




An Ontology-Based Augmented Observation for Decision-Making in Partially Observable Environments

Saeedeh Ghanadbashi¹^a, Akram Zarchini²^b and Fatemeh Golpayegani¹^c

¹*School of Computer Science, University College Dublin, Ireland*

²*Department of Computer Engineering, Sharif University of Technology, Islamic Republic of Iran*

Keywords: Multi-Agent Systems (MAS), Partially Observable Environment, Autonomy, Reinforcement Learning (RL), Ontology, Job Shop Scheduling Environment.


Abstract: Decision-making is challenging for agents operating in partially observable environments. In such environments, agents' observation is often based on incomplete, ambiguous, and noisy sensed data, which may lead to perceptual aliasing. This means there might be distinctive states of the environment that appear the same to the agents, and agents fail to take suitable actions. Currently, machine learning, collaboration, and practical reasoning techniques are used to improve agents' observation and their performance in such environments. However, their long exploration and negotiation periods make them incapable of reacting in real time and making decisions on the fly. The Ontology-based Observation Augmentation Method (OOAM) proposed here, improves agents' action selection in partially observable environments using domain ontology. OOAM generates an ontology-based schema (i.e., mapping low-level sensor data to high-level concepts), and infers implicit observation data from explicit ones. OOAM is evaluated in a job shop scheduling environment, where the required sensed data to process the orders can be delayed or corrupted. The results show that the average utilization rate and the total processed orders have increased by 17% and 25% respectively compared to Trust Region Policy Optimization (TRPO) as a state-of-the-art method.


1 INTRODUCTION


Multi-agent Systems (MAS) are comprised of agents interacting in an environment, coordinating their behavior, and making decisions autonomously to solve complex problems (Stankovic et al., 2009). The complexity of an agent's decision-making process is affected by the properties of its environment (Wooldridge, 2009). In many real-world problems, noisy and inaccurate sensed data or missing data cause partial observability leading to incomplete and noisy state observation. In such environments, an agent's observed state differs from the environment's state, and the agent must construct its state representation. Reinforcement Learning (RL) is a common technique used in such environments and is a trial-and-error learning technique that enables agents to find suitable actions to maximize the total cumulative reward. Learning is difficult in partially ob-

servable environments, where the same observation may be obtained from two different states, and the agent most often requires two different actions in each state. To tackle this challenge, various modified RL algorithms are proposed (Le et al., 2018; Parisotto and Salakhutdinov, 2018; Igl et al., 2018), however, they are not suitable when on the fly decision-making is desirable. Practical reasoning is also used to enable agents to infer knowledge from their environment and interaction with other agents (Golpayegani et al., 2019). However, the uncertainty caused by partially observable environments makes reasoning more complex and leads to inconsistencies in many traditional reasoning systems.

In 1977, Feigenbaum pointed out that artificial intelligence systems' power lies in their ability to encode and exploit domain-specific knowledge, leading to the paradigm that "in the knowledge lies the power" (Feigenbaum, 1977). Domain-specific knowledge is often encoded in ontologies. Ontology describes concepts, properties, relationships, and rules (Zouaq and Nkambou, 2010). Ontology rules are in the form of an implication between an antecedent and a consequent.

^a <https://orcid.org/0000-0003-0983-301X>

^b <https://orcid.org/0000-0003-4738-7604>

^c <https://orcid.org/0000-0002-3712-6550>

An inference engine can generate new relationships based on existing rules. There is some evidence in the literature that ontological knowledge can improve the accuracy of Machine Learning (ML) algorithms (Caruana et al., 2013; Motta et al., 2016), however, the concerns related to partial observability are not addressed.

In this paper, we focus on partially observable environments in which agents cannot access full observation, which may affect their decision-making. We present an **Ontology-based Observation Augmentation Method (OOAM)** that improves agents' action selection in a partially observable environment. To do so, agents map their observations to an ontology-based schema by specifying concepts, relationships, and properties. Then they use an inference engine to extract implicit observation data to be used alongside their learning algorithm. A Job Shop Scheduling (JSS) environment with a high level of partial observability is chosen as a case study for this paper. Our method performance is compared to Trust Region Policy Optimization (TRPO) learning algorithm. The results show that OOAM can improve the agent's observation and outperforms the baseline method.

The paper is organized as follows. A running example is described in Section 2. A review of relevant literature is presented in Section 3. Section 4 briefly presents the required background knowledge. Section 5 describes our method and how it works. Section 6 defines the case study and analyses the results. Finally, our conclusion and future works are discussed in Section 7.

2 RUNNING EXAMPLE: JOB SHOP SCHEDULING

JSS is a problem where multiple jobs/orders are processed on several machines and several operational steps must be included in each job, each of which must be completed in a specific order. For instance, the job may involve manufacturing consumer products like automobiles. Figure 1 illustrates this environment. The job shop scheduler agent is responsible for scheduling jobs so that all of them can be completed in the shortest amount of time.

As orders are generated by sources, they are placed in the queue. A job shop scheduler agent then selects which order to send from the queue to which machine for the next operation step. The processed orders will then be sent to the sinks for consumption. JSS is a suitable case study for this paper as it requires

the learning agents to operate in partially observable environments.

3 RELATED WORK

RL techniques used in partially observable environments estimate unobservable state variables. In (Le et al., 2018), the authors proposed a hierarchical deep RL approach for learning, where the learning agents need an internal state to memorize important events in partially observable environments. In (Oh et al., 2016; Parisotto and Salakhutdinov, 2018), the authors built a memory system to learn to store arbitrary information about the environment over numerous steps to generalize such information for the new environments. Recurrent Neural Networks (RNNs) can compensate for missing information by developing their internal dynamics and memory (Duell et al., 2012; Hausknecht and Stone, 2015; Mnih et al., 2016). Therefore, in many of the current works, the RL agents require some form of memory to learn optimal behaviors over numerous steps. Furthermore, model-based techniques are used where the agent requires learning a suitable model of the environment first.

In all of these current learning techniques, tackling partial observability is the main challenge when on the fly decision-making is required (Dulac-Arnold et al., 2021). To improve RL's performance, agents require mechanisms to augment their observation on the fly. In a JSS environment, for example, we may not have observations for all machines/orders because they are delayed, corrupted, or even missing (Waschneck et al., 2016). Partial observability is the essential characteristic of the JSS environment which can hugely impact the scheduling decision-making process (Buchmeister et al., 2017; Pfister et al., 2018). In the JSS environment, these partial observabilities may appear as dynamic/stochastic events such as machine failure, longer-than-expected processing times, and urgent orders.

The term ontology refers to the semantics of data that are machine-understandable and contributed by users (Fong et al., 2019). In ML, domain-specific knowledge encoded with ontology can be used to constrain search and find optimal or near-optimal solutions faster or find a solution that is generalized better (Kulmanov et al., 2021). As part of ML tasks, ontologies are used in several ways, including enrichment of features derived from the ontology, calculating similarities or distances between instances based on the ontology's structure and knowledge, and determining probabilistic dependencies between instances and features based on the entities' dependencies in

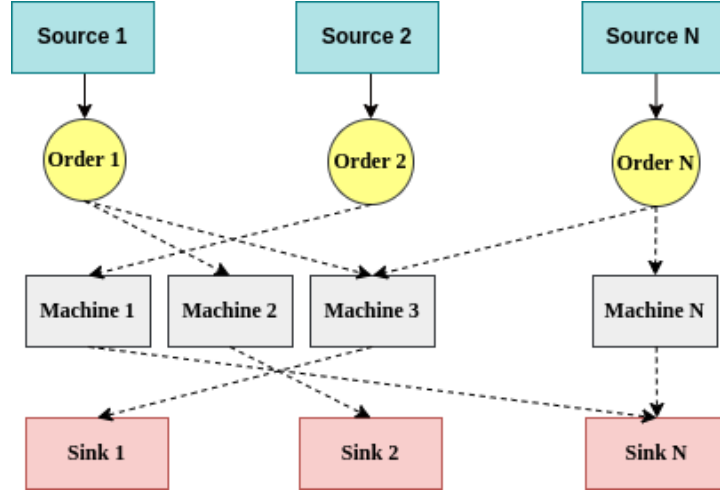


Figure 1: Job shop scheduling environment. Dot lines indicate possible operations.

the ontology (Bloehdorn and Hotho, 2009). (Motta et al., 2016) proposed a summarization method that reinforces learning functions by using semantic relationships between ontology concepts. The authors of (Youn and McLeod, 2007) created an ontology base on classification results and queried this ontology instead of querying the decision tree. To improve the accuracy of an SVM algorithm, (Caruana et al., 2013) used ontology augmentation as a feedback loop. As described in (Ghanadbashi and Golpayegani, 2022), an automatic goal-generation model is described to address the emergent requirements in unknown situations, and ontology, inference rules, and backward reasoning are used to define a new goal. In (Ghanadbashi and Golpayegani, 2021), using an environment ontology, an Ontology-based Intelligent Traffic Signal Control (OITSC) model is proposed, which enhances the RL traffic controllers' observation and improves their action selection when traffic flow is stochastic and dynamic. Even though the use of ontology in these works has shown promising results, they use in very controlled environments and simplifying assumptions. This paper aims to evaluate the proposed augmentation method in a more complex scenario (i.e., multiple noisy parameters must be considered by the agent) in a JSS environment.

4 BACKGROUND

The required background for the proposed method is discussed as follows.

4.1 Partially Observable Markov Decision Process (POMDP)

Partially observable problems are typically formulated as a Partially Observable Markov Decision Process (POMDP). A POMDP provides a discrete-time stochastic control process that describes an environment mathematically. In the standard formulation of POMDP $(\Omega, \mathcal{S}, \mathcal{A}, r, p, \mathcal{W}, \gamma)$, 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 instant reward $r^t = r(s^t, a^t) \in \mathbb{R}$ and transitions to a next state $s^{t+1} \sim p(\cdot | s^t, a^t) \in \mathcal{S}$. $\pi: \mathcal{S} \mapsto P(\mathcal{A})$ denotes a policy in which $P(\mathcal{A})$ represents distributions over the action space \mathcal{A} . The discounted cumulative reward under policy π is $R(\pi) = \mathbb{E}_\pi [\sum_{t=0}^{\infty} \gamma^t r^t]$, where $\gamma \in [0, 1)$ is a discount factor. Ω is the possibly infinite set of observations, and $\mathcal{W}: \mathcal{S} \rightarrow \Omega$ is the function that generates observations o based on the unobservable state s of the process through a set of conditional observation probabilities. At each time, the agent receives an observation $o \in \Omega$ which depends on the new state of the environment, s' , and on the just taken action, a , with probability $\mathcal{W}(o, a, s')$ (Sutton and Barto, 2018). The objective of RL is to search for a policy π that achieves the maximum cumulative reward $\pi^* = \arg \max_\pi R(\pi)$. For convenience, under policy π we define action value function $Q^\pi(s, a) = \mathbb{E}_\pi [R(\pi) | s_0 = s, a_0 = a]$ and value function $V^\pi(s) = \mathbb{E}_\pi [R(\pi) | s_0 = s, a_0 \sim \pi(\cdot | s_0)]$. We also define the advantage function $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$ (Schulman et al., 2015).

4.2 Trust Region Policy Optimization

Approximating π^* can be accomplished using a direct policy search within a given policy class

$\pi_\theta, \theta \in \Theta$, where Θ represents the policy's parameter space. We can update the parameter θ with policy gradient ascent, by computing $\nabla_\theta R(\pi_\theta) = \mathbb{E}_{\pi_\theta} [\sum_{t=0}^{\infty} \gamma^t \nabla_\theta \log \pi_\theta(a^t | s^t)]$, then updating $\theta_{\text{new}} \leftarrow \theta + \alpha \nabla_\theta R(\pi_\theta)$ with some learning rate $\alpha > 0$. Alternatively, we could consider first a trust region optimization problem (Schulman et al., 2015):

$$\max_{\theta_{\text{new}}} \mathbb{E}_{\pi_\theta} \left[\frac{\pi_{\theta_{\text{new}}}(a^t | s^t)}{\pi_\theta(a^t | s^t)} A^{\pi_\theta}(s^t, a^t) \right] \quad (1)$$

$$\|\theta_{\text{new}} - \theta\|_2 \leq \varepsilon$$

for some $\varepsilon > 0$. If we do a linear approximation of the objective in Equation 1, $\mathbb{E}_{\pi_\theta} \left[\frac{\pi_{\theta_{\text{new}}}(a^t | s^t)}{\pi_\theta(a^t | s^t)} A^{\pi_\theta}(s^t, a^t) \right] \approx \nabla_\theta R(\pi_\theta)^T (\theta_{\text{new}} - \theta)$, we recover the policy gradient update by properly choosing ε given α .

Trust Region Policy Optimization (TRPO) uses information theoretic constraints rather than Euclidean constraints (as in Equation 1) between θ_{new} and θ to better capture the geometry on the parameter space induced by the underlying distributions (Schulman et al., 2015). In particular, consider the following trust region formulation:

$$\max_{\theta_{\text{new}}} \mathbb{E}_{\pi_\theta} \left[\frac{\pi_{\theta_{\text{new}}}(a^t | s^t)}{\pi_\theta(a^t | s^t)} A^{\pi_\theta}(s^t, a^t) \right] \quad (2)$$

$$\mathbb{E}_s [\text{KL}[\pi_\theta(\cdot | s) || \pi_{\theta_{\text{new}}}(\cdot | s)]] \leq \varepsilon,$$

where $\mathbb{E}_s[\cdot]$ is the state visitation distribution induced by π_θ . By enforcing the trust region with the KL divergence, the update according to Equation 2 optimizes a lower bound of $R(\pi_\theta)$ during training to avoid taking large steps that irreversibly degrade policy performance as with vanilla policy gradients (see Equation 1).

4.3 Ontology

Ontology is used when a semantic description is needed (for example, when interpreting an unforeseen event is required) (Zouaq and Nkambou, 2010). An ontology describes concepts C , properties F , relationships E , and logical rules J . Relationships express which concepts are associated with which concepts/values by which properties ($E \subseteq C \times F \times C$). The domain and range of a relationship determine what kind of instances it can be used for (i.e., domain) and what kind of values it can have (i.e., range) (Horrocks et al., 2004). To develop an ontology, the ontology development 101 strategy (Noy et al., 2001) and the ontology editing environment Protégé (Musen, 2015) can be used. In addition, the SABiO guidelines for ontology Verification and Validation (V&V) are used for the evaluation of ontolo-

gies (i.e., for identifying missing or irrelevant concepts) (de Almeida Falbo, 2014).

The **inference engine** is the part of an intelligent system that infers new information based on known facts, using logical rules. Ontology engineers manually assert the logical rules using Semantic Web Rule Language (SWRL) for the inference engine to compare them with facts in the knowledge base. When the IF (condition) part of the rule matches a fact, the rule is fired and its THEN (action) part is executed.

The Modus Ponens rule is one of the most important rules of inference, and it states that if “A” and “A \rightarrow B” is true, then we can infer that “B” will be true. If “A” implies “B”, then “A” is called the antecedent, and “B” is called the consequent. An inference engine can search for an answer using two basic approaches. These are:

Forward Chaining: Infer from logical rules in the knowledge base in the forward direction by applying Modus Ponens rule to extract more data until a goal is reached.

Example:

“A” Machine capacity is full.

“A \rightarrow B” If the machine capacity is full, then it is in a working status.

\Rightarrow

“B” Machine is in a working status.

Backward Chaining: Starts with a list of goals and works backward from the consequent to the antecedent to see if any known facts support any of these consequences.

Example:

“A \rightarrow B” If the machine is broken, then it is in a failure status.

“C \rightarrow A” If the machine has metal fatigue, then it is broken.

“C” Machine has metal fatigue.

\Rightarrow

“B” Machine is in a failure status.

Figure 2 shows the ontology we created for modeling the concepts in a JSS environment. The JSS ontology describes semantics such as “an order has a processing time, which can be actual or current” as machine-understandable concepts. In ontology, there are six high-level concepts (i.e., superclass), including Source, Order, WorkArea, Group, Machine, and Sink, each related to a separate entity defined in the environment. A superclass can have subclasses representing more specific concepts than the superclass. An important type of relation is the partOf relation. This defines which subclasses are part of which superclass. For example, Generate, Consume, and Process are partOf OperationStep, which in turn is partOf Order.

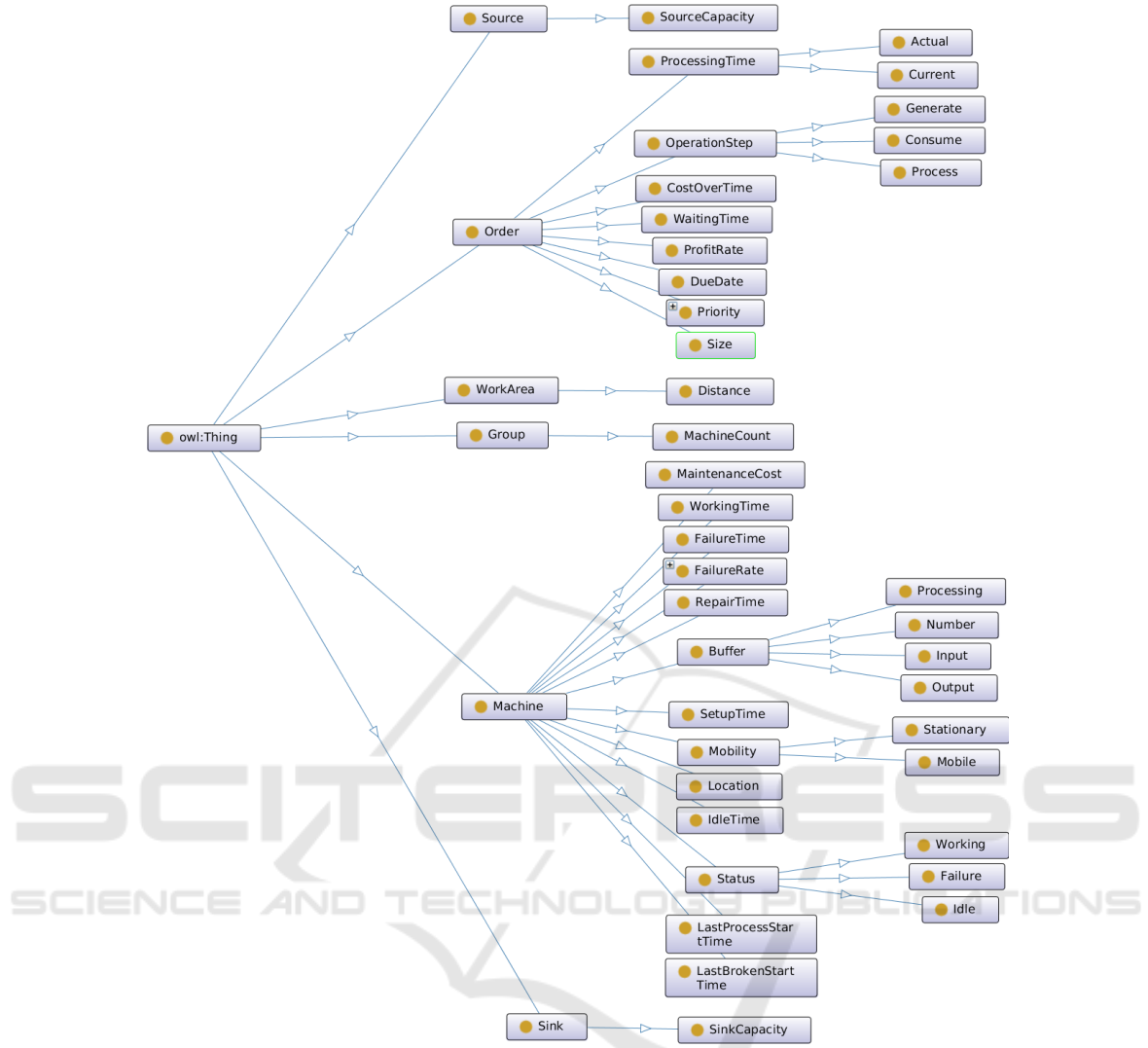


Figure 2: Ontology for job shop scheduling environment.

5 ONTOLOGY-BASED OBSERVATION AUGMENTATION METHOD

In an environment where data is partially observable, the RL agents' observations will be imperfect and noisy, and this causes uncertainty in their action selection (i.e., decision-making) process. We propose an Ontology-based Observation Augmentation Method (OOAM) to complement agents' learning by improving the agent's decision-making process on the fly when a partially observable state $s_{g_i}^t$ is observed. OOAM comprises two stages: in the first stage, the agent g_i uses the domain ontology to generate an ontology-based schema of its observation data, and

in the second stage, the agent uses inference engine to augment its partial observation with explicit data.

The following subsections describe how we have modeled the JSS agent and the two stages of OOAM.

5.1 JSS Agent

The job shop scheduler is modeled as an RL agent that receives reward and state observation from the environment and takes action accordingly. This RL agent uses TRPO as a learning algorithm and its state, action, and reward are modeled as follows:

State: We model information of each state s^t for the job shop scheduler at time step t as Equation 3. The details of the parameters are listed in Table 1.

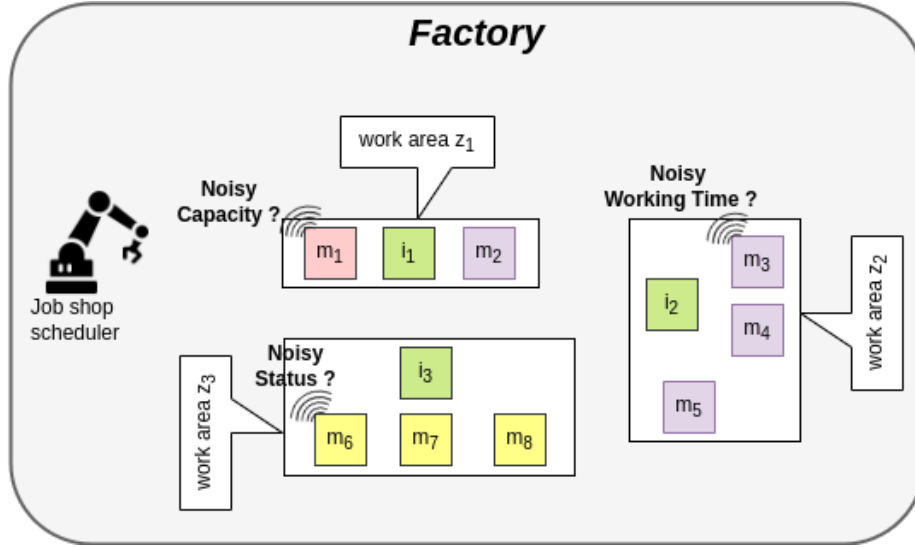


Figure 3: Job shop scheduling environment. The machines of the group n_1 are marked with purple color, group n_2 with pink color, and group n_3 with yellow color. There is noise in the machines' capacity, status, and working time data.

$$s^t = \{M^t, D^t, w_{d_i}^t, f_{d_i}^t, c_{d_i}^t, u_{m_i}^t, l_{m_i}^t, y_{m_i}^t, h_{m_i}^t, k_{m_i}^t, B_{m_i}^t, bi_{m_i}^t, bo_{m_i}^t, bp_{m_i}^t\} \quad (3)$$

In the JSS environment, some machines' capacity, status, and working time observations are noisy, so the agent's observations are partial (see Figure 3).

Action: The action is defined as choosing one of the machines for processing an order.

$$A = \{1, \dots, |M|\} \quad (4)$$

Reward: The reward function is defined as maximizing the average utilization rate of all the machines \bar{E} as shown in Equation 5:

$$R = \max(\bar{E}) \quad (5)$$

Since a failed machine cannot be assigned an order until it is repaired, the percentage utilization of an individual machine E_{m_i} is calculated based on its working time as follows (t' shows the time of the last utilization rate calculation):

$$E_{m_i} = u_{m_i}^t / (t - t' - l_{m_i}^t) \quad (6)$$

Then the average utilization rate is computed as follows:

$$\bar{E} = \frac{1}{|M|} \sum_{i=1}^{|M|} E_{m_i} \quad (7)$$

5.2 Observation Modeling

In this paper, ontology is used to enable agents to **represent and interpret** their observations. To represent an observation, they map low-level sensor data

streams to high-level concepts. Sensors often produce raw data and unstructured streams and they measure phenomena values such as waiting time of an order. Semantic Sensor Network (SSN) ontology is used to describe sensor resources and the data they collect as observations. It has been created as a standard model for sensor networks to describe sensor systems (Haller et al., 2019). We use the data model proposed in (Duy et al., 2017) to use SSN ontology with cross-domain knowledge for annotation and present sensors and sensor data (see Figure 4). We can see the `ssn:observation` class that describes sensor data observed by an agent. The `ssn:property` indicates the property (e.g., waiting time) of the feature of interest (e.g., order) that is described by a JSS ontology.

In this paper, the RL agent uses the same approach to annotate raw environment data streams by semantic description and defined by combined ontology. The environment data stream is indicated by concepts (e.g., machine, order) and their properties (e.g., waiting time, working time). We call this an ontology-based schema. For example, in the ontology-based schema for the JSS environment, "Machine" (i.e., domain) "hasStatus" (i.e., relationship) and can be a "Failure" one (i.e., range). These relationships enable concept inheritance and automated reasoning. We define $L_{g_i}^t = \{C_{g_i}^t, F_{g_i}^t, E_{g_i}^t, J\}$ as the schema describing the data observed by agent g_i at time step t . $C_{g_i}^t$ indicates the concepts, $F_{g_i}^t$ indicates the properties, and $E_{g_i}^t$ shows the relationships between them. J shows logical rules defined by ontology engineers in the domain-specific ontology. When a semantic description is needed, the agent uses an ontology-

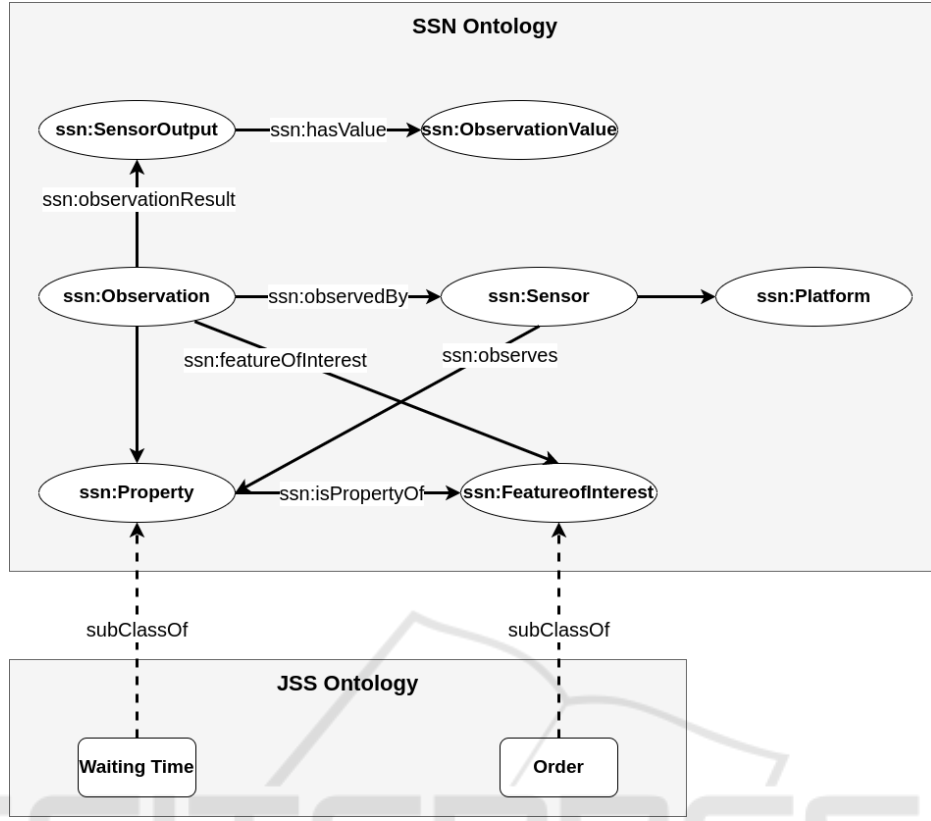


Figure 4: Concept of the data model (Duy et al., 2017).

based schema, which consists of surrounding concepts and the relationships among concepts as observed by agents. From the data with rich semantic description, the agent can interpret and reason for implicit observation data by inference engine approaches.

5.3 Observation Augmentation

In this stage, agents infer implicit information from sensed data (i.e., known facts) using forward chaining over ontology's logical rules (see Section 4.3). By applying forward chaining, to the logical rules $J_1 : c_x \rightarrow c_y$ and $J_2 : c_y \rightarrow Y$ and by assuming that concepts c_x and c_y are explicitly observed by the agent then it can deduce new logical rule $J : c_x \rightarrow Y$ that state implicit observation data for concept c_x will be Y (see Algorithm 1)¹.

In the JSS environment, we have defined several logical rules which will be used by the JSS agent. Tables 2, 3, and 4 are three examples of the observation augmentation stage that can be applied in the JSS en-

 Algorithm 1: Observation Augmentation ($s_{g_i}^t, L_{g_i}^t$).

```

1: for  $c_x$  in  $C_{g_i}^t$  do
2:   if  $c_x$  has noisy value then
3:     for  $c_y$  in  $C_{g_i}^t$  do
4:       Extract rules  $J_1, J_2, \dots, J_n$  related to  $c_x$ 
         and  $c_y$ 
5:       if New rule  $J : c_x \rightarrow Y$  is inferable then
6:         Assign value  $Y$  to  $c_x$ 
7:       end if
8:     end for
9:   end if
10: end for
    
```

vironment to deduce partial observable data.

Table 2 describes the inference rules that can infer the implicit observation related to the capacity of machine a . Suppose that the sum of orders in the input buffer x , the output buffer y , and the processing buffer z of the machine is less than the machine's capacity c . In that case, the machine's remaining capacity is free.

Table 3 describes the inference rules that can infer the implicit observation related to the status of machine a . The machine is currently failing if the last time it failed m was greater than the last time it started a process n .

¹The code of the OOAM-TRPO algorithm is publicly available: <https://github.com/akram0618/ontology-based-observation-augmentation-RL>

Table 1: Details of state representation.

Parameter	Information
M^t	List of machines at time step t .
D^t	List of orders at time step t .
$w_{d_i}^t$	The waiting time of order d_i shows the length of time the order has waited to be completely processed by a machine at time step t .
$f_{d_i}^t$	The actual processing time of order d_i shows the predetermined processing time of the order in the machine.
$c_{d_i}^t$	The current processing time of order d_i shows the length of time since the order arrived in the machine's processing buffer until now.
$u_{m_i}^t$	The working time of machine m_i shows the total processing time of the machine at time step t .
$l_{m_i}^t$	The failure time of machine m_i is calculated by taking the total time the machine failed at time step t .
$y_{m_i}^t$	The last broken start time of machine m_i indicates the last time the machine is failed.
$h_{m_i}^t$	The last process start time of machine m_i indicates the last time a process is started in the machine.
$k_{m_i}^t$	The status of machine m_i shows the machine's status, including failure, working, or idle at time step t .
$bi_{m_i}^t$	The buffer in of machine m_i indicates the number of orders in the input buffer of the machine.
$bo_{m_i}^t$	The buffer out of machine m_i indicates the number of orders in the output buffer of the machine.
$bp_{m_i}^t$	The processing buffer of machine m_i specifies whether any orders are being processed in the machine.
$B_{m_i}^t$	The capacity of machine m_i specifies the total number of orders in the machine.

Table 2: JSS inference rules related to the capacity of the machine a .

Inference rules
$\text{JobShopScheduler}(?i)^2, \text{Machine}(?a),$ $\text{hasInputBuffer}(?a, ?x), \text{hasOutputBuffer}(?a,$ $?y), \text{hasProcessingBuffer}(?a, ?z),$ $\text{hasNumber}(?x, ?n_1), \text{hasNumber}(?y, ?n_2),$ $\text{hasNumber}(?z, ?n_3), \text{hasInitialCapacity}(?a, ?c),$ $\text{hasSum}(?n_1, ?n_2, ?n_3, ?N), \text{isLess}(?N, ?c) - >$ $\text{hasRemainingCapacity}(?a, \text{Free})$

Table 3: JSS inference rules related to the status of the machine a .

Inference rules
$\text{JobShopScheduler}(?i), \text{Machine}(?a),$ $\text{hasLastBrokenStart}(?a, ?m),$ $\text{hasLastProcessStart}(?a, ?n), \text{isGreater}(?m, ?n)$ $- > \text{hasStatus}(?a, \text{Failure})$

Table 4 describes the inference rules that can infer the implicit observation related to the working time of the

²In Semantic Web Rule Language (SWRL), variables are indicated using the standard convention of prefixing them with a question mark.

machine a . Based on the actual processing time of orders in the machine's input p_2 and output buffers p_3 , as well as the order processing time at the moment p_1 , we can estimate the machine's working time.

Table 4: JSS inference rules related to working time of the machine a .

Inference rules
$\text{JobShopScheduler}(?i), \text{Machine}(?a),$ $\text{hasInputBuffer}(?a, ?x), \text{hasOutputBuffer}(?a,$ $?y), \text{Order}(?d_1, ?d_2, ?d_3), \text{inProcess}(?d_1, ?a),$ $\text{inInputBuffer}(?d_2, ?x), \text{inOutputBuffer}(?d_3, ?y),$ $\text{hasCurrentProcessingTime}(?d_1, ?p_1),$ $\text{hasActualProcessingTime}(?d_2, ?p_2),$ $\text{hasActualProcessingTime}(?d_3, ?p_3),$ $\text{hasSum}(?p_1, ?p_2, ?p_3, ?P) - >$ $\text{hasWorkingTime}(?a, ?P)$

6 EVALUATION

We evaluate OOAM in a JSS environment. We have simulated a JSS environment consisting of three sources, i_1, i_2 , and i_3 to generate orders and eight machines, $\{m_1, m_2, \dots, m_8\}$ processing orders based on the specified sequence of operations. Each machine

Table 5: Job shop scheduling environment setting.

Description		
Order load	Light	Three orders are generated at each time step.
	Heavy	Six orders are generated at each time step.
Noise level	Low	Noisy capacity data: In 0.5% of cases where a machine's capacity is full, it is observed as free. Noisy working time data: In two of eight randomly selected machines, the working time is randomly noised with a number between zero and four.
	High	Noisy capacity data: In 1% of cases where a machine's capacity is full, it is observed as free. Noisy status data: The status of four of eight randomly selected machines has noise, i.e., if the machine is a failure, it will be observed as no failure.

Table 6: The percentage change in performance metrics - The TRPO algorithm and the OOAM-TRPO algorithm - Noisy status data.

Scenario	Performance Criteria	
	Utilization Rate	Processed Orders
High-Light	10%	17%
High-Heavy	6%	12%
AVG	8%	15%

Table 7: The percentage change in performance metrics - The TRPO algorithm and the OOAM-TRPO algorithm - Noisy capacity data.

Scenario	Performance Criteria	
	Utilization Rate	Processed Orders
Low-Light	8%	11%
Low-Heavy	19%	21%
High-Light	12%	17%
High-Heavy	36%	40%
AVG	19%	22%

has one processing capacity, so only one order can be processed at a time. The capacity of sources (i.e., the number of orders generated at each time step) is set according to the scenarios defined in the following. Machines are categorized into three groups, n_1 , n_2 , and n_3 , placed at three work areas, z_1 , z_2 , and z_3 .

Scenarios: To evaluate the OOAM model, four scenarios (See Table 5) are defined to cover different order loads (i.e., number of orders per time step) and noise levels.

Baseline: TRPO algorithm (Kuhnle, 2020; Kuhnle et al., 2019) is selected as the baseline algorithm. Simulated episodes are set to 1000, and each episode has 100 simulation steps. The baseline algorithm performance is compared to our proposed model OOAM.

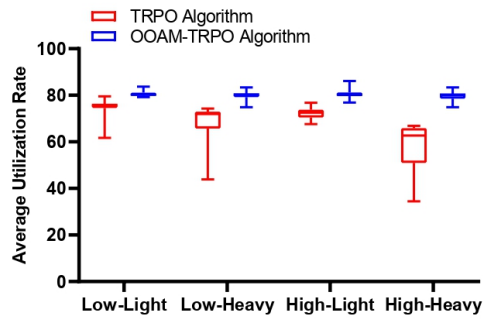
Performance Metrics: The following metrics are used to evaluate the performance of the proposed model:

- Average utilization rate of machines \bar{E} (see Equation 7).
- Total processed orders $|D_p|$: The number of or-

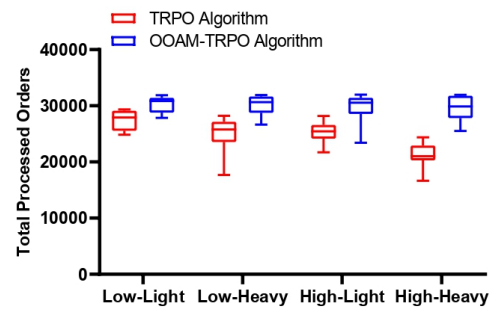
ders successfully processed.

6.1 Results and Discussion

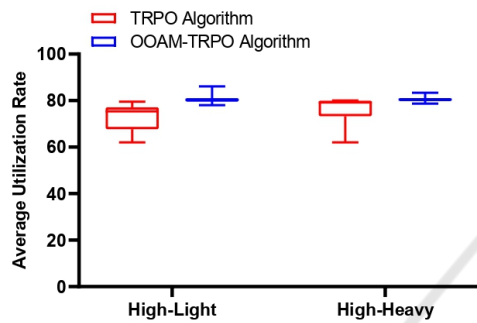
The average utilization rate and total processed orders in 10 runs in each scenario are reported in Figure 5 for the TRPO algorithm and the OOAM-TRPO algorithm. The results show that the OOAM-TRPO algorithm increases the average utilization rate and total processed orders compared to the TRPO algorithm. We observe that the average utilization rate increased by 19%, 8%, and 24%, and the total processed orders increased by 22%, 15%, and 38% in augmenting partial observable data of machines' capacity, status, and working time, respectively (see Tables 7, 6, and 8). Thus, when the job shop scheduler agent augments noisy machines' working time and capacity, the percentage change is more significant than augmenting noisy machines' status. This is because, for the agent, capacity and working time are more important parameters when choosing an appropriate machine to process orders. Also, the average utilization rate and



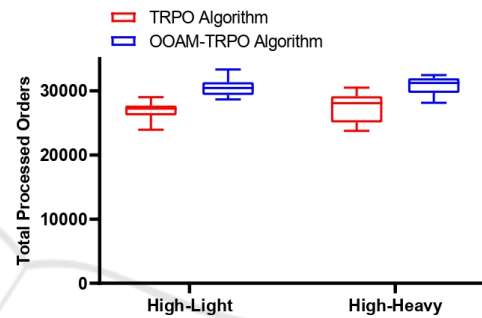
(a) Average utilization rate - Noisy capacity data.



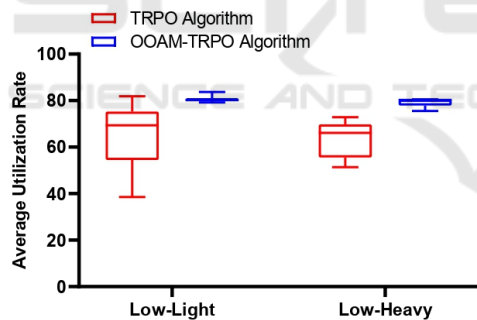
(b) Total processed orders - Noisy capacity data.



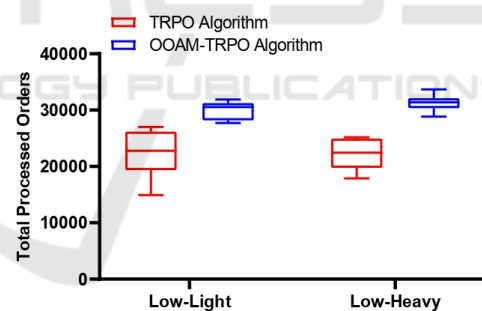
(c) Average utilization rate - Noisy status data.



(d) Total processed orders - Noisy status data.



(e) Average utilization rate - Noisy working time data.



(f) Total processed orders - Noisy working time data.

Figure 5: The TRPO algorithm and the OOAM-TRPO algorithm.

Table 8: The percentage change in performance metrics - The TRPO algorithm and the OOAM-TRPO algorithm - Noisy working time data.

Scenario	Performance Criteria	
	Utilization Rate	Processed Orders
Low-Light	23%	35%
Low-Heavy	24%	41%
AVG	24%	38%

total processed orders improve better in Heavy scenarios than in Light scenarios. The reason for this is that the number of orders generated has increased, resulting in increased times the agent has to choose machines to process orders and an increase in the impact

of improving the machines' noisy data on the agent's performance. With noisy capacity data, where there is both low and high noise, we see that the improvement for High scenarios is more significant than for Low scenarios. Since the amount of noise is more re-

markable, an improvement in noise will significantly impact the agent's decision-making.

7 CONCLUSION AND FUTURE WORK

Many real-world problems are set in partially observable environments. Learning decision-making policies in such environments is challenging because the observation is incomplete, ambiguous, and noisy from the perspective of learning agents and hence negatively impacts agents' action selection. In this paper, the proposed Ontology-based Observation Augmentation Method (OOAM) enables agents to augment their observation through ontologies, improving their action selection in partially observable environments.

This paper can be extended in several directions. An ontology's accuracy and completeness play a significant role in OOAM's performance. Ontologies need to evolve and frequently update in dynamic environments. This problem can be addressed using ontology evolution techniques (Zablith et al., 2015). In this paper, we have only looked at partially observable environments and tested the proposed solution in job shop scheduling scenarios. The generalization of OOAM is not tested so far. However, this method can be generalized to other application domains through modeling and using the relevant ontologies and applying the inference mechanism accordingly. In our future work, OOAM will be tested in other scenarios and environments (e.g., non-deterministic environments). Also, this work can be validated further in a real-world job shop scheduling environment through experimental work. In multi-agent environments, agents can exchange their inferred knowledge of environments by an ontology-based schema. To achieve consistency in distributed systems, agents must be able to coordinate their different/conflicting understandings of the environment, which might be based on different distributed ontologies. Sensor ontology matching/alignment techniques (Xue et al., 2021) can be used for determining the correspondences between heterogeneous concepts that exist in two different ontologies. Also, literature on combining relationship information from multiple data sources to infer previously unobserved relationships could be investigated to deal with partial observability in multi-agent systems (Akdemir et al., 2020).

REFERENCES

- Akdemir, D., Knox, R., and Isidro y Sánchez, J. (2020). Combining partially overlapping multi-omics data in databases using relationship matrices. *Frontiers in Plant Science*, 11:947.
- Bloehdorn, S. and Hotho, A. (2009). Ontologies for machine learning. In *Handbook on Ontologies*, pages 637–661. Springer.
- Buchmeister, B., Ojstersek, R., and Palcic, I. (2017). Advanced methods for job shop scheduling. *Advances in Production and Industrial Engineering*, page 31.
- Caruana, G., Li, M., and Liu, Y. (2013). An ontology enhanced parallel SVM for scalable spam filter training. *Neurocomputing*, 108:45–57.
- de Almeida Falbo, R. (2014). SABiO: Systematic Approach for Building Ontologies. In *Workshop on Ontologies in Conceptual Modeling and Information Systems Engineering co-located with International Conference on Formal Ontology in Information Systems (ONTO.COM/ODISE@FOIS)*.
- Duell, S., Udluft, S., and Sterzing, V. (2012). Solving partially observable reinforcement learning problems with recurrent neural networks. In *Neural Networks: Tricks of the Trade*, pages 709–733. Springer.
- Dulac-Arnold, G., Levine, N., Mankowitz, D. J., Li, J., Paduraru, C., Goyal, S., and Hester, T. (2021). Challenges of real-world reinforcement learning: Definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468.
- Duy, T. K., Quirchmayr, G., Tjoa, A., and Hanh, H. H. (2017). A semantic data model for the interpretation of environmental streaming data. In *International Conference on Information Science and Technology (ICIST)*, pages 376–380. IEEE.
- Feigenbaum, E. A. (1977). The art of artificial intelligence: Themes and case studies of knowledge engineering. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 2. Boston.
- Fong, A. C. M., Hong, G., and Fong, B. (2019). Augmented intelligence with ontology of semantic objects. In *International Conference on Contemporary Computing and Informatics (IC3I)*, pages 1–4. IEEE.
- Ghanadbashi, S. and Golpayegani, F. (2021). An ontology-based intelligent traffic signal control model. In *International Intelligent Transportation Systems Conference (ITSC)*, pages 2554–2561. IEEE.
- Ghanadbashi, S. and Golpayegani, F. (2022). Using ontology to guide reinforcement learning agents in unseen situations. *Applied Intelligence (APIN)*, 52(2):1808–1824.
- Golpayegani, F., Dusparic, I., and Clarke, S. (2019). Using social dependence to enable neighbourly behaviour in open multi-agent systems. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 10(3):1–31.
- Haller, A., Janowicz, K., Cox, S. J., Lefrançois, M., Taylor, K., Le Phuoc, D., Lieberman, J., García-Castro, R., Atkinson, R., and Stadler, C. (2019). The modular SSN ontology: A joint W3C and OGC standard spec-

- ifying the semantics of sensors, observations, sampling, and actuation. *Semantic Web*, 10(1):9–32.
- Hausknecht, M. and Stone, P. (2015). Deep recurrent q-learning for partially observable MDPs. In *AAAI Fall Symposium Series*.
- Horrocks, I., Patel-Schneider, P. F., Boley, H., Tabet, S., Grosz, B., Dean, M., et al. (2004). SWRL: A semantic web rule language combining OWL and RuleML. *W3C Member Submission*, 21(79):1–31.
- Igl, M., Zintgraf, L., Le, T. A., Wood, F., and Whiteson, S. (2018). Deep variational reinforcement learning for POMDPs. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 2117–2126, Vienna. PMLR.
- Kuhnle, A. (2020). Simulation and reinforcement learning framework for production planning and control of complex job shop manufacturing systems. Accessed: 2021-06-01.
- Kuhnle, A., Röhrig, N., and Lanza, G. (2019). Autonomous order dispatching in the semiconductor industry using reinforcement learning. *Procedia CIRP*, 79:391–396.
- Kulmanov, M., Smaili, F. Z., Gao, X., and Hoehndorf, R. (2021). Semantic similarity and machine learning with ontologies. *Briefings in Bioinformatics*, 22(4):bbaa199.
- Le, T. P., Vien, N. A., and Chung, T. (2018). A deep hierarchical reinforcement learning algorithm in partially observable markov decision processes. *IEEE Access*, 6:49089–49102.
- Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937. PMLR.
- Motta, J. A., Capus, L., and Tourigny, N. (2016). Vence: A new machine learning method enhanced by ontological knowledge to extract summaries. In *Proceedings of the Science and Information Conferences (SAI), Computing Conference*, pages 61–70. IEEE.
- Musen, M. A. (2015). The protégé project: A look back and a look forward. *AI Matters*, 1(4):4–12.
- Noy, N. F., McGuinness, D. L., et al. (2001). Ontology development 101: A guide to creating your first ontology. Technical report, Stanford Knowledge Systems Laboratory.
- Oh, J., Chockalingam, V., Lee, H., et al. (2016). Control of memory, active perception, and action in minecraft. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.
- Parisotto, E. and Salakhutdinov, R. (2018). Neural map: Structured memory for deep reinforcement learning. In *Proceedings of the 6th International Conference on Learning Representations (ICLR)*, Vancouver. Open-Review.net.
- Pfitzer, F., Provost, J., Mieth, C., and Liertz, W. (2018). Event-driven production rescheduling in job shop environments. In *International Conference on Automation Science and Engineering (CASE)*, pages 939–944. IEEE.
- Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015). Trust region policy optimization. In *International Conference on Machine Learning (ICML)*, pages 1889–1897. PMLR.
- Stankovic, M., Krcadinac, U., Kovanovic, V., and Jovanovic, J. (2009). Intelligent software agents and multi-agent systems. In *Encyclopedia of Information Science and Technology, Second Edition*, pages 2126–2131. IGI Global.
- Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT Press.
- Waschneck, B., Altenmüller, T., Bauernhansl, T., and Kyek, A. (2016). Production scheduling in complex job shops from an industry 4.0 perspective: A review and challenges in the semiconductor industry. *SAMI@iKNOW*, pages 1–12.
- Wooldridge, M. (2009). *An introduction to multiagent systems*, pages 23–26. John Wiley & Sons.
- Xue, X., Wu, X., Jiang, C., Mao, G., and Zhu, H. (2021). Integrating sensor ontologies with global and local alignment extractions. *Wireless Communications and Mobile Computing*, 2021.
- Youn, S. and McLeod, D. (2007). Efficient spam email filtering using adaptive ontology. In *Proceedings of the 4th International Conference on Information Technology (ITNG)*, pages 249–254, Las Vegas. IEEE.
- Zablith, F., Antoniou, G., d’Aquin, M., Flouris, G., Kondylakis, H., Motta, E., Plexousakis, D., and Sabou, M. (2015). Ontology evolution: A process-centric survey. *The Knowledge Engineering Review (KER)*, 30(1):45–75.
- Zouaq, A. and Nkambou, R. (2010). A survey of domain ontology engineering: Methods and tools. In *Advances in Intelligent Tutoring Systems*, pages 103–119. Springer.