Dynamic Matching with Deep Reinforcement Learning for a Two-Sided Manufacturing-as-a-Service (MaaS) Marketplace

Deepak Pahwa, Binil Starly*
Edward. P. Fitts Department of Industrial and Systems Engineering
111 Lampe Drive, North Carolina State University, Raleigh, NC 27695
*Corresponding author. E-mail address: bstarly@ncsu.edu

Abstract
Suppliers registered within a manufacturing-as-a-service (MaaS) marketplace require real time decision making to accept or reject orders received on the platform. It is desirable for the suppliers to maximize the revenue received from the limited capacity they sell on the platform. Myopic decision-making such as a first come, first serve method in this dynamic and stochastic environment can lead to suboptimal revenue generation. In this paper, this sequential decision making problem is formulated as a Markov Decision Process and solved using deep reinforcement learning (DRL). Empirical simulations demonstrate that DRL has considerably better performance compared to four baselines. This early work demonstrates a learning approach for real-time online decision making for suppliers participating in a MaaS marketplace.

Keywords: Cloud manufacturing, Cyber-enabled Manufacturing, Resource Allocation, Two-sided matching, Dynamic and stochastic knapsack problem (DSKP), Cloud based design and manufacturing (CBDM)

1. Introduction
Two sided manufacturing-as-a-service (MaaS) [1] marketplaces connect clients requiring manufacturing services to suppliers providing those services. The platform removes friction in the manufacturing marketplace by providing decision-making tools such as instant quotations and order acceptance decisions. Orders and machine capacities arrive stochastically over time and suppliers need to make online real time decision making on whether to accept or reject the orders. They have limited resource availability i.e. machine capacity and they face the task of maximizing the value i.e. revenue they derive from the available resources over time. This problem is essentially a dynamic stochastic knapsack problem (DKSP) [2].

Previous works consider resource allocation as a static optimization problem [3, 4] which is not suitable for a dynamic MaaS marketplace. Dynamic methods such as rule based heuristics or rolling horizon approach [5, 6] have also been widely proposed, however, they are myopic in nature. By considering the currently available orders, expected orders and its available capacity in the future, the supplier should be able to refine its strategy to better allocate its available capacity to the incoming orders. This is especially useful in scenarios when the available capacity is limited compared to demand in the marketplace. In a marketplace setting, the capacity committed by a supplier to the marketplace is limited and it is important to devise an order acceptance policy sequentially so that the available capacity can be utilized for accepted orders considering both the immediate and future rewards. Additionally, most dynamic scheduling/ dispatching methods [6] have focused on scheduling jobs on machines in an individual small scale manufacturing system, whereas in a MaaS marketplace, the supplier's decision is based on a network level. Suppliers need to accept or reject the orders in real time and scheduling decisions can be taken later considering orders from the marketplace and other sources. In manufacturing systems with multiple suppliers, centralized approaches [7] that consider all participants (suppliers and orders) for allocation may not result in choosing an optimal action for each supplier individually. With suppliers having different preferences and objectives, decentralized [8] decision-making is desirable to maximize participation within a MaaS platform.

In a MaaS marketplace, the environment is non-stationary and the threshold will change over time. Anshelevich et al. [9] determine a threshold where an agent with a match utility less than the threshold would reject the order and wait for a better match in the future. Therefore, a learning approach where an agent can learn the optimal policy and automatically adapt to a changing environment would be more appropriate. A model based approach such as dynamic programming is not preferred as the transition dynamics are hard to capture in such an environment. This paper formulates this sequential decision making problem with a Markov Decision Process (MDP) framework and uses Deep Reinforcement Learning (DRL) to solve it. Traditional reinforcement learning methods [10] cannot handle large state spaces. Function approximation with neural networks in DRL solves the challenge of computational intractability. DRL has been used in sequential decision making for resource allocation in ride sharing [11, 12], display advertising [13, 14] and cloud computing [15, 16]. This work studies the use of a model free reinforcement learning method named Deep Q-Networks (DQN) [17].

2. Theory

Figure 1 describes the interaction between supplier agent and the environment. A MaaS marketplace is considered where orders arrive with a rate λ_d per period. An order consists of attributes such as 3D design, material requirement and due date. The supplier commits capacity to the platform for the next q periods. Capacity arrives randomly over time with a Poisson distribution parametrized with λ_s hours per period. Material availability of the supplier also changes randomly over time. The supplier quotes a price for the order depending on the utility it would derive from it. The utility depends on the order's attributes and supplier's available capacity and materials. The objective of the supplier is to maximize its revenue over a period of time T. The MDP is formulated as follows:

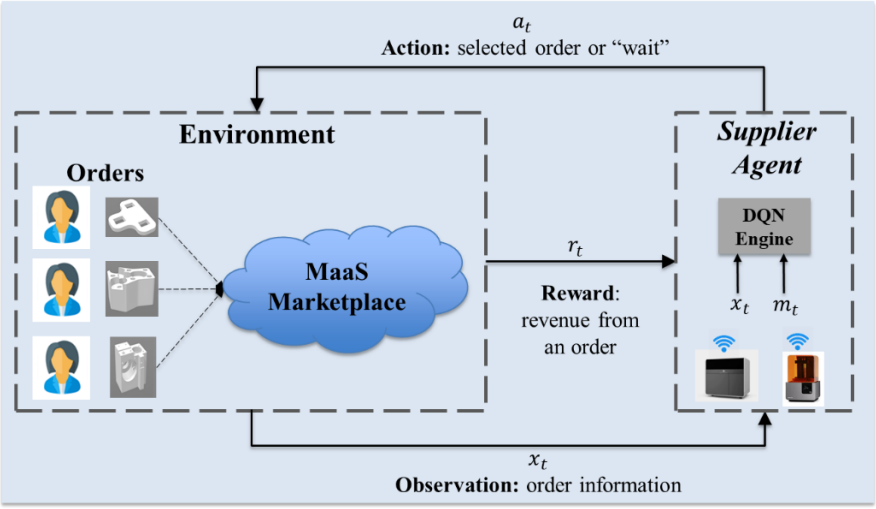

Figure 1: Agent and environment interactions in a MaaS marketplace

State: the state s_t consists of attributes of available orders x_t i.e. part volume, due date and required material, supplier attributes m_t i.e. available capacities and material availability. The number of available orders in a period can vary resulting in a change in state dimension. Neural networks for function approximation require the state dimension to be fixed. Therefore, a fixed number of orders λ_d are considered to be a part of the state and the remaining orders are assigned to a queue. The length of the queue is considered as an additional parameter in the state definition.

Action: An action a_t for the supplier agent is to choose an order to accept. As a supplier can accept multiple orders, the number of possible order combinations for a supplier can grow exponentially with the number of orders. Therefore, a supplier is allowed to choose a single order as an action to keep the number of actions linear in the number of orders. Here as well, the action space is non-stationary and to keep it fixed, the supplier is only allowed to choose from the set of orders considered in the state definition. Additionally, an action "wait" is added where the agent chooses to move to the next period and does not choose any of the available orders.

Reward: The reward function is designed as follows:

$$r_t = \begin{cases} p_i/h_i & if\ a\ valid\ order\ is\ chosen \\ -B & if\ an\ invalid\ order\ is\ chosen \\ -C * c_{t+1} & if\ "wait"\ action\ is\ chosen \end{cases}$$

where p_i is the revenue generated from order i, h_i is the production time of the order i, c_{t+1} is the available capacity in the period $t+1$, C is the penalty for each hour of capacity which goes waste (as the agent decided to move to the next period) and B is the penalty for choosing an invalid order i.e. an order for which it does not have enough capacity to manufacture it by its due date. This reward function ensures that the supplier agent maximizes its revenue over a horizon $t = 1\ to\ T$ while utilizing the capacity to its fullest.

Revenue from an order p_i depends on its utility u_i for the supplier. To determine u_i, expected utility theory [18-19] is used. u_i depends on three factors – the urgency of required delivery, material availability for the order and the available capacity of the supplier. The closer the due date of the part, the lower the utility. The higher the available capacity, the higher the utility and an order for which the required material is available has a higher utility. Revenue is considered to be utility dependent as follows:

$$p_i = bp_i + k * bp_i * (u_i)^{-l}$$
$$bp_i = up_i * v_i$$

where bp_i is the base price of the order which is determined by the volume of the part v_i and price per unit volume up_i. k and l are constants. The revenue and utility are assumed to have a nonlinear relationship with the supplier charging a higher price for a part with lower utility. Other price functions can also be considered. However, the price of the part ideally should be determined using historical data to capture the relationship between price and order/supplier attributes.

The DQN algorithm is described in Figure 2. The agent observes the state and takes an action a_t based on the ε-greedy policy. The policy chooses a_t randomly with a probability ε, and based on the policy ($a_t = \underset{a}{\text{argmax}}\, Q(s_{t+1} | \theta)$) with probability $1-\varepsilon$. The random actions allow the agent to explore the environment whereas the actions based on the policy let the agent exploit the learned policy. The value of ε starts at 1 and exponentially goes down to 0.01 over the learning duration. After the action is taken, the environment is updated and the reward r_t and the next state s_{t+1} is observed. The agent stores the transition (s_t, a_t, r_t, s_{t+1}) in the replay memory. To facilitate stable training of the DQN, we consider two networks – a predictor DQN and a target DQN. A set of transitions are sampled uniformly at random from replay memory. The predictor values y_i are determined using predictor DQN and target values $\hat{y}_i$ are determined using target DQN considering a discount factor γ. The parameters θ of the predictor DQN are updated by minimizing the mean square loss $L(\theta)$ between the target and predictor values. Adam optimizer [20] is used

for minimizing the loss function. After every k iterations, the weights of the predictor DQN are copied to the target DQN to ensure that the target values do not change frequently.

Initialize replay memory D
Use random weights θ to initialize the model
for $n = 1$ to number of episodes do:
    Reset the environment and obtain the initial state s_0
    for every period ($t = 1\ to\ T$) **do**:
        the predictor DQN model observes the state s_t and outputs action a_t based on ε-greedy policy
        the simulator updates the environment and observes the reward r_t and next state s_{t+1}
        the transition of agent (s_t, a_t, r_t, s_{t+1}) is stored in the replay memory D

    for m = 1 to M **do:**
        sample $G \sim U(D)$ transitions (s_t, a_t, r_t, s_{t+1}) from replay memory D
        for each transition in G **do**:
            use the predictor DQN model to calculate $y = Q(s_t, a_t|\ \theta)$
            use the target DQN model to calculate $\hat{y} = r_t + \gamma \max_a Q(s_{t+1}, a|\ \theta')$
            minimize mean square loss $L(\theta) = \mathrm{E}_G[(\hat{y} - y)^2]$ to update the parameters θ of the predictor DQN model

    if $\frac{n}{k} = 0$ **do**:
        copy parameters of predictor DQN network to target DQN network $(\theta = \theta')$

Figure 2: DQN algorithm for order acceptance in MaaS marketplace

The simulator used for testing the algorithm is defined in Figure 3. Orders arrive sequentially on the platform and in each period the platform takes an action to accept orders until it selects an invalid order or the action "wait". For each valid order selected, it updates the environment by updating its capacity and taking that order off from the available orders. An order from the queue moves to the set of available orders. Once, it selects an invalid order or "wait" action, the simulator moves to the next period.

Initialize states
for each period in an episode ($t = 1\ to\ T$) **do**:
Implement Policy: supplier agents selects an action as per the DQN policy. The action is either a valid order or an invalid order or "wait". The policy is implemented in a period until an invalid order or "wait" action is selected.
Update environment:
if action is a valid order:
selected order is assigned and machine capacity is updated considering the production time of the order selected
if action is invalid order or wait:
Generate orders: new orders are generated considering the distributions for order attributes such as material, due date and volume
Generate machine attributes: capacity and material availability is generated for the q_{th} period
Determine Next State: considering the available orders and machine attributes the next state for the supplier agent is generated

Figure 3: MaaS Marketplace Simulator

3. Results and Discussion

The proposed DQN method is compared against four baselines: Tabular Q-learning (TQ) where a table is used to store the $Q(s_t, a_t)$ values in contrast to the function approximation used in DQN. Since the states are stored in a table, the state definition has to be concise in order for the algorithm to be computationally tractable. State was considered to be a tuple of number of orders waiting to be assigned and the sum of available capacity rounded to the nearest integer. The second baseline is optimization using a rolling horizon approach (RHA) where the agent maximizes its revenue in each period given the constraints on capacity and due date. The third baseline is a greedy heuristic (GH) where, in each period, the agent selects a valid order which results in highest revenue per hour of capacity until there are no more valid orders available or it is out of capacity. The fourth baseline is a random algorithm which selects valid orders randomly in each period until there are no more valid orders available or it is out of capacity. The random seed in all algorithms is kept same to ensure a fair comparison.

The neural network in DQN consists of three fully connected hidden layers with 128, 64 and 32 neurons respectively. Each hidden layer uses a rectified linear unit as the activation function and the output layer uses a linear activation function. The algorithm is trained for 3,000 episodes with each episode consisting of 30 periods. Three orders arrive in each period and machine capacity arrives with a Poisson distribution having a rate of 8 hours per period. The constants for revenue function k and l are 0.5 each. The size of replay memory buffer is 20,000 and batch size is 500. The discount factor is 0.9 and the learning rate is 0.0001.

The results shown in Table 1 demonstrate that DQN performs considerably better than the baselines. Note that the DQN not only needs to learn the reward mapping with the state but also to utilize capacity efficiently over time. DQN also has a higher order acceptance rate compared to the baselines. This can be attributed to the fact that the DQN, in order to maximize the revenue over time, chose smaller orders that provide higher revenue per hour leading to higher acceptance rate. It learns to ignore orders, which provide a lower per hour revenue and then reserve the available capacity for potentially better orders in the future periods.

TQ performs slightly better than RHA but significantly worse than DQN primarily because the state definition in TQ captures very limited environment information.

Table 2: Mean and confidence intervals (CI) of normalized revenue and order acceptance rate of DQN vs baselines over 100 episodes

Algorithm	Normalized Revenue (mean)	Normalized Revenue (CI)	Order acceptance rate (mean)	Order acceptance rate (CI)
DQN	121.06	2.03	0.61	0.01
TQ	107.21	1.79	0.57	0.01
RHA	105.99	1.62	0.59	0.01
GH	101.99	1.57	0.55	0.01
Random	100.00	1.70	0.57	0.01

4. Conclusion and Future Work

Advancements in cybermanufacturing technology has given rise to MaaS marketplaces where a large number of suppliers and customers participate in a network. The platform requires online real time decision making for suppliers in order to provide value to the customers. Conventional methods fail due to computational complexity in dealing with large number of participants. This work proposes a method which enables real time decision making with a distributed learning approach where an agent learns to maximize value for a supplier in the platform over a period of time. The method needs to be tested in a large environment with real datasets where the price quote for an order i.e. the value derived from an order and production time i.e. resources consumed, can both be estimated by historical data. Another key next step is to consider a multi agent system where both suppliers and customers participate as self-interested agents.

Acknowledgement

The authors acknowledge the support from National Science Foundation Grant #1937043 and #1764025 for carrying out portions of this work.

References

1. Pahwa, D., Starly, B., & Cohen, P. (2018). Reverse auction mechanism design for the acquisition of prototyping services in a manufacturing-as-a-service marketplace. *Journal of manufacturing systems*, *48*, 134-143.
2. Kleywegt, A. J., & Papastavrou, J. D. (1998). The dynamic and stochastic knapsack problem. *Operations research*, *46*(1), 17-35.
3. Jiang, H., Yi, J., Chen, S., & Zhu, X. (2016). A multi-objective algorithm for task scheduling and resource allocation in cloud-based disassembly. *Journal of Manufacturing Systems*, *41*, 239-255.
4. Liu, Y., Wang, L., Wang, X. V., Xu, X., & Zhang, L. (2019). Scheduling in cloud manufacturing: state-of-the-art and research challenges. *International Journal of Production Research*, *57*(15-16), 4854-4879.
5. Sabuncuoglu, I., & Karabuk, S. (1999). Rescheduling frequency in an FMS with uncertain processing times and unreliable machines. *Journal of Manufacturing Systems*, *18*(4), 268-283.
6. Ouelhadj, D., & Petrovic, S. (2009). A survey of dynamic scheduling in manufacturing systems. *Journal of scheduling*, *12*(4), 417.
7. Li, K., Xiao, W., & Yang, S. L. (2019). Scheduling uniform manufacturing resources via the Internet: A review. *Journal of Manufacturing Systems*, *50*, 247-262.

8. Thekinen, J., & Panchal, J. H. (2017). Resource allocation in cloud-based design and manufacturing: A mechanism design approach. *Journal of Manufacturing Systems*, *43*, 327-338.
9. Anshelevich, E., Chhabra, M., Das, S., & Gerrior, M. (2013, June). On the social welfare of mechanisms for repeated batch matching. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*.
10. Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.
11. Al-Abbasi, A. O., Ghosh, A., & Aggarwal, V. (2019). Deeppool: Distributed model-free algorithm for ride-sharing using deep reinforcement learning. *IEEE Transactions on Intelligent Transportation Systems*, *20*(12), 4714-4727.
12. Jintao, K. E., Yang, H., & Ye, J. (2020). Learning to delay in ride-sourcing systems: a multi-agent deep reinforcement learning framework. *IEEE Transactions on Knowledge and Data Engineering*.
13. Cai, H., Ren, K., Zhang, W., Malialis, K., Wang, J., Yu, Y., & Guo, D. (2017, February). Real-time bidding by reinforcement learning in display advertising. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining* (pp. 661-670).
14. Wu, D., Chen, X., Yang, X., Wang, H., Tan, Q., Zhang, X., ... & Gai, K. (2018, October). Budget constrained bidding by model-free reinforcement learning in display advertising. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management* (pp. 1443-1451).
15. Liu, N., Li, Z., Xu, J., Xu, Z., Lin, S., Qiu, Q., & Wang, Y. (2017, June). A hierarchical framework of cloud resource allocation and power management using deep reinforcement learning. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)* (pp. 372-382). IEEE.
16. Mao, H., Alizadeh, M., Menache, I., & Kandula, S. (2016, November). Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks* (pp. 50-56).
17. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., & Petersen, S. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529-533.
18. Sanayei, A., Mousavi, S. F., Abdi, M. R., & Mohaghar, A. (2008). An integrated group decision-making process for supplier selection and order allocation using multi-attribute utility theory and linear programming. *Journal of the Franklin institute*, *345*(7), 731-747.
19. Fernandez, M. G., Seepersad, C. C., Rosen, D. W., Allen, J. K., & Mistree, F. (2005). Decision support in concurrent engineering–the utility-based selection decision support problem. *Concurrent Engineering*, *13*(1), 13-27.
20. Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.