Deep Reinforcement Learning for Software Agents in Mobility on Demand

Ömer Ibrahim Erduran

Department of Computer Science, Goethe University Frankfurt, Frankfurt am Main, Germany

Abstract

A disadvantage of Software Agents is their lack of Learning capability. Integrating Machine Learning into Software Agents has been applied several times. In most of the works, one single agent is considered for this to confirm and demonstrate the functionality. Scaling the problem up to a Multi-agent scenario is a resource-intensive challenge but has not been tackled yet for more than two agents. This position paper describes an approach to integrate Deep Reinforcement Learning into a Multi-Agent System of autonomous BDI agents interacting in a Mobility on Demand application.

Keywords

BDI Agent, Deep Reinforcement Learning, Agent-Oriented Programming

1. Introduction

Autonomous Ride-hailing applications are emerging from the recent development of self-driving cars with the interactions of novel Autonomous Mobility on Demand (AMoD) systems like Ride-Sharing and Ride-Hailing. In Ride-hailing, a fleet of autonomous vehicles serves mobility for customers acting as Robotaxis. Nowadays, companies like *Waymo* actively develop this form of mobility as customer service. Waymo already uses autonomous transportation with a fleet of vehicles in multiple cities in the US[1]. The interest in Autonomous Mobility on Demand increased over time. Since the acting fleet consists of multiple cooperating autonomous vehicles, research has been done to optimize several challenges like *trip assignment, vehicle repositioning, scheduling, planning, and routing* using Machine Learning (ML) approaches [1, 2]. Building intelligent vehicles as agents is crucial for reliable and robust driving behavior in the real world. Therefore, we consider *Agent-oriented Programming* (AOP) as a suitable approach. In AOP, the most predominant Software Agent architecture is the Belief-Desire-Intention (BDI) Agent architecture. It supports the general *sense-think-act* cycle of cognitive Agents with additional components of decision-making and planning [3]. The main characteristic of Software Agents is that the capabilities have to be implemented explicitly into the Agent architecture. However, the lack of learning capability represents a disadvantage for intelligent agents as they operate in a distributed manner and have to adapt to new situations and act for the overall performance of a whole fleet. *Bordini et al.* state that the pure BDI architecture is not suitable for learning capabilities, and they suggest general approaches to support the BDI architecture with AI techniques such as Machine Learning [4]. In *Reinforcement Learning* (RL), the learning process contains the learning algorithm, the Agent's environment, and the application domain [5]. The learning problem is defined by the underlying *Markov Decision Process* (MDP). In general, the main research focus is directed toward the optimization of learning algorithms and the learning process itself. This entails various tasks such as data preparation, designing RL settings, algorithm selection, and ultimately evaluating learning outcomes within a simulated environment. The architectural components of the agent generally receive less research attention in comparison. Both concepts, BDI and RL, provide the capability that a considered agent can interact with its environment by sensing and acting. Integrating ML techniques into Software Agents is a recent and open issue in AOP [4, 6]. One of the key limitations of the BDI

LWDA'24: Lernen, Wissen, Daten, Analysen. September 23–25, 2024, Würzburg, Germany

erduran@cs.uni-frankfurt.de (Ö. I. Erduran)

0000-0002-1586-0228 (Ö. I. Erduran)

[1] https://www.waymo.com/waymo-one/

architecture is the lack of generating new plans during processing [4]. A thorough survey concerning the BDI architecture and its extension capability has been done by *Silva et al.*[7]. Our paper tackles the *Trip Assignment Problem (TAP)* with DRL. We extend Autonomous BDI Agents with the capability of DRL for AMoD. In this regard, we want to integrate the learning capabilities into a BDI Vehicle Agent. We develop the BDI agents with the *Jadex* Agent Development Framework and *Deep Java Library* (DJL)[2] for integrating Deep Reinforcement Learning. We evaluate the decision results with two alternatives, namely purely DRL and an explicitly formulated utility function in the Vehicle Agent using MATSim, a traffic simulation. Our work results in a high-level decision-making framework for processing Trip requests and the cooperation of vehicle agents. We formulate the cognitive process of an intelligent software agent that learns by interacting with its environment via DRL. The Research goal is schedule optimization on a fleet level and comparison with MATSim. Thus, our research questions are:

- How is DRL integrated into multiple BDI agents interacting with each other?
- What impact does DRL that is integrated into BDI agents have in Ride-Hailing, especially for Trip Request Assignment?
- Is the *BDI-DRL* integration approach better than conventional Scheduling approaches?

2. Background

2.1. BDI architecture

The BDI-Agent architecture has its roots in the integration of cognitive behavior into Multi-Agent Systems. The main components are the *Belief*, *Desires*, *Intentions* of an agent. A *belief base* of a vehicle agent v_i contains the facts about the environment, e.g. a statement which is true or false [8]. The agent contains multiple goals that are backed with plans. A single plan contains several steps of action. The vehicle agent processes these actions to accomplish its goals. The architecture is introduced in[8]. The BDI-DRL agent builds upon this architecture and is introduced in Section 5.

2.2. Deep Reinforcement Learning

Deep Reinforcement Learning comprises the interaction of a reinforcement learning agent in an environment where the policy function is learned by a neural network. In recent works, this learning method has shown promising results not only for domains like games [9] but for Multi-agent constellations in the ride-hailing domain [1], [10]. In our work, we will build up the utility component of the vehicle agents utilizing deep Q-learning. The data-driven approach consists of the reward function, which comprises feedback for the vehicle agent concerning its decision to accept or delegate an incoming trip. With a synthetically generated trip dataset, we will train the learning-based ML component and therefore the decision-making of the vehicle agent. This procedure also affects the further phases of the BDI cycle.

2.3. Trip Assignment in Mobility on Demand Systems

AMoD [11] comprises novel types of transport services like ride-sharing and ride-hailing. The difference between both types is that in ride-sharing, multiple customers share a single vehicle. In ride-hailing, a single customer is still transported from an origin to a destination. Since the focus lies on a single fleet of autonomous vehicle agents, we assume a cooperating setting, where the vehicles delegate new trips to each other and try to maximize the individual utility and the global fleet utility. A trip assignment is where customer requests are assigned to the operating vehicle agents. We focus solely on the ride-hailing scenario, further described in the application scenario.

[2] https://djl.ai/

3. Related Work

Broekens et al. apply RL for action-rule preferences in BDI agents to improve their behavior [12]. By developing the agents in GOAL [13], an RL approach for rule selection is demonstrated. Considering these learned rules, the corresponding actions are executed. *Singh et al.* [14] model *context conditions* for plan selection as decision trees. Context conditions are defined as boolean formulas and are generally specified at design time. Agents can learn from their experiences by estimating the probability of success for different plans. The balance between the exploration of new plans and the exploitation of known successful plans is also considered. *Faccin* demonstrates the learning procedure in plan-based learning by BDI agents [15]. *Karim et al.* have mentioned BDI plans and goals as an intuitive representation of knowledge [16]. In [17] a two-agent setting is investigated for a specific domain. Therefore, we further see our approach as possible to investigate and address its scalability. Integrating BDI agents with Machine Learning has been covered in the survey of *Erduran* [18]. The author lists different approaches in supervised and unsupervised learning as well as RL approaches for different application scenarios and claims that most of the works that have been done in this area represent demonstrations or first approaches. In [19] and [20], the vehicle fleet positioning is tackled with cluster analysis and geographic information data. In [21], decision trees are used for a non-deterministic environment which is improved incrementally. The goal is to extend the BDI architecture to provide a refinement of the agent's plans, where each plan is represented as a decision tree. Another work, where the plan component is extended is in [22]. Here, the authors also tackle the scalability of the extended agent architecture. The specific learning process is the application of decision trees for the plans in different situations. The result is demonstrated by using an energy management controller. In [23], logical decision trees and inductive logic programming are used to learn when the agent's plans are successfully executable while prevailing the practical reasoning. Therefore, a hybrid approach for integrating RL into the BDI architecture is presented in [24]. In [25] the mental state representation of the agent is used for learning purposes. Therefore, GOAL as a programming language is used. An extension of the BDI architecture is presented in [26], where the BDI architecture is described as a human-like and abstract reasoning model. Here, the author also relates the root of BDI architecture to folk psychology with Q-Learning. The work of Bosello and Ricci [27] considers the BDI architecture of *JASON* Agent development framework with the distinction of hard and soft Plans. Here, the *hard plans* are programmed explicitly, whereas the *soft plans* are learned by the agent's experience which is realized by the RL component. On a conceptual level, the capabilities of a BDI Agent are considered by the usage of the sense-think act cycle. *Khaidem et al.* [28] apply Reinforcement Learning for *MATSim* by formulating a *Partially Observable Discrete Event-based Decision Process (PODEDP).* It is a modification of a partially observable Markov Decision Process (POMDP), where the event-based nature of MATSim acts as the environment of the RL setting. In the work of *Pettit et al.* [29], mobility on demand is tackled using Deep Reinforcement Learning. Our work differentiates from previous studies by focusing on a fleet of agents that interact with one another. In addition to the scenario, we aim to advance scalability by optimizing the driving and charging behaviors of an operational fleet of vehicles. While our approach shares similarities with existing work, including the type of dataset used and the formulation of DRL components, our planned contribution is distinct. We intend to integrate DRL with BDI agents in a multi-agent scenario, moving beyond the single-agent context to apply DRL to multiple software agents.

4. Problem formulation

The information a vehicle agent can perceive is represented in the *States.* For the composition of the state space, we consider the *Observations* the agent perceives during processing from different components. The agent has an internal state component, which has reasoning capabilities to produce an output e.g. taking an action or deciding to charge the battery. While interacting with other agents, the agent gets information about the environment, from other agents. Some elements from the problem formulation are taken from [30]. The assignment of incoming trip requests to vehicles in a service fleet is defined as

the TAP [31]. TAP is defined as follows: Given is a fleet of vehicle agents $AG = \{ag_0, ag_1, ..., ag_{N-1}\}$ of size $N \geq 2$ and a set of trip requests $Tr = \{tr_0, tr_1, ..., tr_{N-1}\}$. An i-th trip request tr_i contains the following attributes:

$$tr_i = (customerId_i, jobId_i, tripType_i, bookingTime_i, VATime_i, l_{start_i}, l_{end_i}) \tag{1}$$

A vehicle agent $ag \in AG$ is defined as:

$$ag_i = (vehicleId_i, location_i, capacity_i, availability_i) \tag{2}$$

In DRL, we consider a Markov Decision Process (MDP) which is defined by the tuple $\langle S, A, P, R, \gamma \rangle$, where $S \in \mathbb{R}^{d_s}$ denotes the state space, and $A \in \mathbb{R}^{d_a}$ denotes the action space, the transition function $P : S\text{x}A\text{x}S \to [0, 1]$ and the reward function $R : S\text{x}A \to \mathbb{R}$, as well as the discount factor $\gamma \in [0, 1)$. In short, we formulate TAP as a DRL Problem with:

1. State definition: $s_i = (battery\_level, nc\_trips, tr\_history, tr\_distance, tr\_duration, status) \in S$, The set of all possible states in the environment.

2. Action definition: $a_i = \{commit\_trip, negotiate\_trip, bid, do\_nothing, reschedule\} \in A$, the set of all possible actions the vehicle agent can take.

The Reward function R_t is defined as:

$$R_t = \begin{cases} -50, & if\ 20 \geq battery\_level > 0, \\ -100, & if\ battery\_level = 0, \\ -20, & if\ trip\_missed, \\ +10, & per\ trip\_success, \\ 0, & else. \end{cases}$$

The reward values are initially set and will be refined or adjusted later. The *action-value function* approximated by a neural network with parameters θ, representing the expected cumulative reward of taking action a in state s and following the policy thereafter. The *Bellman equation* for the Q-function is given by:

$$Q(s, a; \theta) = \mathbb{E}_{s' \sim P} \left[r + \gamma \max_{a'} Q(s', a'; \theta) \right],$$

where r is the reward, s' is the next state, γ is the discount factor, and P is the state transition probability. An *experience replay buffer* $\mathcal{B}$ is used to store transitions $(s, a, r, s', \text{done})$. The *Loss function* for the Q-network update is defined as:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s',\text{done}) \sim \mathcal{D}} \left[\left(r + \gamma \max_{a'} Q_{\text{target}}(s', a'; \theta^-) - Q(s, a; \theta) \right)^2 \right]$$

The Q-network is updated by minimizing this loss using stochastic gradient descent.

5. BDI-DRL architecture

An overview of integrating DRL into the BDI agent architecture with an extended simulation environment in the AMoD application is presented in Figure 1. Here, the BDI vehicle agent architecture is depicted that interacts with the traffic simulation MATSim via the *BDI-ABM* interface as well as communicating with other agents. The architecture is based on [8] and is extended with further components that represent the elements of DRL. These DRL components are listed in Table 1 with the decision to integrate the components into the vehicle agents internally or externally. There are different ways to combine DRL algorithms with software agents. One approach is to use DJL to integrate algorithms and methods written in other programming languages. Several ML-based Java libraries are listed in

Table 2 which can also be considered, each of them coming with individual advantages. In the table, we denoted whether the library contains RL/DRL, which programming language it is based on, and whether GPU-enabled processing is supported. After comparing these tools, we use DJL. The advantage is, that it is based on the JVM and provides a tight integration. Furthermore, algorithms written in PyTorch or TensorFlow can be integrated via a corresponding engine. The result of the DRL algorithm's action is the vehicle agent's selected plan. One plan in turn can contain a sequence of actions. The input data, as observed, is the current state of a vehicle agent. The training approach of the Deep Q-Network (DQN) is conducted as follows: During the decision process of the vehicle agent, we store its current state, the decision-making components, and the actions leading to a new state. For the new state, we calculate the immediate reward. As time series data, we can use the vehicle agent's historical actions or deliver real-world synthesized compressed trip requests. We plan to provide a direct invocation of DRL within the BDI agent architecture since this is useful when agents have to make decisions based on the neural network output. This integration offers a synergy that leverages the strengths of both approaches, resulting in enhanced decision-making capabilities. One of the key advantages of this integration is the ability to enable real-time learning and adaptation, allowing agents to evolve based on changes in their environment. This capability not only improves the agents' performance but also enables them to predict future events or behaviors by analyzing historical data.

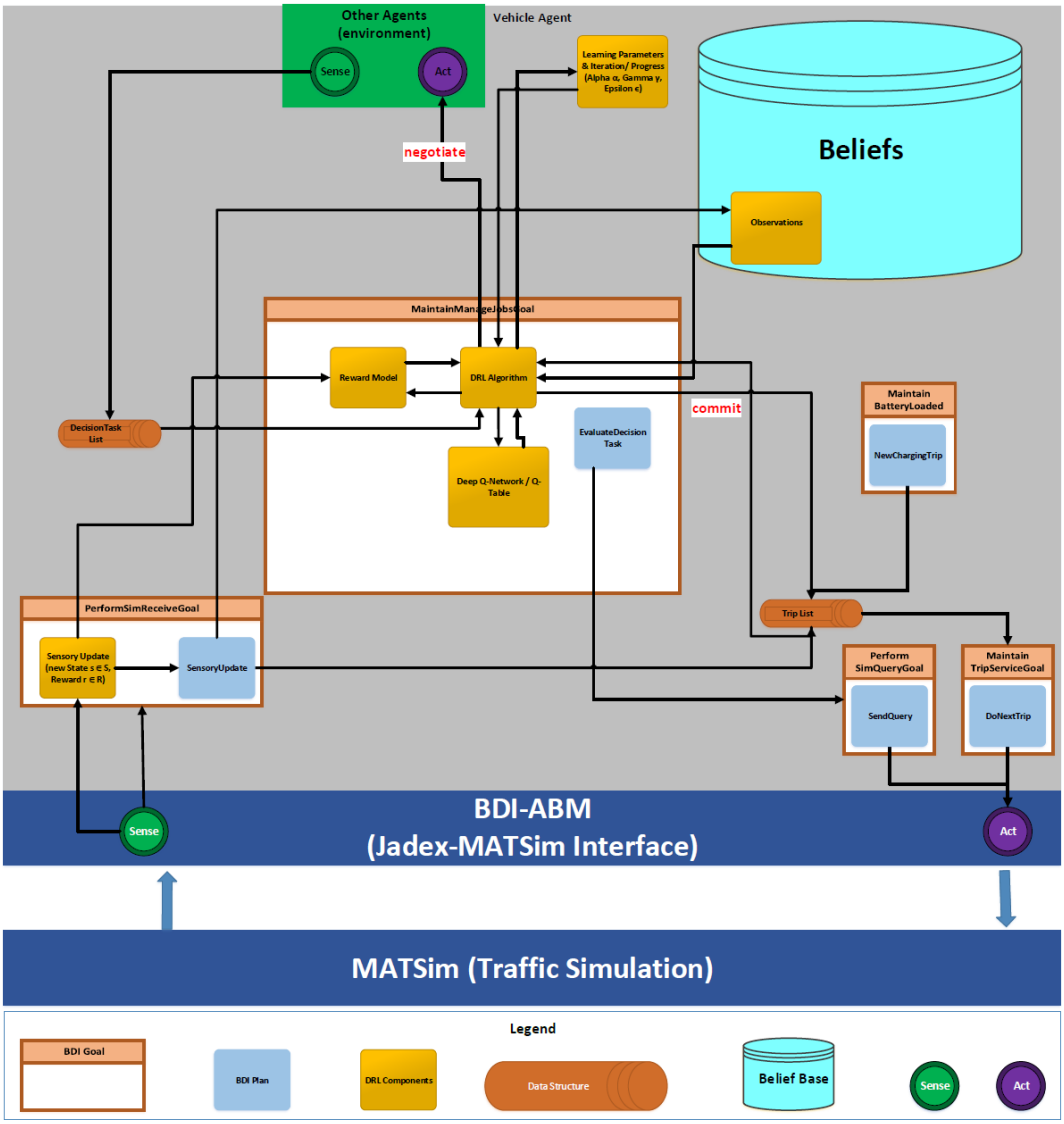

Figure 1: The BDI-DRL architecture

Table 1

Decision of integrating DRL components into the BDI Architecture

DRL Component	Jadex BDI	Integration
State	Beliefs	internal/external
Action	Plan	internal
Learning parameters (α, ϵ, γ)	-	internal/external
Reward	-	external
Reward Model	Goal/Plan	internal
DQN	Goal/Plan	internal
Value Function	Goal/Plan	internal

Table 2

Deep Learning libraries in Java

Library	Language	RL/DRL	GPU	Comment
deeplearning4j (DL4J)[3]	Java	Yes	Yes	RL components deprecated
Deep Java Library (DJL)	Java/Python	Yes	Yes	PyTorch & TensorFlow integration
TensorFlow Java[4]	Java/Python	Yes	Yes	TensorFlow on Java Virtual Machine
Weka[5]	Java	No	No	Data Mining
Neuroph[6]	Java	No	No	Last update in 2020
Encog[7]	Java	No	No	Last update in 2017
JavaML[8]	Java	No	No	Last update in 2017

6. Next steps and Discussion

The investigated platforms bring different functionalities and provide different predefined architectures therefore, investigating which approaches have been applied and also have been investigated in the research field was important to point out potential environments. Our choice is also debatable. This work represents an example of the application-oriented deployment of autonomous vehicle agents with the capabilities of DRL. As potential impacts, we see the issue of real-world deployment of learning agents, where in our agent architecture, we take advantage of cognitive functions enhancing the limits of DRL agents. To be more precise, we want to let agents learn the behavior of high-level decision-making for the given domain. This work follows the call of [4] in which the lack of mental capacity of learning agents is discussed concerning stable applications. For evaluation, the presented integration approach will be compared with a utility-based approach and with other learning algorithms. As time series data, we can use the historical actions of the vehicle agent or furthermore, deliver real-world synthesized compressed trip requests. How those trip requests are generated, is explained in [19]. In the planned experiments, we consider a comparative study with the following two types of vehicle agent configuration with Type 1: ag_{BDI} and Type 2: ag_{BDI-RL}. Type 1 represents the agent framework without DRL and thus a purely symbolic approach and Type 2 represents the presented contribution of DRL integration. For each configuration, we investigate the training phase and testing phase. We split the dataset into a training and test phase. Learning configurations and parameters will be defined. We start the training phase by performing 15.000 epochs for DRL. After each Simulation run, we compare the results with a solely BDI-based MATSim simulation and the considered data set is from the bike sharing service *Call a Bike* from Germany[10].

[3] https://github.com/deeplearning4j/deeplearning4j
[4] https://www.tensorflow.org/jvm/install
[5] https://waikato.github.io/weka-wiki/
[6] https://neuroph.sourceforge.net/download.html
[7] https://www.heatonresearch.com/encog/
[8] https://github.com/AbeelLab/javaml
[9] https://docs.oracle.com/javase/8/docs/technotes/guides/jni/
[10] https://www.deutschebahnconnect.com/en/products/call-a-bike

References

[1] K. Lin, R. Zhao, Z. Xu, J. Zhou, Efficient Large-Scale Fleet Management via Multi-Agent Deep Reinforcement Learning, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM, London United Kingdom, 2018, pp. 1774–1783.

[2] Z. Qin, H. Zhu, J. Ye, Reinforcement Learning for Ridesharing: A Survey (2021).

[3] M. J. Wooldridge, An introduction to multiagent systems, 2nd ed ed., John Wiley & Sons, Chichester, U.K, 2009.

[4] R. H. Bordini, A. El Fallah Seghrouchni, K. Hindriks, B. Logan, A. Ricci, Agent programming in the cognitive era, Autonomous Agents and Multi-Agent Systems 34 (2020) 37.

[5] R. S. Sutton, A. G. Barto, Reinforcement learning: an introduction, Adaptive computation and machine learning series, second edition ed., The MIT Press, Cambridge, Massachusetts, 2018.

[6] A. Ricci, S. Mariani, F. Zambonelli, S. Burattini, C. Castelfranchi, The cognitive hourglass: Agent abstractions in the large models era, AAMAS '24, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 2024, p. 2706–2711.

[7] L. d. Silva, F. Meneguzzi, B. Logan, BDI Agent Architectures: A Survey, in: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, International Joint Conferences on Artificial Intelligence Organization, Yokohama, Japan, 2020, pp. 4914–4921.

[8] M. Mauri, Ö. I. Erduran, T. P. Dieu Anh, M. Minor, Integrating BDI Agents with the MATSim Traffic Simulation for Autonomous Mobility on Demand, in: LWDA: Lernen, Wissen, Daten, Analysen, 2023.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (2015) 529–533.

[10] X. Tang, Z. T. Qin, F. Zhang, Z. Wang, Z. Xu, Y. Ma, H. Zhu, J. Ye, A Deep Value-network Based Approach for Multi-Driver Order Dispatching, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM, Anchorage AK USA, 2019, pp. 1780–1790.

[11] C. Ruch, S. Horl, E. Frazzoli, AMoDeus, a Simulation-Based Testbed for Autonomous Mobility-on-Demand Systems, in: 2018 21st International Conference on Intelligent Transportation Systems (ITSC), IEEE, Maui, HI, 2018, pp. 3639–3644.

[12] J. Broekens, K. Hindriks, P. Wiggers, Reinforcement Learning as Heuristic for Action-Rule Preferences, in: R. Collier, J. Dix, P. Novák (Eds.), Programming Multi-Agent Systems, volume 6599, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 25–40. Series Title: Lecture Notes in Computer Science.

[13] K. V. Hindriks, J. Dix, GOAL: A Multi-agent Programming Language Applied to an Exploration Game, in: O. Shehory, A. Sturm (Eds.), Agent-Oriented Software Engineering, Springer Berlin Heidelberg, Berlin, Heidelberg, 2014, pp. 235–258.

[14] D. Singh, S. Sardina, L. Padgham, S. Airiau, Learning Context Conditions for BDI Plan Selection (2010) 9.

[15] J. G. Faccin, Preference and Context-based BDI Plan Selection using Machine Learning: from Models to Code Generation (2016).

[16] S. Karim, B. Subagdja, L. Sonenberg, Plans as Products of Learning, in: 2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, IEEE, Hong Kong, China, 2006, pp. 139–145.

[17] S. Pulawski, H. K. Dam, A. Ghose, BDI-Dojo: developing robust BDI agents in evolving adversarial environments, in: 2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C), IEEE, DC, USA, 2021, pp. 257–262.

[18] Ö. I. Erduran, Machine Learning for Cognitive BDI Agents: A Compact Survey, in: ICAART (1), 2023, pp. 257–268.

[19] Ö. I. Erduran, M. Minor, L. Hedrich, A. Tarraf, F. Ruehl, H. Schroth, Multi-agent Learning for

Energy-Aware Placement of Autonomous Vehicles, in: 2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA), IEEE, Boca Raton, FL, USA, 2019, pp. 1671–1678.
[20] Ömer Erduran., M. Mauri., M. Minor., Negotiation in ride-hailing between cooperating bdi agents, in: Proceedings of the 14th International Conference on Agents and Artificial Intelligence - Volume 1: ICAART, INSTICC, SciTePress, 2022, pp. 425–432.
[21] S. Airiau, L. Padgham, S. Sardina, Incorporating Learning in BDI Agents (2008).
[22] D. Singh, L. Padgham, B. Logan, Integrating BDI Agents with Agent-Based Simulation Platforms (JAAMAS Extended Abstract) (2017).
[23] A. Guerra-Hernández, A. El Fallah-Seghrouchni, H. Soldano, Learning in BDI Multi-agent Systems, in: J. Dix, J. Leite (Eds.), Computational Logic in Multi-Agent Systems, volume 3259, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 218–233.
[24] A.-H. Tan, Y.-S. Ong, A. Tapanuj, A hybrid agent architecture integrating desire, intention and reinforcement learning, Expert Systems with Applications 38 (2011) 8477–8487.
[25] D. Singh, K. V. Hindriks, Learning to Improve Agent Behaviours in GOAL, in: D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, M. Dastani, J. F. Hübner, B. Logan (Eds.), Programming Multi-Agent Systems, volume 7837, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 158–173.
[26] E. Norling, Folk Psychology for Human Modelling: Extending the BDI Paradigm (2004) 9.
[27] M. Bosello, A. Ricci, From Programming Agents to Educating Agents – A Jason-Based Framework for Integrating Learning in the Development of Cognitive Agents, in: L. A. Dennis, R. H. Bordini, Y. Lespérance (Eds.), Engineering Multi-Agent Systems, volume 12058, Springer International Publishing, Cham, 2020, pp. 175–194.
[28] L. Khaidem, M. Luca, F. Yang, A. Anand, B. Lepri, W. Dong, Optimizing Transportation Dynamics at a City-Scale Using a Reinforcement Learning Framework, IEEE Access 8 (2020) 171528–171541.
[29] J. F. Pettit, R. Glatt, J. R. Donadee, B. K. Petersen, Increasing performance of electric vehicles in ride-hailing services using deep reinforcement learning, arXiv:1912.03408 [cs, eess] (2019).
[30] M. Mauri, Ö. I. Erduran, M. Minor, Jadex BDI Agents Integrated with MATSim for Autonomous Mobility on Demand, EMAS@AAMAS (2024).
[31] Z. T. Qin, J. Tang, J. Ye, Deep Reinforcement Learning with Applications in Transportation, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM, Anchorage AK USA, 2019, pp. 3201–3202.