CIRP Annals - Manufacturing Technology 69 (2020) 397–400

Contents lists available at ScienceDirect

CIRP Annals - Manufacturing Technology

journal homepage: https://www.editorialmanager.com/CIRP/default.aspx

Real-time order dispatching for a fleet of autonomous mobile robots using multi-agent reinforcement learning

Andreja Malus, Dominik Kozjek, Rok Vrabič (2)*

Faculty of Mechanical Engineering, University of Ljubljana, Slovenia

ARTICLE INFO

Article history:
Available online 17 May 2020

Keywords:
Logistics
Machine learning
Distributed control

ABSTRACT

Autonomous mobile robots (AMRs) are increasingly being used to enable efficient material flow in dynamic production environments. Dispatching transport orders in such environments is difficult due to the complexity arising from the rapid changes in the environment as well as due to a tight coupling between dispatching, path planning, and route execution. For order dispatching, an approach is proposed that uses multi-agent reinforcement learning, where AMR agents learn to bid on orders based on their individual observations. The approach is investigated in a robot simulation environment. The results show a more efficient order allocation compared to commonly used dispatching rules.

1. Introduction

To respond promptly to the constantly increasing market requirements, modern production systems are becoming increasingly dynamic and complex [1]. In order to cope with the resulting requirements for the material flow in these systems, internal logistics operations are commonly performed by mobile material transport systems, whose operations are closely coupled to the production processes in plants. Consequently, coordination and control of the transport system have to cope with a high degree of uncertainty while handling unexpected events such as breakdowns, delays, rush orders, etc.

The mobile material transport systems typically consist of autonomous guided vehicles (AGVs) – driverless mobile vehicles capable of following predefined transport routes. These routes are usually defined by a magnetic tape that is placed on the floor. Recently, however, a more advanced type of vehicles called autonomous mobile robots (AMRs) is becoming increasingly utilised [2]. AMRs differ from AGVs in their navigation capabilities. They are equipped with various sensors that detect static and dynamic objects in their surroundings and thus enable autonomous localisation and navigation. Their paths are generated based on static and dynamic obstacles in real time enabling AMRs to move around freely, without predefined routes.

While the flexibility of the system is improved, real-time path generation brings additional challenges that must be met by the fleet management system (FMS), which performs activities such as transport order dispatching, vehicle routing, and task execution scheduling. In the case of AMRs, these can be closely coupled, which leads to high computational complexity of the overall system. For example, even assigning a few transport orders to a couple of AMRs suffers from a combinatorial explosion when all possible AMR paths and time windows for their execution are considered. Therefore, centralised AMR fleet management and order execution optimisation often fail to perform in real time.

This can be addressed by decoupling the FMS activities and considering them separately. For example, in order to consider dispatching separately, routing and scheduling have to be excluded from the problem formulation. From the perspective of dispatching, an AMR only needs to be able to move from point A (e.g. pick-up location) to point B (e.g. drop-off location). Which route it takes, how it avoids obstacles, etc., is left to be solved autonomously by the AMR. Then, a centralised approach is usually used, where simple rules (heuristics) define the assignment of the orders to the AMRs. These rules usually do not take into account problem-specific settings, such as the plant layout or individual AMR capabilities, and may lead to suboptimal solutions or even to unstable behaviour of the system as a whole, e.g. due to deadlocks.

Instead, the paper proposes a distributed approach to real-time order dispatching that uses multi-agent reinforcement learning (RL). The approach enables the adaptation of dispatching to specific situations that arise as a result of the plant layout and the dynamics of the transport order arrivals. AMRs are represented by agents that learn to bid on individual transport orders based on their individual observations of the environment. The approach is validated using a physics-based AMR simulation. The results are compared with the commonly used real-time dispatching rules.

2. Order dispatching for mobile transport systems

In job shops, transport orders can be diverse, and their arrivals can be hard to predict in advance, making long-term planning difficult. Due to the above-mentioned computational complexity, it is often better to make dispatching decisions in real time, i.e. to match an order with a vehicle as soon as a new order appears,

* Corresponding author.
E-mail address: rok.vrabic@fs.uni-lj.si (R. Vrabič).

https://doi.org/10.1016/j.cirp.2020.04.001
0007-8506/

using simple rules. The search for a suitable match can be initiated on the order side, where an appropriate vehicle from the fleet has to be selected for the order, or on the vehicle side, where the most appropriate order is selected and assigned to the individual vehicle [3].

Single-attribute dispatching rules are divided into three categories: (1) distance-based, which take into account travel distances, (2) workload-based, which take into account the workstation queue occupancy, and (3) time-based rules, which take into account the order of arrival of the transport orders [4]. Evaluations using discrete-event simulations have shown that the performance of different dispatching rules is strongly influenced by various factors such as plant layout, order inter-arrival time, the ratio of AGV travel time to assembly time, etc. [5].

Selection of the best rule is not straightforward. In [6], in order to improve the efficiency of dispatching, the performance of various dispatching rules is analysed using machine learning. The selection of the dispatching rule can be even more difficult when considering freely moving robot vehicles with higher variability of travel times. In addition, a well-functioning dispatching rule applied in a dynamic and changing environment may become inadequate after a reconfiguration of the system.

Alternatively, dispatching can be approached as a distributed problem where agents [7] interact through a bidding process to assign delivery tasks or develop policies for dispatching through multi-agent reinforcement learning. Since most of the computation is distributed amongst individual agents, the benefits of this approach include better scalability and faster response to changes, without the need for system-wide re-computation. Distributed systems can also be more robust, but on the other hand, they can exhibit unpredictable emerging behaviour. However, with the development of new frameworks and learning algorithms, the capabilities of distributed approaches are rapidly increasing.

3. Multi-agent reinforcement learning

In reinforcement learning (RL), agents learn by trial-and-error, observing the environment, selecting and executing actions, and collecting rewards. Their goal is to learn to act in a way that maximizes the collected reward.

The single-agent RL problem is commonly represented as a system agent-environment, modelled as a stochastic process with the Markov property that the future state of the system depends only on the current state and not on the past states. The environment is represented as a set of states S and the agent has a set of available actions A in each state. For every action $a \in A$ that the agent can take in a state $s \in S$, there exists a probability of transition of the environment from the state s to a next state s': $P(s'|s, a)$. After each action taken, the agent receives feedback in the form of a reward $R(s', s)$. The cumulative reward collected over m consecutive steps $\sum_{k=0}^{m} r_k$ measures the performance of the agent's decision making. The action selection function in RL is represented by a policy π, which is a mapping from states to actions.

The expected value of the future reward, also called the value function, is used to evaluate the policy by assigning a numerical value to individual states or state-action pairs. A discount factor is usually used to assign a higher importance to the rewards in the near future compared to the rewards received later, which prevents infinite function values in decision problems with infinite planning horizons, thus ensuring convergence of learning. Compared to the state value function $V(s)$, the Q-value function $Q(s, a)$ defines the value of every action in every state and therefore works even if the probability of transitions between the states is unknown. The drawback, however, is that with large state and action spaces, Q-values are difficult to estimate accurately and usually require more observations by the agent.

An approach that solves the problems of large state spaces is to operate directly on policies. In this case, a policy is approximated with a differentiable parametrised function, e.g. a neural network [8], and evaluated with a performance measure $J(\pi)$, e.g. the cumulative discounted reward. The optimisation update of the policy follows the principle of stochastic gradient ascent [8]. In a set of RL methods, called actor-critic, both a value function (critic) and a policy (actor) are approximated. The value function is used for estimation of the policy gradient to update the policy and contributes to faster and more stable learning of the policy [9].

Single agent RL algorithms have shown prominent results in games with large state spaces, such as chess, Go, and Atari games [10]. RL has also been applied to real world problems, like production order dispatching [11] and global production schedule optimisation [12].

When multiple agents act in their shared environment, agents are often able to only partially observe the whole state of the environment and have policies to select actions based only on their own observations. The learning problem is then formulated as the decentralised partially observable Markov decision process (MDP) M [13], and is represented as a tuple in Eq. (1), where I is the set of agents, S is the state space, $\{A_i\}$ is a set of individual agent actions, $\{O_i\}$ is a set of agents' partial observations, P represents the transition probabilities, i.e. the environment model, R is a set of individual rewards, and γ is the discount factor.

$$M = \langle I, S, \{A_i\}, \{O_i\}, P, R, \gamma \rangle \quad (1)$$

Since multiple agents act in parallel in the same environment, the observations and the rewards received by one of the agents are not necessarily the consequence of the actions performed by that agent but are a collective result of all the actions of the agents. Hence, it is more difficult to infer the causality between actions and rewards. When multiple agents learn concurrently, each is adapting its own policy, resulting in a nonstationary environment for the other agents [14]. For the learning process, this can lead to an increase in the number of steps needed for the policies to stabilise or can prevent the convergence of the policies entirely. Furthermore, since other agents are a part of any one agent's environment, the problem space is much larger than in case of single-agent RL problems. It is therefore important to carefully define the learning problem, taking these effects into account.

4. Real-time order dispatching using multi-agent RL

To apply RL to real-time order dispatching, defining the state space together with the transitions between the states is challenging because a poor definition can suffer from a large state space or from the delay between actions and rewards that blurs causal relationships. The latter is a key issue in dispatching settings since rewards or penalties are only awarded after a transport order has been completely executed, which may be long after it has been dispatched. This issue has to be taken into account when formulating the dispatching problem.

Order dispatching is modelled as a multi-agent RL problem, where learning agents are decision-making units representing AMR vehicles and interacting with the learning environment, as shown in Fig. 1.

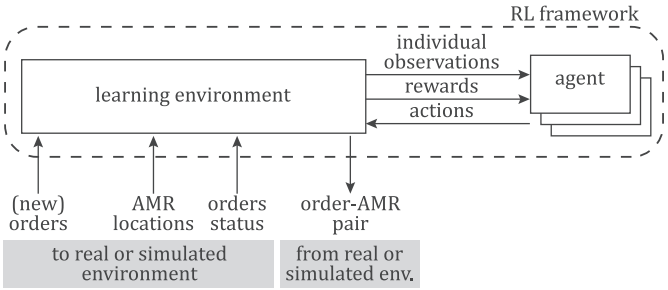

Fig. 1. RL framework architecture.

The learning environment collects information about new orders, the current locations of AMR vehicles and the status of orders already assigned. This data is used to form observations for the agents and to calculate and distribute rewards. Each agent receives its own observation and reward, and then responds back to the learning environment with its selected action. Based on the actions of the agents, movement destinations are passed to the AMRs, which, in turn, perform the transports.

A transport order T is described by the tuple, shown in Eq. (2), where x_T and y_T represent the x and y coordinates of the order's start locations, x'_T and y'_T represent the x and the y coordinates of the order's end locations, and d_T is the due time, expressed in absolute time.

$$\mathrm{T} = \langle x_T, y_T, x'_T, y'_T, d_T \rangle \tag{2}$$

The observation o_i of the agent i is a tuple, presented in Eq. (3), where x_T and y_T are the coordinates of the start location of the currently offered transport order, and x_i and y_i are the coordinates of the agent's current location. n_i is the number of orders currently assigned to the agent. x'_i and y'_i are the coordinates of the end location of the agent's last assigned transport order, in case the agent has any orders assigned ($n_i > 0$), or the agent's current location, otherwise.

$$o_i = \langle x_T, y_T, x_i, y_i, n_i, x'_i, y'_i \rangle \tag{3}$$

An agent's observation is a partial representation of the global state. Each agent has its own policy which is a mapping from observations to actions. The agent's actions are bid values that the agent proposes to the environment for the observed order. When the agent receives its observation, it selects the bid value, which is a continuous value a between 0 and 1, and passes it back to the learning environment. The offered order is assigned to the AMR of the agent, which bids the highest value amongst agents with less than two currently assigned orders. It is only allowed for the agent to have one order that is being executed and one that will be executed immediately afterwards, due to real-time dispatching constraints. As the order is assigned to the agent, it is added to the agent's assigned orders and carried out autonomously by the corresponding AMR vehicle in a first-in-first-out manner.

After the AMR has completed the transport order, the order is removed from the agent's assigned orders and the actions are evaluated by the learning environment. The reward mechanism needs to be designed so that it encourages cooperation between the agents. Therefore, all AMR agents receive the same rewards or penalties for each completed order. If an order is completed in time, all agents receive a positive constant reward, otherwise, they receive a penalty that increases quadratically with tardiness.

The problem of delayed rewards is addressed as follows. The observations are given to the agents only when an order is ready for despatch. In other words, a transition between the MDP states occurs only once per dispatched order. This significantly reduces the delay (number of state transitions) between dispatching and rewarding, which in turn permits the agents to have shorter memory for learning. A drawback of this approach is that the rewards are also given at the time of despatch, which means that they have to be accumulated in the meantime. Therefore, the agent cannot explicitly learn how it was awarded for a particular action. Another drawback is that the state transitions do not occur at regular time intervals, which can lead to smaller or larger accumulations of awards or penalties. Nevertheless, within the proposed setting, the agents will be able to assess whether their actions have led to a positive or a negative net result for the system as a whole and can still use this information to improve their policies.

For policy improvement, using a state-of-the-art RL algorithm is proposed. The Twin Delayed Deep Deterministic (TD3) policy gradient algorithm [14] is suggested. TD3 is an actor-critic algorithm for continuous action space, that concurrently approximates two Q-functions and a policy using neural networks. It is an upgrade of the DDPG algorithm [9], which has shown great performance in environments with continuous action spaces but is sometimes sensitive to hyperparameters and other kinds of tuning, making its application difficult in some cases.

The proposed RL framework learns to assign arriving orders to AMRs based on order and AMR locations, and AMR immediate plans. It is therefore expected that it will learn to take the layout into account. This is validated by simulation experiments.

5. Simulation experiments

The RL framework was implemented using the RLlib library [15], which enables implementation of RL with distributed and scalable computation, tuning of learning parameters, and use of RL algorithms for multi-agent applications. The architecture of the simulation of the AMR fleet is presented in Fig. 2.

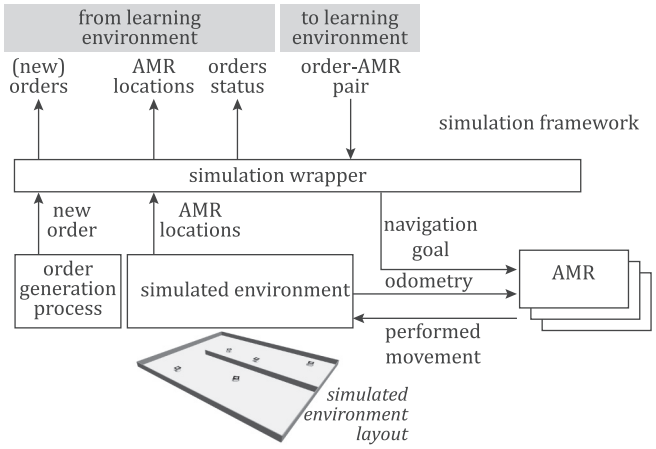

Fig. 2. Simulation framework architecture.

Two simulation frameworks were developed: a fast, but physically inaccurate one for data generation and learning, written in Python, and a computationally more intensive physics-based simulation framework for validation of the learned policies, developed using Robot Operating System (ROS) and the Gazebo simulator. In both cases, the simulation with two main components, transport order generation process and transport order execution, is controlled by the simulation wrapper which communicates directly with the learning environment.

Transport order arrivals are modelled as a Poisson process with arrival rate λ. The due time of the order was set in proportion to the Euclidian distance between the start and the end locations of the order. A fleet of five AMRs was simulated. The layout was rectangular with a wall that partially divides the upper and lower halves, allowing free movement on one side of the layout but restricting the passage on the other, as illustrated in Fig. 2. This simple layout does not promote the issues of deadlock and congestion to emerge, however, evasive manoeuvring still occurs regularly. The TD3 learning algorithm was used, with neural networks using two hidden layers with 400 neurons each. Each episode ran until 10 random transport orders were completed. The fast simulation took several hours to complete one million orders on a high-end PC, while the extensive simulation was approximately 100 times more computationally intensive.

6. Results

The results of the simulation experiments are shown in Fig. 3. In Fig. 3(a) an example of the RL learning curve is presented, showing how the mean reward per episode increases with the number of decision steps. Although the value of the reward per episode levels off at around 10 in about 300,000 steps, the policy is still refined thereafter, but with little effect.

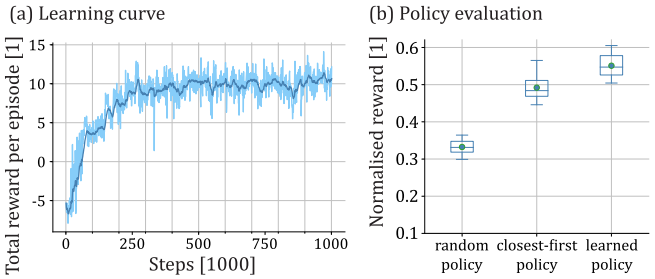

Fig. 3. (a) Policy training in the fast simulation and (b) policy evaluation in the physics-based simulation.

Since, in the proposed setting, the AMR agents bid on each individual order when it arrives, dispatching rules are selected for comparison and performance evaluation that apply to the same conditions. The learned policy is compared to the random policy, that assigns the order to a randomly selected free AMR, and to the closest-first policy, which is the implementation of the nearest-vehicle-first rule, in Fig. 3(b). It is shown that the learned policy slightly outperforms the closest-first policy (0.55 vs. 0.49 of maximum possible reward).

To gain an insight into the learning process, the evolution of the policy of one of the agents is shown using heatmaps in Fig. 4. The heatmaps show the value of the agent's action, i.e. the bid, for an order with a fixed starting location and every possible current location of the agent.

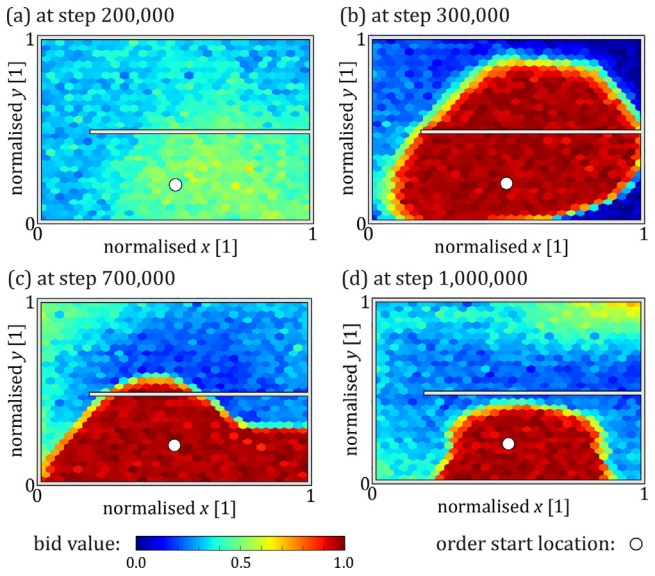

Fig. 4. Policy improvement over time (fast simulation).

After 200,000 steps (Fig. 4(a)), the agent's bids are on average low, and only slightly biased towards locations closer to the starting point of the order. After 300,000 steps, the policy becomes similar to closest-first rule, with the distance between the AMR and the pick-up point being the defining feature (Fig. 4(b)). After that, the policy improves slightly as it starts taking the layout features into account (Figs. 4(c) and (d)).

7. Conclusion

The paper presents how multi-agent RL can be applied to the problem of order dispatching for an AMR fleet. In the proposed problem abstraction, agents are given only a simple order specification. Based on their location and their immediate plans, they then learn how to form a bid for the order. With the proposed reward mechanism, all agents are rewarded for successful order completions by any single agent, thus rewarding cooperation. The training is carried out using a fast, simplified simulation. The learned policy is then transferred to a physics-based simulation of the AMR fleet, where the approach is validated. It is shown that the learned policy outperforms the closest-first rule by learning to use the features of the layout.

Since the presented approach is shown to successfully form an individual agent's policy to take layout into account this implies that the approach can be scaled to an arbitrary number of AMRs. However, additional effort is required to apply the approach to more complicated layouts. While the results in the presented simple scenario are promising, the implications of extending the approach to larger AMR fleets and more complicated layouts should be further investigated in the future. Extensions of the problem, such as consideration of orders with additional constraints or fleets of AMRs with different load-carrying capacities, should be considered.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was partially supported by the Ministry of Higher Education, Science and Technology of the Republic of Slovenia, grant no. C3330-16-529000, and by the Slovenian Research Agency, grant no. P2-0270.

References

[1] Frazzon E, Kück M, Freitag M (2018) Data-Driven Production Control for Complex and Dynamic Manufacturing Systems. *CIRP Annals* 67/1:515–518.
[2] Krüger J, Wang L, Verl A, Bauernhansl T, Carpanzano E, Makris S, Fleischer J, Reinhart G, Franke J, Pellegrinelli S (2017) Innovative Control of Assembly Systems and Lines. *CIRP Annals* 66/2:707–730.
[3] Egbelu P, Tanchoco J (1984) Characterization of Automatic Guided Vehicle Dispatching Rules. *International Journal of Production Research* 22/3:359–374.
[4] Le-Anh T, de Koster M (2006) A Review of Design and Control of Automated Guided Vehicle Systems. *European Journal of Operational Research* 171/1:1–23.
[5] De Koster R, Le-Anh T, van der Meer J (2004) Testing and Classifying Vehicle Dispatching Rules in Three Real-World Settings. *Journal of Operations Management* 22/4:369–386.
[6] Heger J, Voß T (2019) Dynamic Priority Based Dispatching of AGVs in Flexible Job Shops. *Procedia CIRP* 79:445–449.
[7] Monostori L, Váncza J, Kumara S (2006) Agent-Based Systems for Manufacturing. *CIRP Annals* 55/2:697–720.
[8] Sutton RS, McAllester D Singh SP, Mansour Y (2000) Policy Dradient Methods for Reinforcement Learning with Function Approximation. *Proceedings of Advances in Neural Information Processing Systems* 12:1057–1063.
[9] Lillicrap T, Hunt J, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D (2015) Continuous Control with Deep Reinforcement Learning. *arXiv preprint arXiv* 1509:02971.
[10] Silver D, Schrittwieser J, Simonyan K, Antonoglou I, Huang A, Guez A, Hubert T, Baker L, Lai M, Bolton A, Chen Y (2017) Mastering the Game of Go Without Human Knowledge. *Nature* 550/7676:354–359.
[11] Stricker N, Kuhnle A, Sturm R, Friess S (2018) Reinforcement Learning for Adaptive Order Dispatching in the Semiconductor Industry. *CIRP Annals* 67/1:511–514.
[12] Waschneck B, Reichstaller A, Belzner L, Altenmüller T, Bauernhansl T, Knapp A, Kyek A (2018) Optimization of Global Production Scheduling with Deep Reinforcement Learning. *Procedia CIRP* 72:1264–1269.
[13] Oliehoek F, Amato C (2016) *A Concise Introduction to Decentralized POMDPs*, Springer International Publishing.
[14] Fujimoto S, Hoof H, Meger D (2018) Addressing Function Approximation Error in Actor-Critic Methods. In: *Proceedings of International Conference on Machine Learning* 80/1587-1596.
[15] Liang E, Liaw R, Moritz P, Nishihara R, Fox R, Goldberg K, Gonzales J, Jordan M, Stoica I (2018) Rllib: Abstractions for Distributed Reinforcement Learning. In: *Proceedings of International Conference on Machine Learning* 80/3053-3062.