamma^t r\right]$, where $\gamma \in [0, 1)$ is a discount factor. The goal of RL is to find a policy π that achieves the maximum cumulative reward $\pi^* = \arg\max_\pi J(\pi)$. Under policy π, the value function can be defined as the expected cumulative reward of an agent in a certain state s, $V^\pi(s) = \mathbb{E}_\pi[J(\pi) \mid s_0 = s, a_0 \sim \pi(\cdot \mid s_0)]$. The action value function returns the expected cumulative reward for using action a in a certain state s, $Q^\pi(s, a) = \mathbb{E}_\pi[J(\pi) \mid s_0 = s, a_0 = a]$. Given a certain state s, the advantage function measures how much a certain action a is a good or bad decision, $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$.

Partially Observable MDPs: In practice, most real-world problems are not fully captured by MDPs, and some are partially observable. A Partially Observable MDP (POMDP) has the following equation: $(\Omega, \mathcal{S}, \mathcal{A}, r, p, \mathcal{W}, \gamma)$ [55]. POMDP has two extra components: the possibly infinite set Ω of observations, and the $\mathcal{W} : \mathcal{S} \to \Omega$ function, which creates observations o^t based on the unobservable state s^t using a set of conditional observation probabilities. At each time, the agent receives an observation $o^t \in \Omega$, with probability $\mathcal{W}(o^t, a^t, s^t)$, that depends on the recently changed environment's state s^t and on the just taken action a^t. Due to partial observability, learning is challenging in these circumstances since the same observation might be received from two separate states, each requiring a different optimal action.

3.2. Trust Region Policy Optimization

With a direct policy search within the specified policy class $\pi_\theta, \theta \in \Theta$, where Θ stands for the policy's parameter space, it is possible to approximate optimal policy π^*. We can update the parameter θ using policy gradient ascent, by computing $\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} A^{\pi_\theta}(s^t, a^t) \nabla_\theta \log \pi_\theta(a^t \mid s^t)\right]$, then updating $\theta_{\text{new}} \leftarrow \theta + \alpha \nabla_\theta J(\pi_\theta)$ with some learning rate $\alpha > 0$. Another alternative would be to consider first a trust region optimization problem [54]:

$$\begin{aligned} \max_{\theta_{\text{new}}} & \mathbb{E}_{\pi_\theta}\left[\frac{\pi_{\theta_{\text{new}}}(a^t \mid s^t)}{\pi_\theta(a^t \mid s^t)} A^{\pi_\theta}(s^t, a^t)\right] \\ & \|\theta_{\text{new}} - \theta\|_2 \leq \epsilon \end{aligned} \tag{1}$$

for some $\epsilon > 0$. If we approximate the objective of the (1) linearly, $\mathbb{E}_{\pi_\theta}\left[\frac{\pi_{\theta_{\text{new}}}(a^t|s^t)}{\pi_\theta(a^t|s^t)} A^{\pi_\theta}(s^t, a^t)\right] \approx \nabla_\theta J(\pi_\theta)^T(\theta_{\text{new}} - \theta)$, by correctly selecting ϵ given α, we are able to recover the policy gradient update.

In this paper, the baseline algorithm uses Trust Region Policy Optimization (TRPO) [54] as its learning algorithm. TRPO captures the geometry of the parameter space influenced by the underlying distributions more effectively by employing information theoretic constraints rather than Euclidean constraints (as in (1)). Specifically, consider the following trust region formulation:

$$\begin{aligned} \max_{\theta_{\text{new}}} & \mathbb{E}_{\pi_\theta}\left[\frac{\pi_{\theta_{\text{new}}}(a^t \mid s^t)}{\pi_\theta(a^t \mid s^t)} A^{\pi_\theta}(s^t, a^t)\right] \\ & \mathbb{E}_s[\mathbb{KL}[\pi_\theta(\cdot \mid s) || \pi_{\theta_{\text{new}}}(\cdot \mid s)]] \leq \epsilon, \end{aligned} \tag{2}$$

where the state visitation distribution induced by π_θ is denoted by $\mathbb{E}_s[\cdot]$. By enforcing the trust region with the KL divergence, the update in (2) optimizes a lower bound of $J(\pi_\theta)$ during training to prevent making significant changes that irreversibly worsen policy performance as with the (1) for vanilla policy gradients.

3.3. Reward Machine

A Reward Machine (RM) [44] takes abstract descriptions of the environment as input and output reward functions. The assumption is that, according to the transitions made in the RM, the agent will be rewarded by various reward functions at various times. An RM sets several tasks for an RL agent in the environment and is defined over a collection of propositional symbols E that is related to a set of high-level events from the environment that the agent may detect using a labeling function $L : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to 2^E$. Given an environment experience $= (s, a, s')$, L assigns truth values to symbols in E, where s' is the observation observed after performing the action a when observing s. An RM is a tuple $\mathcal{R} = (U, u_0, \delta_u, \delta_r)$, where U is a finite set of states, $u_0 \in U$ is an initial state, δ_u is the state-transition function $\delta_u : U \times 2^E \to U$, and δ_r is the reward-transition function $\delta_r : U \times U \to [S \times A \times S \to \mathbb{R}]$.

3.4. Ontology

Ontology [56] is utilized for semantic descriptions when needed (for instance, when an interpretation of an unanticipated or unforeseen event is needed). An ontology describes concepts C, properties F, and relations R. For example, the concept "Order" can be defined as an Order $\sqsubseteq$ Thing $\sqcap \exists$has.WaitingTime within the JSS ontology. It denotes that the concept "Order" is a Thing (that is, relation) and possesses the property "WaitingTime". An ontology can represent all relations between concepts using a hierarchical approach, including conditional relations, multiple relations, and the inverse relations between successor and predecessor. The domain and range of a relation determine the concepts' types and ranges. Instances are the "concrete things" represented by a concept; for example,

"automobile repair" is an instance of the concept "Order". To account for all the concept's relations, we need to look at all (possibly complex) entailed descriptions of that concept. This is called subsumers [57]. Suppose that we have these relations {WaitingTime $\sqsubseteq$ Time, Order $\sqsubseteq$ Thing $\sqcap$ $\exists$has.WaitingTime} in the ontology. Then, the subsumer for the concept Order will be {Thing, Time, $\exists$has.WaitingTime}.

4. Job Shop Scheduling: Partially Observable and Dynamic Environment

JSS refers to the problem of processing multiple orders on multiple machines, with several operational steps required for each order completed in a particular sequence. For example, the order might involve manufacturing consumer products such as automobiles. Figure 1 illustrates this environment. The JSS environment is selected to illustrate how our proposed model can work in real-world applications. Dynamism and partial observability are essential characteristics of the JSS environment. Unforeseen events like machine failures, extended processing times, or urgent orders can significantly impact the scheduling process [58,59].

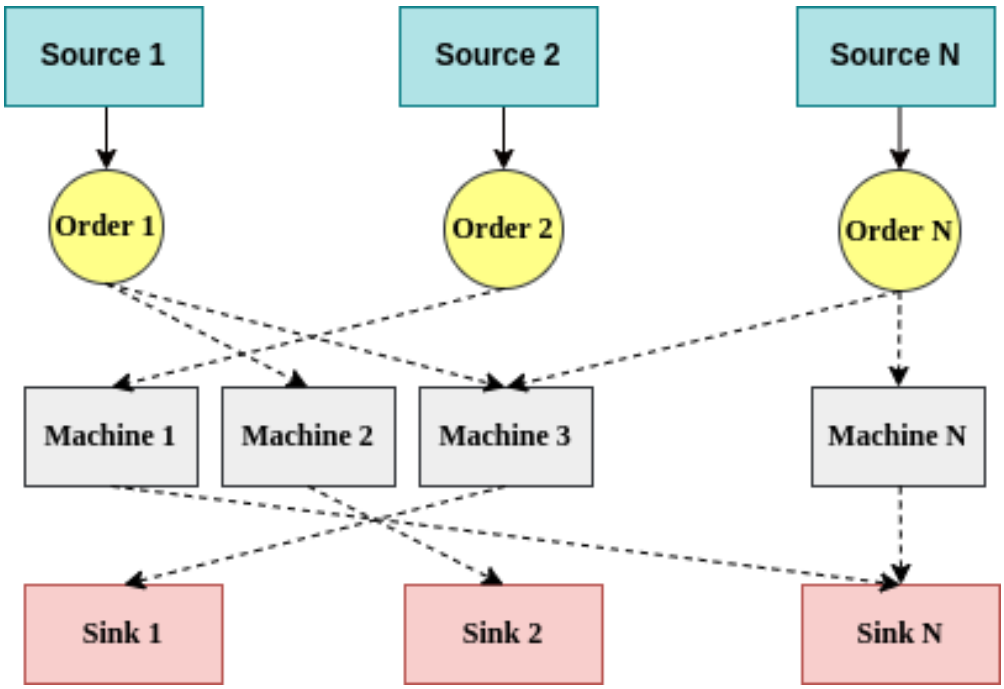


Figure 1. Abstract view of the job shop scheduling environment. Dotted lines indicate possible operations.

We have based our evaluation on a JSS environment (see Figure 2) using [60] with the following details.

JSS Entities include the following:

- Orders $\{d_0, d_1, \ldots, d_i\}$ should be processed by machines based on a specified sequence of operations.
- Sources I_1, I_2, and I_3 generate orders.
- Machines/Resources $\{m_0, m_1, \ldots, m_{15}\}$ can process one order at a time. Machines are categorized into three groups: n_1, n_2, and n_3. Each group consists of machines that can perform similar operations, following the Group Technology (GT) principle of forming part families based on process similarities. This grouping allows for streamlined processing of orders, as machines within the same group can be interchangeably used for specific types of operations, thereby reducing setup times and enhancing throughput.
- Work areas z_1, z_2, and z_3 are located in different locations in the factory to facilitate the efficient movement of orders. Each work area contains a subset of machine groups, ensuring that orders can be processed with minimal transportation delays between operations.

- Sinks x_1, x_2, and x_3 are entities that consume processed orders, and their capacity is the number of orders they consume at each time step.
- The scheduler agent selects an order and sends it to a machine/sink for its next operation step. The steps involved are as follows: The agent selects an order from the queue of orders at the source. Then, it moves to a work area and a group and selects a machine in the desired group to process the order. Finally, it selects an order from the queue of orders in the machine and places the processed order in a sink.

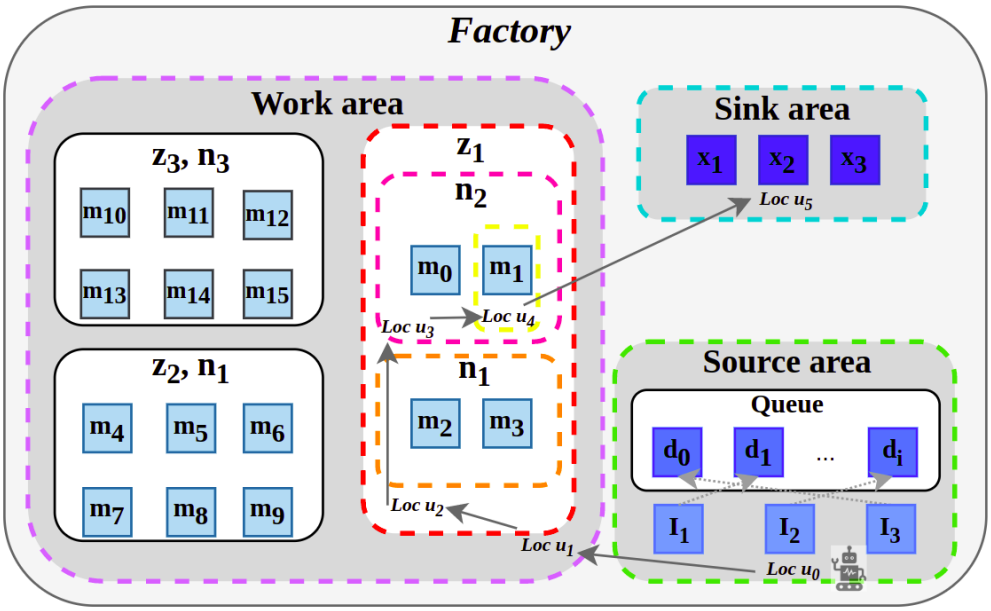


Figure 2. Simulated job shop scheduling environment.

JSS Tasks: There are different tasks associated with every factory location that must be performed sequentially by the JSS agent. We assume our factory has six different locations $\{u_0, \ldots, u_5\}$ and the tasks associated with each location are as follows:

- τ_{u_0} selects an order from the source order queue.
- τ_{u_1} selects a work area for moving the order there.
- τ_{u_2} selects a group for selecting a machine from it.
- τ_{u_3} selects a machine for processing the order.
- τ_{u_4} selects an order from the machine order queue to be processed.
- τ_{u_5} selects a sink to consume the processed order.

JSS Strategies: In the current implementation of JSS (the code of the baseline method and ONTOADAPT-REWARD-based proposed method is accessible to the public: https://github.com/akram0618/ontology-based-adaptive-reward-machine-RL (accessed on 1 January 2024)), the following strategies are used to complete the described tasks:

- Random strategy for tasks τ_{u_1}, τ_{u_2}, and τ_{u_5}, which selects a work area, group, or sink randomly;
- LIFO strategy for task τ_{u_0}, which selects the last order entered into the queue for resource allocation;
- FIFO strategy for task τ_{u_4}, which selects the first order entered into the queue for processing;
- RL strategy for task τ_{u_3}, which selects the machine with the highest reward.

Figure 2 illustrates the JSS environment. The factory is divided into three areas: sink, source, and work. When the JSS agent is located in each area, it can only observe the entities that are specifically located in that area. For example, in location u_0, only I_1, I_2, and I_3 and their order queue are visible (i.e., the green dotted rectangle area). When the agent reaches the location u_1, it can only observe work areas z_1, z_2, and z_3 (i.e., the purple dotted rectangle area), while at the location u_2, it observes groups n_1 and n_2 (i.e., the red rectangle

area). Location u_3 is where the agent can observe machines m_0 and m_1 (the pink dotted rectangle), whereas location u_4 is where it can observe only machine m_1 and its order queue (the yellow dotted rectangle). Finally, in location u_5, the agent observes sinks x_1, x_2, and x_3 (i.e., the blue dotted rectangle area).

5. An Ontology-Based Adaptive Reward Machine for Reinforcement Learning Algorithms

In ONTOlogy-based ADAPTive REWARD Machine (ONTOADAPT-REWARD) model, firstly, the JSS agent models its state as an ontology-based schema. Next, based on the concepts and properties of the agent's state, it extracts a relevant subsumer and labels its observation as a new propositional symbol. An ontology-based schema is also used to derive reward functions for each state (see ONTOADAPT-REWARD process in Figure 3 and the algorithmic procedure in Algorithm 1).

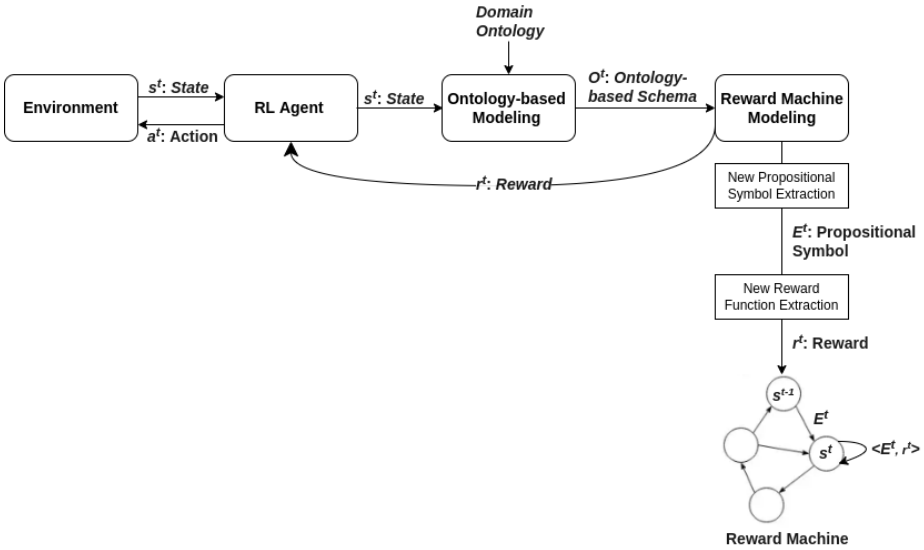


Figure 3. Ontology-based Adaptive Reward Machine (ONTOADAPT-REWARD) model.

Algorithm 1 Ontology-based Adaptive Reward Machine (ONTOADAPT-REWARD).

1: **Input:** $\{\tau_{u_0}, \ldots, \tau_{u_i}\}$, $s^t = \{s_{\tau_{u_0}}, \ldots, s_{\tau_{u_i}}\}$, $\{\mathcal{A}_{\tau_{u_0}}, \ldots, \mathcal{A}_{\tau_{u_i}}\}$, $\{r_{\tau_{u_0}}, \ldots, r_{\tau_{u_i}}\}$ //RL Formulation
2: **for** $s_{\tau_{u_i}}$ in s^t **do**
3: $s_{\tau_{u_i}} -> O^t = (C, F, R)$ //Ontology-based Modeling
4: $O^t -> E_{\tau_{u_i}}$ //New Propositional Symbol Extraction
5: $O^t -> r_{\tau_{u_i}}$ //New Reward Function Extraction
6: $\mathcal{M}_{\tau_{u_i}} = (s_{\tau_{u_i}}, \mathcal{A}_{\tau_{u_i}}, r_{\tau_{u_i}}, p_{\tau_{u_0}}, \gamma_{\tau_{u_0}})$ //Learning
7: **end for**

5.1. JSS Ontology

To represent the relevant concepts in the JSS environment, we use the ontology shown in Figure 4, with seven high-level concepts (i.e., superclass), including Belief, Source, Order, WorkArea, Group, Machine, and Sink, each associated with an individual entity defined within the JSS environment. A superclass can have subclasses shown by the "part of" relation. Subclasses can represent concepts more specific than those of a superclass. For example, Working, Failure, and Idle are part of Status, which is part of Machine. Belief is based on human experiences with concepts; for example, WorkingTime is associated with positive belief, and WaitingTime is associated with negative belief. It means that high waiting time has a negative impact on the agent's performance.

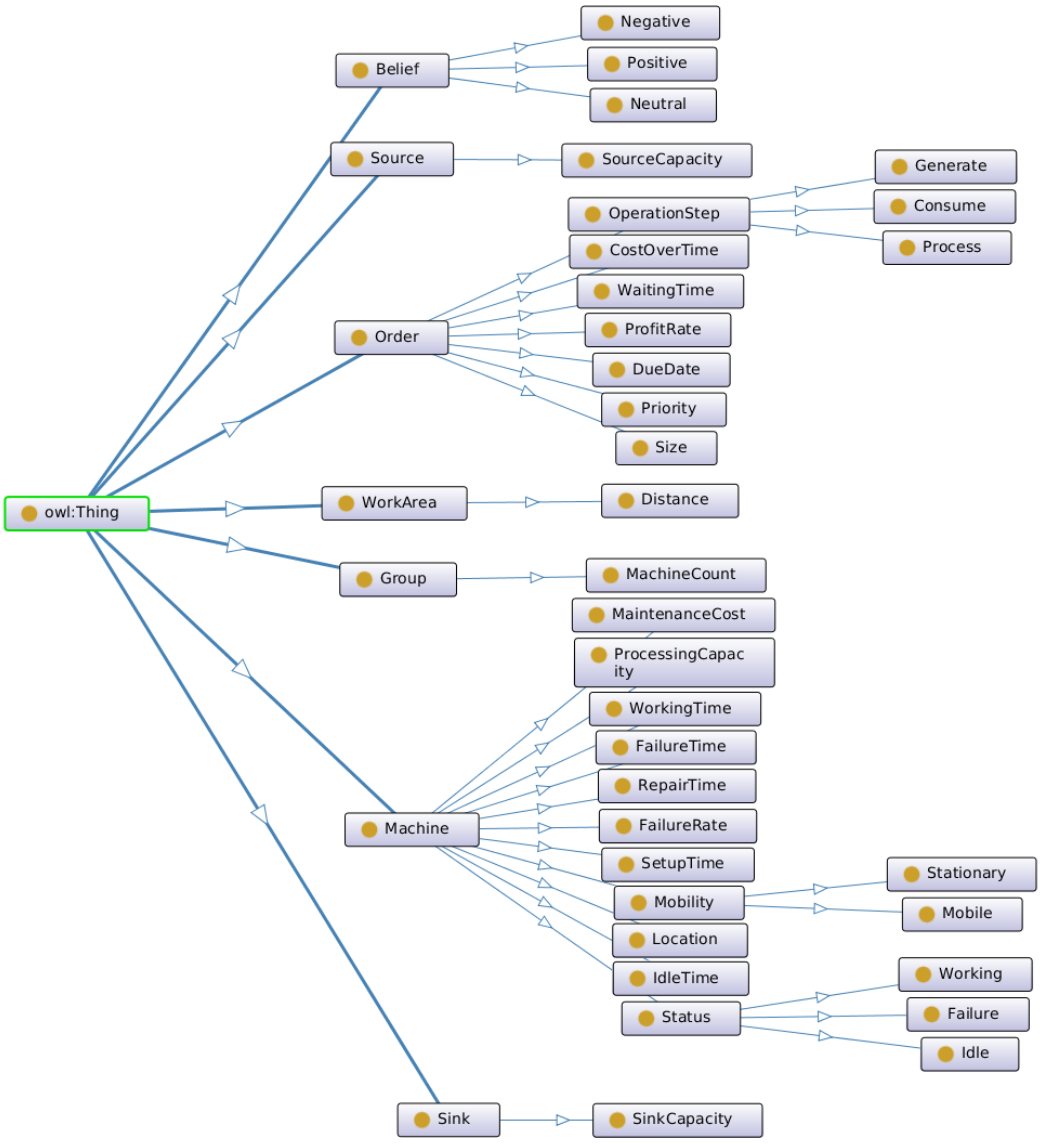


Figure 4. Job shop scheduling environment ontology.

5.2. Modeling JSS as an RL Process

To define RMs for the JSS problem, we need to define actions, states, and reward functions for each task within the environment to formulate them as RL problems (see line 1 of Algorithm 1).

Actions: There are different action sets for different JSS tasks:

- $\mathcal{A}_{\tau_{u_0}} = \{d_0, \ldots, d_i\}$, selecting one of the orders in the source order queue.
- $\mathcal{A}_{\tau_{u_1}} = \{z_1, z_2, z_3\}$, selecting one of the three work areas.
- $\mathcal{A}_{\tau_{u_2}} = \{n_1, n_2, n_3\}$, selecting one of the three groups of machines.
- $\mathcal{A}_{\tau_{u_3}} = \{m_0, \ldots, m_{15}\}$, selecting one of the sixteen machines.
- $\mathcal{A}_{\tau_{u_4}} = \{d_0, \ldots, d_5\}$, selecting one of the six orders in the machine order queue.
- $\mathcal{A}_{\tau_{u_5}} = \{x_1, x_2, x_3\}$, selecting one of the three sinks.

State: We represent the information of each state s^t for the JSS agent at time step t using Equation (3). Specific parameters are detailed in Table 1.

$$s^t = \{M^t, D^t, Z^t, N^t, X^t, w^t_{d_i}, v^t_{d_i}, y^t_{d_i}, b^t_{m_i}, q^t_{m_i}, l^t_{m_i}, j^t_{m_i}, k^t_{z_i}, h^t_{n_i}, e^t_{x_i}\} \tag{3}$$

Table 1. State representation parameters.

Parameter	Information
M^t	Machines at time step t.
D^t	Orders at time step t.
Z^t	Work areas at time step t.
N^t	Groups at time step t.
X^t	Sinks at time step t.
$w^t_{d_i}$	The waiting time of order d_i represents the duration in seconds that the order has waited to be completely processed by a machine at time step t.
$v^t_{d_i}$	The due date of order d_i shows a required/preferable time of the order completion. It can have the following values: Low due date (1000 s), medium due date (3000 s), and high due date (5000 s).
$y^t_{d_i}$	The priority of order d_i indicates the importance of the order at time step t, which can include low, medium, or high priorities.
$b^t_{m_i}$	The working time of machine m_i indicates the total processing time of the machine at time step t.
$q^t_{m_i}$	The idle time of machine m_i shows a period of time in which the machine is available but is not doing anything productive.
$l^t_{m_i}$	The failure time of machine m_i is determined by summing the total time it was in a failure status at time step t.
$j^t_{m_i}$	The failure rate of machine m_i would be equivalent to the failure probability of the machine at time step t, including low, medium, or high values.
$k^t_{z_i}$	The distance of a work area z_i from the scheduler agent at time step t.
$h^t_{n_i}$	The machine count shows the number of machines in group n_i at time step t.
$e^t_{x_i}$	The sink capacity shows the capacity of sink x_i at time step t.

The JSS agent's state for different tasks is as follows:

- $s_{\tau_{u_0}} = \{D^t, w^t_{d_i}, v^t_{d_i}, y^t_{d_i}\}$.
- $s_{\tau_{u_1}} = \{M^t, Z^t, N^t, b^t_{m_i}, q^t_{m_i}, l^t_{m_i}, j^t_{m_i}, k^t_{z_i}, h^t_{g_i}\}$.
- $s_{\tau_{u_2}} = \{M^t, N^t, b^t_{m_i}, q^t_{m_i}, l^t_{m_i}, j^t_{m_i}, h^t_{g_i}\}$.
- $s_{\tau_{u_3}} = \{M^t, D^t, w^t_{d_i}, v^t_{d_i}, y^t_{d_i}, b^t_{m_i}, q^t_{m_i}, l^t_{m_i}, j^t_{m_i}, h^t_{g_i}\}$.
- $s_{\tau_{u_4}} = \{M^t, D^t, w^t_{d_i}, v^t_{d_i}, y^t_{d_i}, b^t_{m_i}, q^t_{m_i}, l^t_{m_i}, j^t_{m_i}\}$.
- $s_{\tau_{u_5}} = \{X^t, e^t_{x_i}\}$.

Reward: The only predefined reward function is maximizing the average utilization rate of all the machines $\overline{E}$ as depicted in Equation (4):

$$\pi^* = \arg\max_{\pi} \overline{E} \tag{4}$$

Because a failed machine cannot receive an order until it is repaired, the individual machine's utilization percentage E_{m_i} is calculated based on its working time as follows (where t' indicates the last utilization rate calculation time):

$$E_{m_i} = b^t_{m_i} / (t - t' - l^t_{m_i}) \tag{5}$$

After that, the average utilization rate is calculated as follows:

$$\overline{E} = \frac{1}{|M|} \sum_{i=0}^{|M|-1} E_{m_i} \tag{6}$$

Other reward functions will be extracted in Section 5.4.2 based on the proposed ONTOADAPT-REWARD model.

5.3. Ontology-Based Modeling of State

This paper uses ontology to allow agents to represent their observations within an ontological structure, mapping low-level sensor data streams to high-level concepts. Sensors typically produce raw, unstructured data streams that measure phenomena such as the waiting time of an order. Data collected by sensor resources are described by the Semantic Sensor Network (SSN) ontology, which is a model developed using a standard model of sensor networks [61]. Agents use the SSN ontology with cross-domain knowledge to annotate and present sensor data using the data model described in [62] (see Figure 5). Sensor data observed by an agent are described by the "ssn:Observation" class. The "ssn:Property" specifies the property (e.g., waiting time) of the feature of interest (e.g., order) that is described by the JSS ontology.

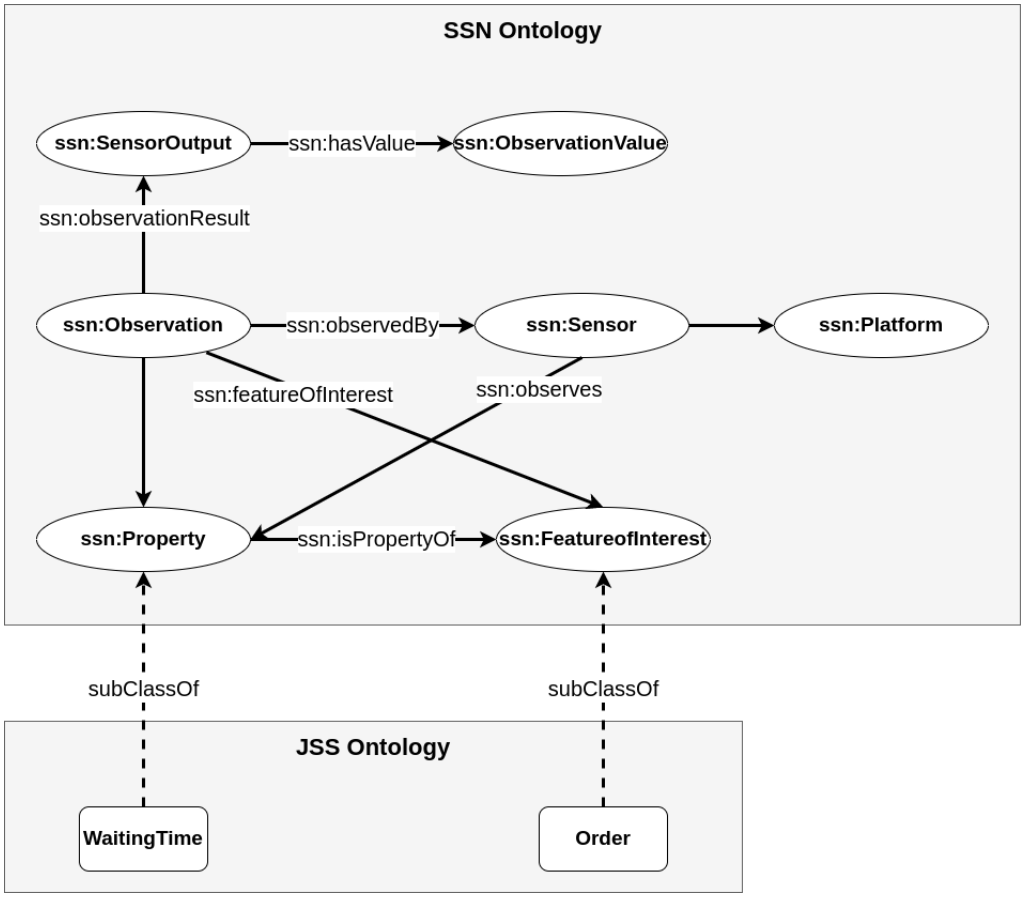


Figure 5. Concept of the data model proposed in [62] to annotate and present sensor data.

In this paper, the RL agent annotates raw environment data streams using a semantic description defined by a combined ontology, where the environment data stream is indicated by concepts (e.g., machine and order) and their properties (e.g., working time and waiting time). This is referred to as an ontology-based schema (see line 3 of Algorithm 1). Based on this schema, we can access the ontological knowledge structure of the state and its ability to conduct logical reasoning.

We define $O^t_{g_i} = (C, F, R)$ as the schema that describes the data monitored/observed by the JSS agent g_i at the time step t. C represents the set of concepts $C = \{c_1, c_2, \ldots, c_x\}$, F represents properties $F = \{f_1, f_2, \ldots, f_x\}$, and R represents the set of relations over these concepts, indicating how concepts and concepts/values are connected through properties

($R \subseteq C \times F \times C$). For example, in an ontology-based schema for the JSS environment, the concept "Order" is defined as follows:

Order $\sqsubseteq$ Thing $\sqcap$ $\exists$has.DueDate $\sqcap$ $\exists$has.Priority $\sqcap$ $\exists$has.WaitingTime $\sqcap$ $\forall$has.OperationStep $\sqcap$ $\exists$has.ProfitRate $\sqcap$ $\forall$has.Size $\sqcap$ $\exists$has.CostOverTime

This means the concept Order has different properties, including DueDate, Priority, WaitingTime, OperationStep, ProfitRate, Size, and CostOverTime. $\forall$ means that all concept instances possess the property, while $\exists$ means that some instances possess the property.

Depending on the task, the JSS agent may be in a different state (see Section 5.2) and observe different sets of concepts C and properties F when performing various tasks. For example, when the agent is in location u_0, it can only observe the concept "Order" and its properties (Refer to Figure 2):

$C_{\tau_{u_0}}$ = {Order}

$F_{\tau_{u_0}}$ = {WaitingTime, DueDate, Priority}

Consider an ontology-based schema for the JSS environment as follows:

{**NegativeBelief** $\sqsubseteq$ Belief,
PositiveBelief $\sqsubseteq$ Belief,
NeutralBelief $\sqsubseteq$ Belief,
WaitingTime $\sqsubseteq$ Time $\sqcap$ $\forall$has.NegativeBelief,
WorkingTime $\sqsubseteq$ Time $\sqcap$ $\forall$has.PositiveBelief,
SetupTime $\sqsubseteq$ Time $\sqcap$ $\forall$has.NegativeBelief,
IdleTime $\sqsubseteq$ Time $\sqcap$ $\forall$has.NegativeBelief,
FailureTime $\sqsubseteq$ Time $\sqcap$ $\forall$has.NegativeBelief,
RepairTime $\sqsubseteq$ Time $\sqcap$ $\forall$has.NegativeBelief,
Priority $\sqsubseteq$ Number $\sqcap$ $\forall$has.NegativeBelief,
Size $\sqsubseteq$ Number $\sqcap$ $\forall$has.NeutralBelief,
CostOverTime $\sqsubseteq$ Number $\sqcap$ $\forall$has.NegativeBelief,
MaintenanceCost $\sqsubseteq$ Number $\sqcap$ $\forall$has.NegativeBelief,
ProfitRate $\sqsubseteq$ Number $\sqcap$ $\forall$has.PositiveBelief,
Distance $\sqsubseteq$ Number $\sqcap$ $\forall$has.NegativeBelief,
Location $\sqsubseteq$ Number $\sqcap$ $\forall$has.NeutralBelief,
FailureRate $\sqsubseteq$ Number $\sqcap$ $\forall$has.NegativeBelief,
MachineCount $\sqsubseteq$ Number $\sqcap$ $\forall$has.PositiveBelief,
SourceCapacity $\sqsubseteq$ Number $\sqcap$ $\forall$has.PositiveBelief,
SinkCapacity $\sqsubseteq$ Number $\sqcap$ $\forall$has.PositiveBelief,
ProcessingCapacity $\sqsubseteq$ Number $\sqcap$ $\forall$has.PositiveBelief,
DueDate $\sqsubseteq$ Time $\sqcap$ $\forall$has.PositiveBelief,
OperationStep $\sqsubseteq$ Generate $\sqcup$ Process $\sqcup$ Consume $\sqcap$ $\forall$has.NeutralBelief,
Mobility $\sqsubseteq$ Mobile $\sqcup$ Stationary $\sqcap$ $\forall$has.NeutralBelief,
Status $\sqsubseteq$ (Working $\sqcap$ $\forall$has.PositiveBelief) $\sqcup$ (Failure $\sqcap$ $\forall$has.NegativeBelief) $\sqcup$ (Idle $\sqcap$ $\forall$has.NegativeBelief),
Belief $\sqsubseteq$ Thing $\sqcap$ PositiveBelief $\sqcup$ NegativeBelief $\sqcup$ NeutralBelief,
Source $\sqsubseteq$ Thing $\sqcap$ $\forall$has.SourceCapacity $\sqcap$ $\exists$has.Order,
Order $\sqsubseteq$ Thing $\sqcap$ $\exists$has.DueDate $\sqcap$ $\exists$has.Priority $\sqcap$ $\exists$has.WaitingTime $\sqcap$ $\forall$has.OperationStep $\sqcap$ $\exists$has.ProfitRate $\sqcap$ $\forall$has.Size $\sqcap$ $\exists$has.CostOverTime,
WorkArea $\sqsubseteq$ Thing $\sqcap$ $\forall$has.Distance $\sqcap$ $\exists$has.Group,
Group $\sqsubseteq$ Thing $\sqcap$ $\forall$has.MachineCount $\sqcap$ $\forall$has.Machine,
Machine $\sqsubseteq$ Thing $\sqcap$ $\exists$has.Order $\sqcap$ $\forall$has.ProcessingCapacity $\sqcap$ $\exists$has.WorkingTime $\sqcap$ $\exists$has.FailureTime $\sqcap$ $\exists$has.IdleTime $\sqcap$ $\exists$has.SetupTime $\sqcap$ $\exists$has.RepairTime $\sqcap$ $\forall$has.FailureRate $\sqcap$ $\exists$has.MaintenanceCost $\sqcap$ $\forall$has.Status $\sqcap$ $\forall$has.Location $\sqcap$ $\forall$has.Mobility,
Sink $\sqsubseteq$ Thing $\sqcap$ $\forall$has.SinkCapacity}

Different sets of concepts and properties can be observed by the JSS agent when performing various tasks:

$C_{\tau_{u_0}}$ = {Order}, $F_{\tau_{u_0}}$ = {WaitingTime, DueDate, Priority}

$C_{\tau_{u_1}}$ = {Machine, WorkArea, Group}, $F_{\tau_{u_1}}$ = {WorkingTime, FailureTime, IdleTime, FailureRate, Distance, MachineCount}

$C_{\tau_{u_2}}$ = {Machine, Group}, $F_{\tau_{u_2}}$ = {WorkingTime, FailureTime, IdleTime, FailureRate, MachineCount}

$C_{\tau_{u_3}}$ = {Machine, Order}, $F_{\tau_{u_3}}$ = {WaitingTime, DueDate, Priority, WorkingTime, FailureTime, IdleTime, FailureRate, MachineCount}

$C_{\tau_{u_4}}$ = {Machine, Order}, $F_{\tau_{u_4}}$ = {WaitingTime, DueDate, Priority, WorkingTime, FailureTime, IdleTime, FailureRate}

$C_{\tau_{u_5}}$ = {Sink}, $F_{\tau_{u_5}}$ = {SinkCapacity}

5.4. Reward Machine Modeling

According to Section 3.3, RMs are defined in terms of propositional symbols E that denote features or events of the concrete state of the environment. Intuitively, an RM represents what reward function should currently be used to provide the reward signal, given the sequence of propositional symbols that the agent has seen so far. Ontology can help RL agents define new propositional symbols and additional rewards to guide their learning process beyond those supplied by the underlying MDP. In the following subsections, we describe the details of this process.

5.4.1. New Propositional Symbol Extraction

Based on the concepts and properties of the JSS agent's state, the agent extracts a relevant subsumer, then labels its observation by this subsumer as a new propositional symbol/event $E_{\tau_{u_0}}, \ldots, E_{\tau_{u_5}}$ (see line 4 of Algorithm 1). Finally, the agent creates a new transition to the new state in its RM based on the new propositional symbol.

For example, when the agent is in state $s_{\tau_{u_0}}$, it usually observes subsumer $E_{\tau_{u_0}}$ as follows:

$E_{\tau_{u_0}}$ = {Thing, Time, Number, Belief, (∃has.DueDate, ∀has.PositiveBelief), (∃has.WaitingTime, ∀has.NegativeBelief), (∃has.Priority, ∀has.NegativeBelief)}.

Negative belief associated with the Priority property indicates that a large number of orders with high priority can increase the processing complexity.

However, the agent observes a new subsumer (e.g., $E_{\tau_{u_1}}$) as:

$E_{\tau_{u_1}}$ = {Thing, Time, Number, Belief, (∀has.Distance, ∀has.NegativeBelief), ∃has.Group, (∀has.MachineCount, ∀has.PositiveBelief), ∀has.Machine, (∃has.WorkingTime, ∀has.PositiveBelief), (∃has.IdleTime, ∀has.NegativeBelief), (∃has.FailureTime, ∀has.NegativeBelief), (∀has.FailureRate, ∀has.NegativeBelief)}.

And it recognizes the need to create a new transition to the new state $s_{\tau_{u_1}}$.

In the following, we show subsumers extracted for all states:

$E_{\tau_{u_0}}$ = {Thing, Time, Number, Belief, (∃has.DueDate, ∀has.PositiveBelief), (∃has.WaitingTime, ∀has.NegativeBelief), (∃has.Priority, ∀has.NegativeBelief)}.

$E_{\tau_{u_1}}$ = {Thing, Time, Number, Belief, (∀has.Distance, ∀has.NegativeBelief), ∃has.Group, (∀has.MachineCount, ∀has.PositiveBelief), ∀has.Machine, (∃has.WorkingTime, ∀has.PositiveBelief), (∃has.IdleTime, ∀has.NegativeBelief), (∃has.FailureTime, ∀has.NegativeBelief), (∀has.FailureRate, ∀has.NegativeBelief)}.

$E_{\tau_{u_2}}$ = {Thing, Time, Number, Belief, (∀has.MachineCount, ∀has.PositiveBelief), ∀has.Machine, (∃has.WorkingTime, ∀has.PositiveBelief), (∃has.IdleTime, ∀has.NegativeBelief), (∃has.FailureTime, ∀has.NegativeBelief), (∀has.FailureRate, ∀has.NegativeBelief)}.

$E_{\tau_{u_3}}$ = {Thing, Time, Number, Belief, (∀has.MachineCount, ∀has.PositiveBelief), (∃has.WorkingTime, ∀has.PositiveBelief), (∃has.IdleTime, ∀has.NegativeBelief), (∃has.FailureTime, ∀has.NegativeBelief), (∀has.FailureRate, ∀has.NegativeBelief), ∃has.Order, (∃has.DueDate, ∀has.PositiveBelief), (∃has.WaitingTime, ∀has.NegativeBelief), (∃has.Priority, ∀has.NegativeBelief)}.

$E_{\tau_{u_4}}$ = {Thing, Time, Number, Belief, (∃has.WorkingTime, ∀has.PositiveBelief), (∃has.IdleTime, ∀has.NegativeBelief), (∃has.FailureTime, ∀has.NegativeBelief), (∀has.FailureRate, ∀has.NegativeBelief), ∃has.Order, (∃has.DueDate, ∀has.PositiveBelief), (∃has.WaitingTime, ∀has.NegativeBelief), (∃has.Priority, ∀has.NegativeBelief)}.

$E_{\tau_{u_5}}$ = {Thing, Number, Belief, (∀has.SinkCapacity, ∀has.PositiveBelief)}.

5.4.2. New Reward Function Extraction

A set of reward functions for different states can be deduced using properties with negative or positive beliefs in an ontology-based schema (see line 5 of Algorithm 1). To do so, the agent aims to maximize/minimize the value of properties associated with positive/negative beliefs. For example, for the task τ_{u_0}, one of the reward functions could be defined as maximizing the due date of orders because it is associated with a positive belief in the JSS ontology. So, the agent tries to select orders with a low due date to be processed first (i.e., keep orders with a high due date to be processed later):

$r^1_{\tau_{u_0}}$ = Maximizing the due date of orders (i.e., select an order with a low due date to be processed first).

The following shows extracted reward functions for all states.

$r^1_{\tau_{u_0}}$ = Maximizing the due date of orders (i.e., select an order with a low due date to be processed first), $r^2_{\tau_{u_0}}$ = Minimizing the waiting time of orders, and $r^3_{\tau_{u_0}}$ = Minimizing the priority of orders.

$r^1_{\tau_{u_1}}$ = Minimizing the distance of a work area, $r^2_{\tau_{u_1}}$ = Maximizing the machine count of groups, $r^3_{\tau_{u_1}}$ = Maximizing the working time of machines, $r^4_{\tau_{u_1}}$ = Minimizing the failure rate of machines, $r^5_{\tau_{u_1}}$ = Minimizing the failure time of machines, and $r^6_{\tau_{u_1}}$ = Minimizing the idle time of machines.

$r^1_{\tau_{u_2}}$ = Maximizing the machine count of a group, $r^2_{\tau_{u_2}}$ = Maximizing the working time of machines, $r^3_{\tau_{u_2}}$ = Minimizing the failure rate of machines, $r^4_{\tau_{u_2}}$ = Minimizing the failure time of machines, and $r^5_{\tau_{u_2}}$ = Minimizing the idle time of machines.

$r^1_{\tau_{u_3}}$ = Maximizing machines working time, $r^2_{\tau_{u_3}}$ = Maximizing the due date of orders, $r^3_{\tau_{u_3}}$ = Minimizing the waiting time of orders, $r^4_{\tau_{u_3}}$ = Minimizing the priority of orders, $r^5_{\tau_{u_3}}$ = Minimizing the failure rate of machines, $r^6_{\tau_{u_3}}$ = Minimizing the failure time of machines, $r^7_{\tau_{u_3}}$ = Minimizing the idle time of machines, and $r^8_{\tau_{u_3}}$ = Maximizing the machine count of a group.

$r^1_{\tau_{u_4}}$ = Minimizing the waiting time of orders, $r^2_{\tau_{u_4}}$ = Maximizing the due date of orders, $r^3_{\tau_{u_4}}$ = Minimizing the priority of orders, $r^4_{\tau_{u_4}}$ = Maximizing the working time of machines, $r^5_{\tau_{u_4}}$ = Minimizing the failure rate of machines, $r^6_{\tau_{u_4}}$ = Minimizing the failure time of machines, and $r^7_{\tau_{u_4}}$ = Minimizing the idle time of machines.

$r^1_{\tau_{u_5}}$ = Maximizing the capacity of sinks.

Action–Reward Bindings: The reward function cannot be used for learning if the RL agent's different action selection (task) does not affect its value (i.e., exposing action–reward bindings, see Table 2). As an example, selecting a machine to process an order τ_{u_3} does not impact maximizing the machine count of a group $r^8_{\tau_{u_3}}$, so the group's property (i.e., machine count) cannot be used as a reward function for the task τ_{u_3}.

Table 2. Action–reward bindings: ✓ indicates that the reward function can be used for the task. × indicates that the reward function cannot be used for the task.

Task	Reward Function							
τ_{u_0}	$r^1_{\tau_{u_0}}$	$r^2_{\tau_{u_0}}$	$r^3_{\tau_{u_0}}$	-	-	-	-	-
	✓	✓	✓	-	-	-	-	-
τ_{u_1}	$r^1_{\tau_{u_1}}$	$r^2_{\tau_{u_1}}$	$r^3_{\tau_{u_1}}$	$r^4_{\tau_{u_1}}$	$r^5_{\tau_{u_1}}$	$r^6_{\tau_{u_1}}$	-	-
	✓	✓	✓	✓	✓	✓	-	-
τ_{u_2}	$r^1_{\tau_{u_2}}$	$r^2_{\tau_{u_2}}$	$r^3_{\tau_{u_2}}$	$r^4_{\tau_{u_2}}$	$r^5_{\tau_{u_2}}$	-	-	-
	✓	✓	✓	✓	✓	-	-	-
τ_{u_3}	$r^1_{\tau_{u_3}}$	$r^2_{\tau_{u_3}}$	$r^3_{\tau_{u_3}}$	$r^4_{\tau_{u_3}}$	$r^5_{\tau_{u_3}}$	$r^6_{\tau_{u_3}}$	$r^7_{\tau_{u_3}}$	$r^8_{\tau_{u_3}}$
	✓	✓	✓	✓	✓	✓	✓	×
τ_{u_4}	$r^1_{\tau_{u_4}}$	$r^2_{\tau_{u_4}}$	$r^3_{\tau_{u_4}}$	$r^4_{\tau_{u_4}}$	$r^5_{\tau_{u_4}}$	$r^6_{\tau_{u_4}}$	$r^7_{\tau_{u_4}}$	-
	✓	✓	✓	×	×	×	×	-
τ_{u_5}	$r^1_{\tau_{u_5}}$	-	-	-	-	-	-	-
	✓	-	-	-	-	-	-	-

5.5. Learning

We assume we have a separate learner in each of the states of RM; for example, for task τ_{u_0}, the learner will be formulated by the following MDP:

$\mathcal{M}_{\tau_{u_0}} = (s_{\tau_{u_0}}, \mathcal{A}_{\tau_{u_0}}, r_{\tau_{u_0}}, p_{\tau_{u_0}}, \gamma_{\tau_{u_0}})$

The agent selects actions using the policy of the current learner. Also, we can use a multi-advisor RL [63] in which we have multi-learners solving a task with various reward functions. Their advice, taking the form of action values, is then communicated to an aggregator responsible for merging the advisors' recommendations into a global policy by voting (i.e., selecting the action that receives the most individual recommendations from the advisors).

6. Evaluation

We evaluate ONTOADAPT-REWARD in a JSS environment. Our simulated JSS setup includes three sources, denoted as I_1, I_2, and I_3, which generate orders. Sixteen machines, labeled $\{m_0, m_2, \ldots, m_{15}\}$, process these orders according to specified operation sequences. Each machine operates with a single processing capacity, allowing only one order to be processed at a time. The capacity of the sources (i.e., the number of orders generated at each time step) is determined by predefined scenarios. Machines are categorized into three groups, n_1, n_2, and n_3, located across three work areas, z_1, z_2, and z_3. The processed orders are consumed by three sinks, x_1, x_2, and x_3. We define 30%, 50%, and 20% of orders with low, medium, and high priority, respectively. For 50% of machines, the failure rate is low, medium for 30%, and high for 20%, corresponding to a Mean Time Between Failures (MTBF) of 2000, 1000, and 500, respectively.

Baseline Method: The baseline method is based on the framework for job shop manufacturing systems proposed by [60,64]. Heuristics algorithms (e.g., FIFO and LIFO) and TRPO as the RL algorithm are used to optimize the scheduling decisions in different states according to Table 3. TRPO is chosen due to its ability to ensure stable and reliable policy updates, which is critical in complex environments like JSS.

Table 3. The baseline method and the ONTOADAPT-REWARD-based proposed method: settings.

			ONTOADAPT-REWARD-Based Proposed Method	
		Baseline Method	**Without Multi-Advisor**	**With Multi-Advisor**
State	$s_{\tau_{u_0}}$	LIFO	TRPO ($r^1_{\tau_{u_0}}$)	TRPO ($r^1_{\tau_{u_0}}, r^2_{\tau_{u_0}}, r^3_{\tau_{u_0}}$)
	$s_{\tau_{u_1}}$	Random	Random	Random
	$s_{\tau_{u_2}}$	Random	Random	Random
	$s_{\tau_{u_3}}$	TRPO	TRPO ($r^1_{\tau_{u_3}}$)	TRPO ($r^1_{\tau_{u_3}}$)
	$s_{\tau_{u_4}}$	FIFO	TRPO ($r^1_{\tau_{u_4}}$) FIFO	TRPO ($r^1_{\tau_{u_4}}$) FIFO
	$s_{\tau_{u_5}}$	Random	Random	Random

6.1. ONTOADAPT-REWARD-Based JSS

The learning method defines separate TRPO learners for different tasks and states, each with distinct ontology-based reward functions. These reward functions are designed to guide the learning process toward desirable scheduling outcomes, such as minimizing waiting time and maximizing machine utilization. Based on the ONTOADAPT-REWARD model, we propose the following method (see Figure 6):

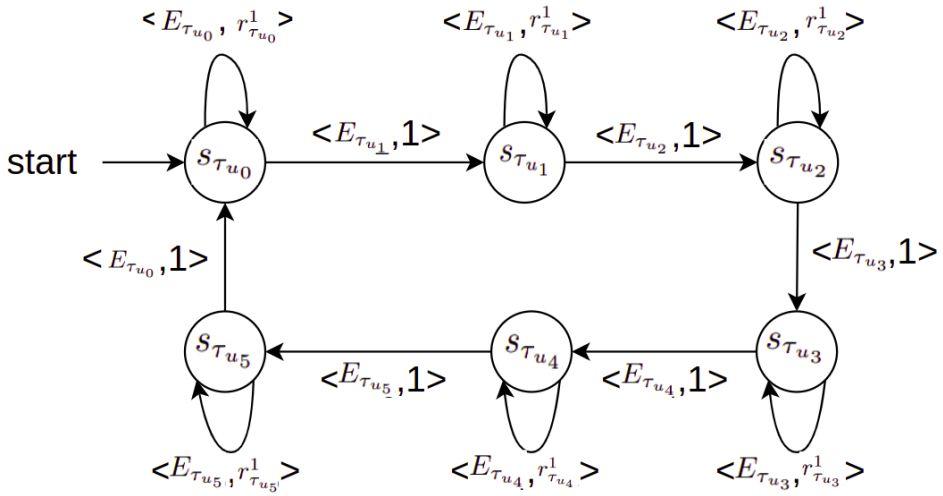


Figure 6. A reward machine for the job shop scheduling environment. $s_{\tau_{u_0}}, \ldots, s_{\tau_{u_5}}$ indicates the agent's different states, $E_{\tau_{u_0}}, \ldots, E_{\tau_{u_5}}$ are subsumers extracted from the agent's states as new propositional symbols/events, and $r^1_{\tau_{u_0}}, \ldots, r^1_{\tau_{u_5}}$ are the ontology-based reward functions. We assume that the agent receives a fixed reward value equal to 1 to transition from one state to the next.

In state u_0, the agent defines three separate TRPO learners $\mathcal{M}^1_{\tau_{u_0}}, \mathcal{M}^2_{\tau_{u_0}}$, and $\mathcal{M}^3_{\tau_{u_0}}$ for task τ_{u_0} with different reward functions $r^1_{\tau_{u_0}}, r^2_{\tau_{u_0}}, r^3_{\tau_{u_0}}$ respectively. For the sake of simplicity, for tasks τ_{u_1} and τ_{u_2} the agent uses a random strategy same as the baseline method; however, there is the possibility that the agent defines separate learners for task τ_{u_1} using ontology-based reward functions $\{r^1_{\tau_{u_1}}, \ldots, r^6_{\tau_{u_1}}\}$ and for task τ_{u_2} with reward functions $\{r^1_{\tau_{u_2}}, \ldots, r^5_{\tau_{u_2}}\}$. The agent uses one learner $\mathcal{M}^1_{\tau_{u_3}}$ for task τ_{u_3} with reward function $r^1_{\tau_{u_3}}$ (i.e., other learners can be defined using reward functions $\{r^2_{\tau_{u_3}}, \ldots, r^7_{\tau_{u_3}}\}$). In state u_4, in 4 out of 16 machines, the agent uses one learner $\mathcal{M}^1_{\tau_{u_4}}$ for task τ_{u_4} with reward function $r^1_{\tau_{u_4}}$ (it is possible to launch two more learners for this task with reward functions $r^2_{\tau_{u_4}}$ and $r^3_{\tau_{u_4}}$). For the 12 machines left, the agent uses the baseline FIFO strategy. For task τ_{u_5}, the agent uses the random strategy same as the baseline method; however, the agent may define a separate learner for this task using the ontology-based reward function $r^1_{\tau_{u_5}}$ (see Table 3).

6.2. Model Evaluation

Ensuring the reliability of the RL-based model requires thorough evaluation of the training results. To evaluate the model, we implemented the following steps:

Performance Metrics Evaluation: To evaluate the proposed method's performance, we use the following metrics:

1. Average utilization rate of machines $\overline{E}$ (see Equation (6)).
2. Average waiting time $\overline{w}$: The average waiting time of orders is computed as follows:

$$\overline{w} = \frac{1}{|D|} \sum_{i=0}^{|D|-1} w_{d_i}^t \tag{7}$$

3. Total failed orders $|D_f|$: The number of failed orders due to delay (i.e., due date requirement).
4. Total processed orders $|D_p|$: The count of successfully processed orders.

The performance of the baseline method is compared to that of our proposed method.

Scenario-based Validation: The model's performance was validated across multiple scenarios with varying order loads and due date requirements (refer to Table 4). We conducted 10 runs for each scenario to ensure consistency and reliability of the results.

Table 4. Job shop scheduling environment scenarios.

	Description	
Order load	Light	Generating three orders at each time step.
	Heavy	Generating six orders at each time step.
Due date level	High	5% of orders have a low due date, 80% of them have a medium due date, and 15% have a high due date.
	Low	25% of orders have a low due date, 70% of them have a medium due date, and 5% have a high due date.

6.3. Results and Discussion

Figure 7 presents the average utilization rate, waiting time, total processed orders, and failed orders across 10 runs in each scenario for both the baseline method and the proposed ONTOADAPT-REWARD-based method. The results show that the proposed method increases the average utilization rate and the total processed orders and decreases the average waiting time and the total failed orders compared to the baseline method. We observe an increase in the total number of processed orders by 6%, 14%, 14%, and 15%, and the average utilization rate is increased by 5%, 11%, 12%, and 16% in Light-High, Light-Low, Heavy-High, and Heavy-Low scenarios, respectively (see Table 5). So, the percentage change is more significant when the number of orders increases. Also, the total failed orders are decreased by 68%, 73%, 41%, and 32%, and the average waiting time is decreased by 13%, 8%, 9%, and 6% in Light-High, Light-Low, Heavy-High, and Heavy-Low scenarios, respectively. The improvement in the Low scenarios is lower than in the High scenarios. This is because the number of low due date orders increases, and our proposed method shows less improvement due to the limited number of resources.

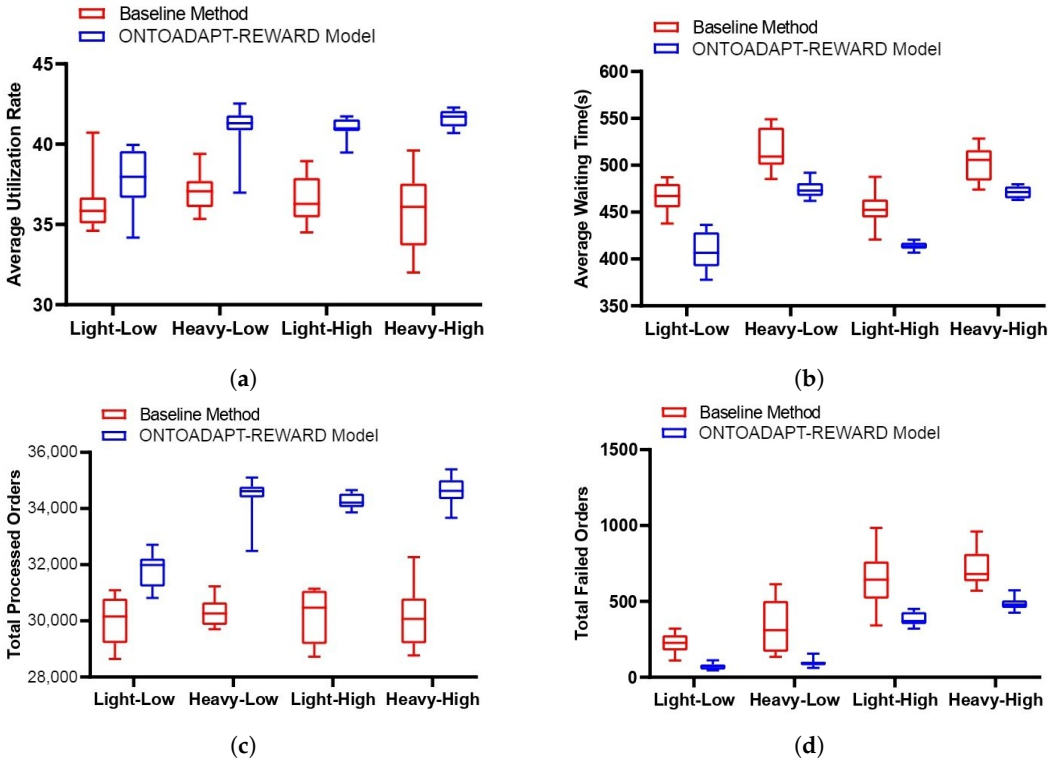


Figure 7. The baseline method and the ONTOADAPT-REWARD-based proposed method (without multi-advisor RL). (**a**) Average utilization rate. (**b**) Average waiting time. (**c**) Total processed orders. (**d**) Total failed orders.

Table 5. The percentage change in performance metrics: the baseline method and the ONTOADAPT-REWARD-based proposed method (without multi-advisor RL).

Scenario	Performance Metrics				
	Utilization Rate (Increase)	Waiting Time (Decrease)	Processed Orders (Increase)	Failed Orders (Decrease)	AVG
Light-High	5%	13%	6%	68%	23%
Light-Low	11%	8%	14%	73%	27%
Heavy-High	12%	9%	14%	41%	19%
Heavy-Low	16%	6%	15%	32%	17%
AVG	11%	9%	12%	54%	-

Similarly, compared to the baseline method, the ONTOADAPT-REWARD-based proposed method (with multi-advisor RL) increases the average utilization rate and the total number of processed orders while decreasing the average waiting time and the total number of failed orders (see Figure 8). We observe that the total processed orders is increased by 16%, 16%, 16%, and 18%, and the average utilization rate is increased by 15%, 14%, 16%, and 19% in Light-High, Light-Low, Heavy-High, and Heavy-Low scenarios, respectively (see Table 6). So, the percentage change is more significant with a high number of orders. Also, the total failed orders are decreased by 51%, 53%, 18%, and 4%, and the average waiting time is decreased by 3%, 0%, 1%, and −1% in Light-High, Light-Low, Heavy-High, and Heavy-Low scenarios, respectively. In Low scenarios, limited resources result in less improvement due to the increased number of low due date orders. When we compare the results of the proposed method with and without multi-advisor RL, we observe that the improvement without multi-advisor RL is more significant than the other one. The reason can be the multiple conflicting reward functions (objectives), which must be balanced based on their relative importance, so a simple voting algorithm does not work well, and a more comprehensive reward evaluation is required.

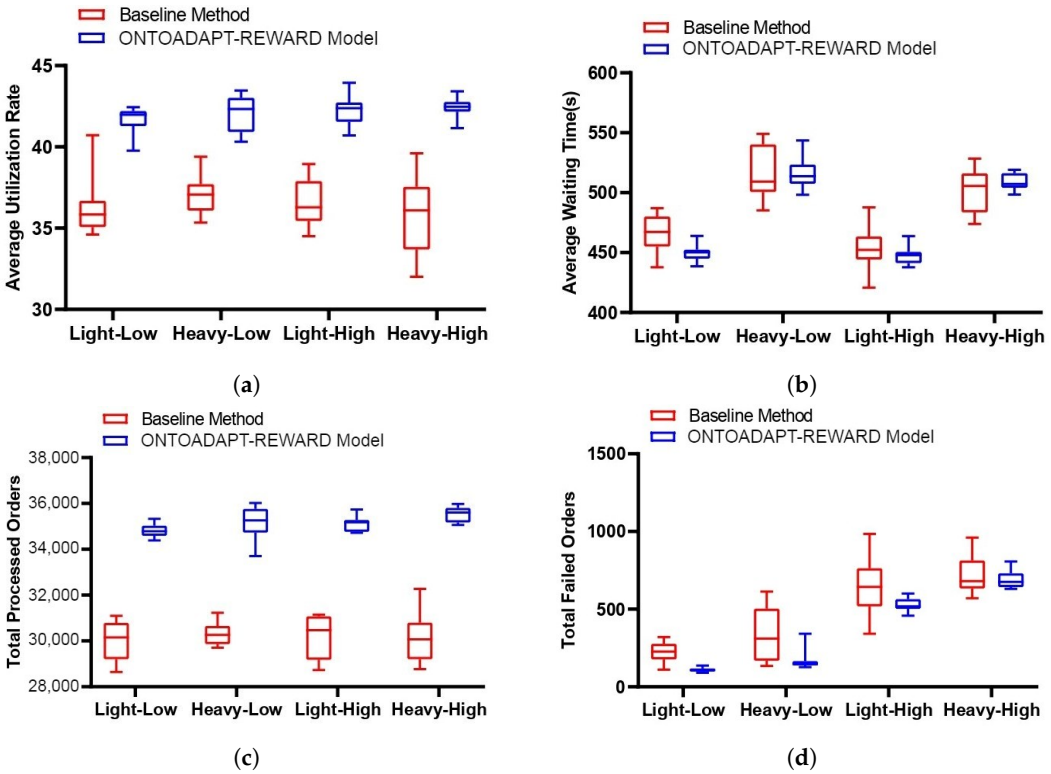


Figure 8. The baseline method and the ONTOADAPT-REWARD-based proposed method (with multi-advisor RL). (**a**) Average utilization rate. (**b**) Average waiting time. (**c**) Total processed orders. (**d**) Total failed orders.

Table 6. The percentage change in performance metrics: the baseline method and the ONTOADAPT-REWARD-based proposed method (with multi-advisor RL).

Scenario	Performance Metrics				
	Utilization Rate (Increase)	Waiting Time (Decrease)	Processed Orders (Increase)	Failed Orders (Decrease)	AVG
Light-High	15%	3%	16%	51%	21%
Light-Low	14%	0%	16%	53%	21%
Heavy-High	16%	1%	16%	18%	13%
Heavy-Low	19%	−1%	18%	4%	10%
AVG	16%	1%	17%	32%	-

7. Conclusions and Future Work

Job Shop Scheduling (JSS) often involves conflicting objectives, dynamic factors, and intricate trade-offs between goals like minimizing production time and cost and maximizing resource utilization. Adaptive reward functions allow the Reinforcement Learning (RL) agent to dynamically prioritize these objectives based on evolving conditions, enabling more effective decision-making and resulting in more sustainable manufacturing by optimizing resource usage, machine usage, and reducing idle times.

The Reward Machine (RM) provides a structured, automata-based representation of a reward function for RL agents to decompose problems into subproblems that can be learned efficiently. RL agents can focus on learning and optimizing actions within these subproblems, leading to more efficient decision-making and improved scheduling outcomes. Additionally, RMs facilitate better generalization across different scheduling scenarios, enhancing the adaptability of RL-based job shop schedulers. As problems become more complex, the RM design can become cumbersome. In addition, RMs need to be adaptive when dealing with unexpected events in dynamic and partially observable environments.

To create/modify RMs dynamically/adaptively through ontological knowledge, we propose an Ontology-based Adaptive Reward Machine (ONTOADAPT-REWARD) model.

In this paper, we have tested the proposed model in JSS scenarios, specifically applying it to a complex manufacturing environment involving various production stages. This research opens up multiple avenues for further exploration. In our future work, we will test ONTOADAPT-REWARD in additional scenarios. For instance, we could explore how the proposed solution performs when considering multiple production lines or non-deterministic production environments. While our current validation has been conducted in a simulated environment, we have a plan to test and refine our approach in real-world settings, considering more boundary conditions, starting from lead time to organizational and availability issues, tool wear, tool change, and maintenance. Future work could incorporate mechanisms to manage machine breakdowns by reallocating orders or rescheduling tasks to minimize downtime. Integrating predictive maintenance and machine health monitoring could help anticipate breakdowns and proactively adjust schedules. Agent-based modeling [65] can simulate the behavior of machines, maintenance processes, and scheduling decisions dynamically in response to changing conditions. For a comprehensive assessment of the energy-saving potential of the ONTOADAPT-REWARD model, future work could integrate energy-specific metrics into the evaluation framework. Moreover, our ONTOADAPT-REWARD model does not specifically address the material handling system (i.e., the movement, protection, storage, and control of materials) within the factory layout. Instead, it focuses on optimizing the scheduling and processing of orders through various machines and work areas. While the handling system undoubtedly impacts overall efficiency, our model assumes ideal conditions for material movement without explicitly modeling these dynamics. Future work could incorporate a more detailed handling system to explore its effects on scheduling decisions and overall manufacturing efficiency. Future research should also investigate the application of the ONTOADAPT-REWARD model to different production types, such as batch production, to generalize its applicability across various manufacturing processes and systems. This could involve adapting the model to handle the unique characteristics and constraints of batch production, including varying batch sizes, inter-batch setup times, and production sequencing.

The ontology used in the ONTOADAPT-REWARD model must be accurate and complete for the model to perform well. Dynamic environments require ontologies to evolve and update frequently, adapting to changing requirements. So, ontology evolution techniques [66] becomes crucial, for example, when production constraints may change over time. Comprehending, exploring, and exploiting an ontology-based schema can be complex, and summarization techniques [67,68] can be employed to provide a concise overview of critical concepts for better understanding and decision-making. In addition, concepts and properties within the ontology can differ depending on the specific context, such as during high-demand production periods, resource shortages, or power outages, leading to different ontologies for the RL agent to adapt accordingly. The impact of an incorrect belief about a property must be assessed to prevent scheduling errors. Additionally, a mechanism should be defined for dynamic weighting and objective priorities where there are multiple objectives in a manufacturing facility, including minimizing production costs, maximizing throughput, and minimizing job completion times simultaneously. Furthermore, actions with the same reward function should use a common learner to leverage the agent's experiences efficiently and improve decision-making across various production stages. By using the relations defined in an ontology-based schema, it is possible to extract concepts related to each property, thereby reducing the agent's state dimensionality for each learner and enhancing computational efficiency. It is especially important in a factory with hundreds of machines and jobs. Therefore, ONTOADAPT-REWARD is suitable for both small and large corporations due to its scalability. The model adapts to varying levels of complexity and size by concentrating on relevant concepts such as machine availability and order priorities, making it feasible to find optimal schedules efficiently regardless of the number of machines and jobs involved. Also, we can leverage the scalability of edge computing [69]

to handle increasing data volumes and computational requirements as the manufacturing environment scales.

Author Contributions: Conceptualization, S.G.; methodology, S.G. and F.G.; validation, A.Z. and S.G.; writing—original draft preparation, S.G.; writing—review and editing, S.G., F.G. and A.Z.; visualization, A.Z. and S.G.; supervision, F.G.; F.G. and S.G. contributed equally to this paper. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by Science Foundation Ireland grant number 21/RC/10295_P2.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Briefing, U.S. International energy outlook 2013. *US Energy Inf. Adm.* **2013**, *506*, 507.
2. International Energy Agency. *Global Energy Review 2022*; International Energy Agency: Paris, France, 2022.
3. Yin, L.; Li, X.; Gao, L.; Lu, C.; Zhang, Z. A novel mathematical model and multi-objective method for the low-carbon flexible job shop scheduling problem. *Sustain. Comput. Inform. Syst.* **2017**, *13*, 15–30. [CrossRef]
4. Eslami, Y.; Lezoche, M.; Panetto, H.; Dassisti, M. On analysing sustainability assessment in manufacturing organisations: A survey. *Int. J. Prod. Res.* **2021**, *59*, 4108–4139. [CrossRef]
5. Popper, J.; Motsch, W.; David, A.; Petzsche, T.; Ruskowski, M. Utilizing multi-agent deep reinforcement learning for flexible job shop scheduling under sustainable viewpoints. In Proceedings of the International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME), Mauritius, 7–8 October 2021; IEEE: Piscataway, NJ, USA, 2021; pp. 1–6.
6. Goutham, P.M.; Rohit, Y.; Nanjundeswaraswamy, T. A review on smart manufacturing, technologies and challenges. *Int. Res. J. Eng. Technol. IRJET* **2022**, *9*, 663–678.
7. Yang, L.; Li, J.; Chao, F.; Hackney, P.; Flanagan, M. Job shop planning and scheduling for manufacturers with manual operations. *Expert Syst.* **2021**, *38*. [CrossRef]
8. Pach, C.; Berger, T.; Sallez, Y.; Bonte, T.; Adam, E.; Trentesaux, D. Reactive and energy-aware scheduling of flexible manufacturing systems using potential fields. *Comput. Ind.* **2014**, *65*, 434–448. [CrossRef]
9. Tang, D.; Dai, M. Energy-efficient approach to minimizing the energy consumption in an extended job-shop scheduling problem. *Chin. J. Mech. Eng.* **2015**, *28*, 1048–1055. [CrossRef]
10. Yang, S.; Wang, D.; Chai, T.; Kendall, G. An improved constraint satisfaction adaptive neural network for job-shop scheduling. *J. Sched.* **2010**, *13*, 17–38. [CrossRef]
11. Zhou, T.; Zhu, H.; Tang, D.; Liu, C.; Cai, Q.; Shi, W.; Gui, Y. Reinforcement learning for online optimization of job-shop scheduling in a smart manufacturing factory. *Adv. Mech. Eng.* **2022**, *14*, 1–19.
12. Zeng, Y.; Liao, Z.; Dai, Y.; Wang, R.; Li, X.; Yuan, B. Hybrid intelligence for dynamic job-shop scheduling with deep reinforcement learning and attention mechanism. *arXiv* **2022**, arXiv:2201.00548.
13. Ghanadbashi, S.; Zarchini, A.; Golpayegani, F. An ontology-based augmented observation for decision-making in partially observable environments. In Proceedings of the International Conference on Agents and Artificial Intelligence (ICAART), Lisbon, Portugal, 22–24 February 2023; SCITEPRESS: Setubal, Portugal, 2023; pp. 343–354.
14. Ghanadbashi, S. Ontology-enhanced decision-making for autonomous agents in dynamic and partially observable environments. *arXiv* **2024**, arXiv:2405.17691.
15. Wang, C.; Zeng, L. Optimization of multi-objective job-shop scheduling under uncertain environment. *J. Eur. Syst. Autom.* **2019**, *52*, 179–183. [CrossRef]
16. Zhang, H.; Buchmeister, B.; Li, X.; Ojstersek, R. Advanced metaheuristic method for decision-making in a dynamic job shop scheduling environment. *Mathematics* **2021**, *9*, 909. [CrossRef]
17. Cunha, B.; Madureira, A.M.; Fonseca, B.; Coelho, D. Deep reinforcement learning as a job shop scheduling solver: A literature review. In Proceedings of the Hybrid Intelligent Systems: 18th International Conference on Hybrid Intelligent Systems (HIS 2018), Porto, Portugal, 13–15 December 2018; Springer: Cham, Switzerland, 2020; pp. 350–359.
18. Palacio, J.C.; Jiménez, Y.M.; Schietgat, L.; Van Doninck, B.; Nowé, A. A Q-Learning algorithm for flexible job shop scheduling in a real-world manufacturing scenario. *Procedia CIRP* **2022**, *106*, 227–232. [CrossRef]
19. Cao, Z.; Zhou, L.; Hu, B.; Lin, C. An adaptive scheduling algorithm for dynamic jobs for dealing with the flexible job shop scheduling problem. *Bus. Inf. Syst. Eng.* **2019**, *61*, 299–309. [CrossRef]
20. Bülbül, K.; Kaminsky, P. A linear programming-based method for job shop scheduling. *J. Sched.* **2013**, *16*, 161–183. [CrossRef]

21. Yu, Y.; Ying, Y. The dynamic job shop scheduling approach based on data-driven genetic algorithm. *Open Electr. Electron. Eng. J.* **2014**, *8*, 41–45. [CrossRef]
22. Bürgy, R. A neighborhood for complex job shop scheduling problems with regular objectives. *J. Sched.* **2017**, *20*, 391–422. [CrossRef]
23. Nguyen, S.; Zhang, M.; Johnston, M.; Tan, K.C. Genetic programming for job shop scheduling. In *Evolutionary and Swarm Intelligence Algorithms*; Springer: Cham, Switzerland, 2019; pp. 143–167.
24. Abdullah, S.; Abdolrazzagh-Nezhad, M. Fuzzy job-shop scheduling problems: A review. *Inf. Sci.* **2014**, *278*, 380–407. [CrossRef]
25. Li, R.; Gong, W.; Lu, C. Self-adaptive multi-objective evolutionary algorithm for flexible job shop scheduling with fuzzy processing time. *Comput. Ind. Eng.* **2022**, *168*, 108099. [CrossRef]
26. Pathak, D.; Agrawal, P.; Efros, A.A.; Darrell, T. Curiosity-driven exploration by self-supervised prediction. In Proceedings of the International Conference on Machine Learning (ICML), Sydney, NSW, Australia, 6–11 August 2017; PMLR: Breckenridge, CO, USA, 2017; pp. 2778–2787.
27. Yang, D.; Tang, Y. Adaptive inner-reward shaping in sparse reward games. In Proceedings of the International Joint Conference on Neural Networks (IJCNN), Glasgow, UK, 19–24 July 2020; IEEE: New York, NY, USA, 2020; pp. 1–8.
28. Hu, Y.; Wang, W.; Jia, H.; Wang, Y.; Chen, Y.; Hao, J.; Wu, F.; Fan, C. Learning to utilize shaping rewards: A new approach of reward shaping. *Adv. Neural Inf. Process. Syst.* **2020**, *33*, 15931–15941.
29. De Hauwere, Y.M.; Devlin, S.; Kudenko, D.; Nowé, A. Context-sensitive reward shaping for sparse interaction multi-agent systems. *Knowl. Eng. Rev.* **2016**, *31*, 59–76. [CrossRef]
30. Tenorio-Gonzalez, A.C.; Morales, E.F.; Villasenor-Pineda, L. Dynamic reward shaping: Training a robot by voice. In Proceedings of the Ibero-American Conference on Artificial Intelligence (IBERAMIA), Bahia Blanca, Argentina, 1–5 November 2010; Springer: Berlin/Heidelberg, Germany, 2010; pp. 483–492.
31. Michini, B.; Cutler, M.; How, J.P. Scalable reward learning from demonstration. In Proceedings of the International Conference on Robotics and Automation (ICRA), Karlsruhe, Germany, 6–10 May 2013; IEEE: New York, NY, USA, 2013; pp. 303–308.
32. Hadfield-Menell, D.; Milli, S.; Abbeel, P.; Russell, S.J.; Dragan, A. Inverse reward design. *Adv. Neural Inf. Process. Syst.* **2017**, *30* , 6765–6774.
33. Baier, C.; Katoen, J.P. *Principles of Model Checking*; MIT Press: Cambridge, MA, USA, 2008.
34. Wainwright, M.J.; Jordan, M.I. Graphical models, exponential families, and variational inference. *Found. Trends Mach. Learn.* **2008**, *1*, 1–305. [CrossRef]
35. Toro Icarte, R.; Waldie, E.; Klassen, T.; Valenzano, R.; Castro, M.; McIlraith, S. Learning reward machines for partially observable reinforcement learning. *Adv. Neural Inf. Process. Syst.* **2019**, *32* , 15497–15508. [CrossRef]
36. Furelos-Blanco, D.; Law, M.; Jonsson, A.; Broda, K.; Russo, A. Hierarchies of reward machines. *arXiv* **2022**, arXiv:2205.15752.
37. Zhou, W.; Li, W. A hierarchical bayesian approach to inverse reinforcement learning with symbolic reward machines. *arXiv* **2022**, arXiv:2204.09772.
38. Brewster, C.; O'Hara, K. Knowledge representation with ontologies: The present and future. *IEEE Intell. Syst.* **2004**, *19*, 72–81. [CrossRef]
39. Ghanadbashi, S.; Zarchini, A.; Golpayegani, F. Ontology-based adaptive reward functions. In Proceedings of the Modelling and Representing Context (MRC) at 26th European Conference on Artificial Intelligence (ECAI), Krakow, Poland, 30 September–4 October; Springer: Berlin/Heidelberg, Germany, 2023; pp. 1–7.
40. Chen, J.; Frank Chen, F. Adaptive scheduling and tool flow control in flexible job shops. *Int. J. Prod. Res. IJPR* **2008**, *46*, 4035–4059. [CrossRef]
41. Zhang, B.X.; Yi, L.X.; Xiao, S. Study of stochastic job shop dynamic scheduling. In Proceedings of the International Conference on Machine Learning and Cybernetics, Guangzhou, China, 18–21 August 2005; IEEE: New York, NY, USA, 2005; Volume 2, pp. 911–916.
42. Dominic, P.D.; Kaliyamoorthy, S.; Kumar, M.S. Efficient dispatching rules for dynamic job shop scheduling. *Int. J. Adv. Manuf. Technol.* **2004**, *24*, 70–75. [CrossRef]
43. Chen, S.; Huang, Z.; Guo, H. An end-to-end deep learning method for dynamic job shop scheduling problem. *Machines* **2022**, *10*, 573. [CrossRef]
44. Toro Icarte, R.; Klassen, T.; Valenzano, R.; McIlraith, S. Using reward machines for high-level task specification and decomposition in reinforcement learning. In Proceedings of the International Conference on Machine Learning (ICML), Stockholm, Sweden, 10–15 July 2018; PMLR: Breckenridge, CO, USA, 2018; pp. 2107–2116.
45. Meuleau, N.; Peshkin, L.; Kim, K.E.; Kaelbling, L.P. Learning finite-state controllers for partially observable environments. *arXiv* **2013**, arXiv:1301.6721.
46. Mahmud, M.M.H. Constructing states for reinforcement learning. In Proceedings of the International Conference on Machine Learning (ICML), Haifa, Israel, 21–24 June 2010; OmniPress: West Norriton, PA, USA, 2010; pp. 727–734.
47. Ghanadbashi, S.; Golpayegani, F. Using ontology to guide reinforcement learning agents in unseen situations. *Appl. Intell. APIN* **2022**, *52*, 1808–1824. [CrossRef]
48. Ng, A.Y.; Harada, D.; Russell, S. Policy invariance under reward transformations: Theory and application to reward shaping. In Proceedings of the International Conference on Machine Learning (ICML), Bled, Slovenia, 27–30 June 1999; Volume 99, pp. 278–287.

49. Mannion, P.; Devlin, S.; Duggan, J.; Howley, E. Reward shaping for knowledge-based multi-objective multi-agent reinforcement learning. *Knowl. Eng. Rev.* **2018**, *33*, e23. [CrossRef]
50. Laud, A.D. Theory and Application of Reward Shaping in Reinforcement Learning. Ph.D. Thesis, University of Illinois at Urbana-Champaign, Urbana, IL, USA, 2004.
51. Zou, H.; Ren, T.; Yan, D.; Su, H.; Zhu, J. Reward shaping via meta-learning. *arXiv* **2019**, arXiv:1901.09330.
52. Niekum, S.; Spector, L.; Barto, A. Evolution of reward functions for reinforcement learning. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), Dublin, Ireland, 12–16 July 2011; pp. 177–178.
53. Hussein, A.; Elyan, E.; Gaber, M.M.; Jayne, C. Deep reward shaping from demonstrations. In Proceedings of the International Joint Conference on Neural Networks (IJCNN), Anchorage, AK, USA, 14–19 May 2017; IEEE: New Yoek, NY, USA, 2017; pp. 510–517.
54. Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; Moritz, P. Trust region policy optimization. In Proceedings of the International Conference on Machine Learning (ICML), Lille, France, 7–9 July 2015; PMLR: Breckenridge, CO, USA, 2015; pp. 1889–1897.
55. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; MIT Press: Cambridge, MA, USA, 2018.
56. Zouaq, A.; Nkambou, R. A survey of domain ontology engineering: Methods and tools. In *Advances in Intelligent Tutoring Systems*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 103–119.
57. Alsubait, T.; Parsia, B.; Sattler, U. Measuring similarity in ontologies: A new family of measures. In Proceedings of the International Conference on Knowledge Engineering and Knowledge Management (EKAW), Linkoping, Sweden, 24–28 November 2014; Springer: Berlin/Heidelberg, Germany, 2014; pp. 13–25.
58. Pfitzer, F.; Provost, J.; Mieth, C.; Liertz, W. Event-driven production rescheduling in job shop environments. In Proceedings of the International Conference on Automation Science and Engineering (CASE), Munich, Germany, 20–24 August 2018; IEEE: New Yoek, NY, USA, 2018; pp. 939–944.
59. Buchmeister, B.; Ojstersek, R.; Palcic, I. Advanced methods for job shop scheduling. *Adv. Prod. Ind. Eng. APEM* **2017**, 31. [CrossRef]
60. Kuhnle, A. Simulation and Reinforcement Learning Framework for Production Planning and Control of Complex Job Shop Manufacturing Systems. 2020. Available online: https://github.com/AndreasKuhnle/SimRLFab (accessed on 1 June 2022).
61. Haller, A.; Janowicz, K.; Cox, S.J.; Lefrançois, M.; Taylor, K.; Le Phuoc, D.; Lieberman, J.; García-Castro, R.; Atkinson, R.; Stadler, C. The modular SSN ontology: A joint W3C and OGC standard specifying the semantics of sensors, observations, sampling, and actuation. *Semant. Web* **2019**, *10*, 9–32. [CrossRef]
62. Duy, T.K.; Quirchmayr, G.; Tjoa, A.; Hanh, H.H. A semantic data model for the interpretion of environmental streaming data. In Proceedings of the International Conference on Information Science and Technology (ICIST), Nis, Serbia, 28–30 June 2017; IEEE: New York, NY, USA, 2017; pp. 376–380.
63. Laroche, R.; Fatemi, M.; Romoff, J.; van Seijen, H. Multi-advisor reinforcement learning. *arXiv* **2017**, arXiv:1704.00756.
64. Kuhnle, A.; Röhrig, N.; Lanza, G. Autonomous order dispatching in the semiconductor industry using reinforcement learning. *Procedia CIRP* **2019**, *79*, 391–396. [CrossRef]
65. Golpayegani, F.; Clarke, S. Co-Ride: Collaborative preference-based taxi-sharing and taxi-dispatch. In Proceedings of the IEEE 30th International Conference on Tools with Artificial Intelligence, ICTAI 2018, Volos, Greece, 5–7 November 2018; Tsoukalas, L.H., Grégoire, É., Alamaniotis, M., Eds.; IEEE: New York, NY, USA, 2018; pp. 864–871. [CrossRef]
66. Zablith, F.; Antoniou, G.; d'Aquin, M.; Flouris, G.; Kondylakis, H.; Motta, E.; Plexousakis, D.; Sabou, M. Ontology evolution: A process-centric survey. *Knowl. Eng. Rev. KER* **2015**, *30*, 45–75. [CrossRef]
67. Pires, C.E.; Sousa, P.; Kedad, Z.; Salgado, A.C. Summarizing ontology-based schemas in PDMS. In Proceedings of the International Conference on Data Engineering Workshops (ICDEW), Long Beach, CA, USA, 1–6 March 2010; IEEE: New York, NY, USA, 2010; pp. 239–244.
68. Pouriyeh, S.; Allahyari, M.; Kochut, K.; Arabnia, H.R. A comprehensive survey of ontology summarization: Measures and methods. *arXiv* **2018**, arXiv:1801.01937.
69. Golpayegani, F.; Chen, N.; Afraz, N.; Gyamfi, E.; Malekjafarian, A.; Schäfer, D.; Krupitzer, C. *Adaptation in Edge Computing: A Review on Design Principles and Research Challenges*; ACM Transactions on Autonomous and Adaptive Systems: New York, NY, USA, 2024.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.