European Journal of Operational Research 318 (2024) 408–423

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/eor

Discrete optimization

Dynamic resource matching in manufacturing using deep reinforcement learning

Saunak Kumar Panda [a], Yisha Xiang [a,*], Ruiqi Liu [b]

[a] Department of Industrial Engineering, University of Houston, Houston, TX 77004, USA
[b] Department of Mathematics and Statistics, Texas Tech University, Lubbock, TX 79409, USA

ARTICLE INFO

Keywords:
Assignment
Matching problem
Manufacturing
Markov decision process
Deep reinforcement learning

ABSTRACT

Matching plays an important role in the logical allocation of resources across a wide range of industries. The benefits of matching have been increasingly recognized in manufacturing industries. In particular, capacity sharing has received much attention recently. In this paper, we consider the problem of dynamically matching demand-capacity types of manufacturing resources. We formulate the multi-period, many-to-many manufacturing resource-matching problem as a sequential decision process. The formulated manufacturing resource-matching problem involves large state and action spaces, and it is not practical to accurately model the joint distribution of various types of demands. To address the curse of dimensionality and the difficulty of explicitly modeling the transition dynamics, we use a model-free deep reinforcement learning approach to find optimal matching policies. Moreover, to tackle the issue of infeasible actions and slow convergence due to initial biased estimates caused by the maximum operator in Q-learning, we introduce two penalties to the traditional Q-learning algorithm: a domain knowledge-based penalty based on a prior policy and an infeasibility penalty that conforms to the demand–supply constraints. We establish theoretical results on the convergence of our domain knowledge-informed Q-learning providing performance guarantee for small-size problems. For large-size problems, we further inject our modified approach into the deep deterministic policy gradient (DDPG) algorithm, which we refer to as domain knowledge-informed DDPG (DKDDPG). In our computational study, including small- and large-scale experiments, DKDDPG consistently outperformed traditional DDPG and other RL algorithms, yielding higher rewards and demonstrating greater efficiency in time and episodes.

1. Introduction

Matching plays an important role in the logical allocation of resources across a wide range of industries such as transportation, college admissions (Roth & Sotomayor, 1989), organ allocation (Roth, Sönmez, & Ünver, 2004), and online dating. In the transportation sector, matching is the core issue in ride-sharing and its many variants (e.g., carpooling, P2P (peer-to-peer) ride-sharing). Ride-sharing has successfully promoted sustainable transportation, reduced car utilization, increased vehicle occupancy, and public transit among other benefits (Mitropoulos, Kortsari, & Ayfantopoulou, 2021). Matching also plays a critical role in organ allocation with the most common example being kidney allocation, where donors and patients are matched based on their compatibility which depends on factors such as organ quality and patient condition.

The benefits of matching have been increasingly recognized in manufacturing industries. In particular, capacity sharing has received much attention recently. Capacity investment is expensive across manufacturing sectors (e.g., semiconductors, and consumer electronics). For example, a new semiconductor fab costs one to four billion dollars to build, and the price for a single machine may be as high as four to five million dollars with a high obsolescence rate (Renna & Argoneto, 2011; Wu, Hsiung, & Hsu, 2005). In recent years, manufactured products have had a short product life cycle and high demand volatility, making the capacity investment not only expensive but also risky. Capacity sharing provides a viable solution to address the capacity limit facing small manufacturers and helps alleviate the cost burdens large manufacturers carry. Further accelerating the growth of the capacity sharing market is the recent paradigm shift in the manufacturing sector to digital and cloud manufacturing (Liu, Wang, & Wang, 2018) that allows users to request services ranging from product design, manufacturing, testing, management and all other stages of a product life-cycle through the cloud. This provides the critical technical infrastructure for crowd-sourcing and matching between customers and manufacturers.

A few recent works have studied how to optimize matching in manufacturing. Yang, Chen, and Kumara (2021) study a two-sided

* Corresponding author.
E-mail addresses: spanda@uh.edu (S.K. Panda), yxiang4@uh.edu (Y. Xiang), ruiqliu@ttu.edu (R. Liu).

https://doi.org/10.1016/j.ejor.2024.05.027
Received 13 February 2023; Accepted 14 May 2024
Available online 15 May 2024

additive-manufacturing (AM) market for one period and design a bipartite matching framework to match customers with manufacturers. Pahwa, Dur, and Starly (2020) consider the bipartite matching problem in manufacturing-as-a-service marketplaces in a dynamic environment and propose approximate stable matching algorithms to optimize the revenue for the marketplace platform. While there have been some recent advances in this field, matching problems in manufacturing industries are still largely under-explored.

To meet the increasing need for resource sharing in manufacturing, we consider a dynamic manufacturing resource matching problem in a finite-time horizon. Although sharing some general characteristics with common matching problems such as heterogeneous supply–demand types and similar reward structures, resource matching in manufacturing is distinctively different in several key aspects. First, the matching of the types needs to be optimized over a finite horizon. That is, at each decision period, there are demands for manufacturing capacities that need to be fulfilled, and this matching process evolves through time. Dynamic matching needs to be differentiated from the instantaneous matching commonly seen in ride-sharing. Second, resource matching typically involves many-to-many matching, since a single order can be fulfilled by multiple manufacturers based on factors such as capacity and distance. Similarly, a single manufacturer can serve multiple customers depending on the type of orders or demand types. Third, the matching framework typically consists of large state and action spaces. State and action space sizes grow exponentially as the number of firms and demand–supply types increases, while many-to-many matching further expands the action space. Lastly, the matching of resources is constrained by manufacturers' capacities. A manufacturer can only share the capacity that is available and a customer is not incentivized to take more than the amount demanded. While feasibility is also an important consideration in an organ allocation problem, a feasibility constraint makes solving a dynamic, many-to-many matching problem significantly more difficult.

In this paper, we explicitly consider the aforementioned characteristics pertaining to resource matching in manufacturing and formulate the problem as a sequential decision process. Specifically, we consider a two-sided matching with random demands and fixed capacities over a finite-time horizon. The matching is many-to-many constrained by the demand and supply quantities. Demands for capacities are allowed to be backlogged. Each matching is associated with a reward and the objective is to maximize the expected total rewards. A challenging modeling element here concerns the transition dynamics of the matching system of interest. It is difficult to accurately model the joint distribution of all types of demands, which inevitably calls for a model-free method. Therefore, we resolve to reinforcement learning (RL) which does not require the knowledge of a probabilistic model for system transitions. Our work represents an initial attempt to solve a complex, dynamic manufacturing resource-matching problem via RL.

Our problem is distinguished by its inclusion of high-dimensional states and actions, with the action space expanding significantly as the state space increases. This challenge is further compounded by demand uncertainty over the planning horizon. Specifically, for a matching problem with m demand types and n supply types, with N_s being the supply limit and N_d being the truncated demand limit, our state space is given by $\mathcal{S} \subseteq \mathbb{R}^{N_d^m \times m}$ with cardinality $|\mathcal{S}| = N_d^m$, and the corresponding action space is given by $\mathcal{A} \subset \mathbb{R}^{N_s^{n^2}} \times m \times n$ with cardinality $|\mathcal{A}| \in \mathbb{R}^{N_s^{n^2}}$. This issue renders the problem practically intractable by any conventional exact methods or advanced MDP algorithms, especially in applications of our interest such as inventory control (Dulac-Arnold, Evans, van Hasselt, Sunehag, Lillicrap, Hunt, Mann, Weber, Degris, & Coppin, 2016; Vanvuchelen & Boute, 2022). The authors illustrate the gravity of the issue using a basic joint replenishment problem, wherein simultaneous replenishment decisions are made for multiple items. With only binary decisions (e.g., order/no order) available for each item, the total number of actions reaches up to a billion in scenarios involving up to 40 items.

While model-free RL eliminates the need for explicit modeling of transition dynamics, conventional tabular model-free methods like Q-learning, while simple and exhibiting excellent learning ability, prove inefficient in solving complex problems in large state–action environments since many of the state–actions might not have been experienced previously (Jang, Kim, Harerimana, & Kim, 2019). Moreover, Jin, Allen-Zhu, Bubeck, and Jordan (2018) have mathematically shown Q-learning to have a runtime of $\mathcal{O}(T)$ and a space of $\mathcal{O}(SAH)$, where H and T are number of steps per episode and total number of steps respectively. Furthermore, traditional Q-learning employing a greedy target policy often encounters biased estimates early in the learning process, affecting solution quality and convergence speed. To address these issues, we introduce a penalty to the Q-learning update equation to penalize actions deviating from a given prior policy. Additionally, we introduce an infeasibility penalty to tackle encountering infeasible actions. The convergence of our modified Q-learning method with a general penalty function is theoretically guaranteed, with two variations in the update rule based on the behavior of the regularization parameter, β. We augment the traditional Deep Deterministic Policy Gradient (DDPG) framework by integrating the modified update rule into its architecture, embedding it within the critic loss function to efficiently handle large state and action spaces. This adaptation allows our proposed DKDDPG to explore the solution space adeptly and identify optimal policies efficiently within the dynamic matching problem domain, improving the convergence rate while providing reasonably accurate solutions.

Our approach differs from existing methods in two significant ways. Firstly, the inclusion of a regularizer term, informed by problem-specific prior policies, is expected to enhance Q-learning performance, addressing the issues of biased initial estimates and slow convergence. This regularizer also enables the utilization of high-quality solutions from one-period matching problem solvers, guiding the agent towards optimal policies more effectively and eliminating pathological policies. Secondly, while our modified Q-learning algorithm shows promise in small-scale scenarios, scaling to large-scale matching scenarios necessitates adapting the modified update rule to appropriate deep neural networks. Training traditional deep RL methods such as DQN proves infeasible for many industrial applications that contain large action spaces as shown by Vanvuchelen and Boute (2022). Consequently, we tailor an existing deep RL method, DDPG, by integrating the modified update rule into the critic network's loss function, ensuring performance enhancement.

The main contribution of this paper is three-fold: First, we formulate the multi-period, many-to-many manufacturing resource matching problem as a sequential decision process. This is among the first efforts in providing a multi-period decision framework for resource matching in manufacturing. Second, we prove the convergence of domain knowledge-informed Q-learning algorithm, providing a performance guarantee for small-size problems that can be solved by Q-learning and theoretical guidance for designing efficient algorithms for large-size problems. Lastly, we develop our DKDDPG algorithm which utilizes the domain knowledge-informed Q-learning algorithm, and conduct a computational study to show that it obtains accurate solutions efficiently.

The remainder of the paper is organized as follows. In Section 2, we review relevant literature in the context of matching problems and discuss methods in the RL literature. In Section 3, we design the dynamic resource matching in manufacturing as a sequential decision-making problem using the conventional Markov decision process (MDP) framework. In Section 4, we introduce Q-learning with a penalty and provide theoretical results and proof for its convergence. Section 5 provides details about DDPG and its enhanced version DKDDPG and discusses its implementation for our matching problem. In Section 6, we present computational results comparing the performance of DKDDPG with contemporary RL algorithms in the context of our problem.

2. Literature review

In this section, we review two relevant literature streams: matching and reinforcement learning.

2.1. Matching problem

Matching problems can be categorized into three classes: one-to-one, one-to-many, and many-to-many. One-to-one has mostly been studied in the context of ride-sharing (Tafreshian, Masoud, & Yin, 2020), stable marriage (Knuth, 1996), kidney allocation (Roth et al., 2004), car-pooling (e.g., Uber) and home-sharing (e.g., Airbnb). The one-to-one problem is typically modeled as a bipartite matching problem where the bipartite graph $G = (\mathcal{R}, \mathcal{D}, E)$ consists of two disjoint sets of nodes: one corresponding to the set of riders ($\mathcal{R}$) and one to the set of drivers ($\mathcal{D}$). Many polynomial-time exact or heuristic methods have been developed for the bipartite matching problem (Riesen, Fankhauser, & Bunke, 2007). Tafreshian et al. (2020) review a large variety of exact and approximation algorithms for this problem with polynomial worst-case running time bounds such as greedy algorithm, iterative algorithm based on augmenting path and a scaling algorithm to name a few. Karp, Vazirani, and Vazirani (1990) propose the Ranking algorithm, a simple randomized online algorithm that achieves a competitive ratio. Antoniadis, Gouleakis, Kleer, and Kolev (2020) propose using machine learning predictions to solve the online bipartite matching and improve its performance guarantee.

Many-to-one matching problems are those where multiple matches are possible for a single node. They are commonly seen in college admissions problem designed by Gale and Shapley (1962), which is a generalization of the marriage problem. Baïou and Balinski (2000) characterize the stable admissions polytope using a system of linear inequalities. Sethuraman, Teo, and Qian (2006) associate a geometric structure to the system of inequalities and provide a simple visual proof of the integrality of the Baïou–Balinski formulation. The many-to-many matching model introduced by Roth (1984, 1985) is another generalization of the marriage model where multiple nodes on both sides of a bipartite graph can be matched. The matching of demand and supply types is a form of many-to-many matching. Many single-period demand and supply matching problems are modeled as a single-commodity network flow problem (SCNF) and a number of heuristic algorithms have been developed to solve the formulated SCNF problem. For example, to solve the single-commodity, uncapacitated, fixed-charge network flow problem, Ortega and Wolsey (2003) develop a branch-and-cut algorithm that determines the right cut set for generating inequalities.

Existing literature on matching problems, while abundant, is mostly defined in a single-period deterministic problem setting. Only a few works have considered dynamic matching in a finite horizon setting. Kurino (2014) and Bloch and Houy (2009) devise a dynamic allocation problem of on-campus student housing. In their models, agents have deterministic arrivals and departures. Ünver (2010) studies dynamic kidney exchange with inter-temporal random arrivals of patient–donor pairs with the objective of maximizing the number of matched compatible pairs. They consider a general dynamic problem from the point of view of a central authority (e.g., a hospital) whose objective is to minimize the long-run total discounted waiting cost. They separately derive efficient dynamic matching mechanisms that conduct two-way and multi-way exchanges, involving the matching of two infeasible donor–patient pairs and multiple infeasible donor–patient pairs respectively. In the context of demand–supply matching, Hu and Zhou (2022) model dynamic matching of demand–supply types as a finite-horizon problem with a stochastic dynamic program where the optimal expected total discounted surplus is maximized for each demand–supply type. They formulate the problem in an MDP framework, explore the structural properties of the optimal policy, and propose heuristic policies to solve the dynamic matching problem.

Few of the aforementioned works have holistically considered the key characteristics of matching in manufacturing (e.g., many-to-many, dynamic matching, feasibility constraint), and are therefore not applicable to manufacturing resource matching. On the other hand, limited works that consider matching in manufacturing often make simplified assumptions and are mostly limited to bipartite frameworks and single-period optimization problems. Yang et al. (2021) consider a resource allocation problem between customers and additive manufacturing (AM) manufacturers. They assume that each order demands a single-unit resource and formulate the allocation problem as an integer program. They consider one-to-one matching and the stable matching algorithm is leveraged to optimize matches between customers and AM providers. Pahwa et al. (2020) propose approximate stable matching solutions using mechanism design and mathematical programming approaches to solve a bipartite matching problem in manufacturing in a dynamic environment. The matching considered in this system is many-to-one between the orders and suppliers. Analytical models and efficient algorithms are a critical necessity for solving large-scale dynamic resource matching in manufacturing.

2.2. Reinforcement learning

RL (Sutton & Barto, 2018) has proven efficient in solving complex sequential decision problems (e.g., telecommunications, robot control, and game playing among others (Li, 2017)). RL is concerned with optimizing an agent's choice of action to maximize a cumulative reward. Unlike model-based methods such as dynamic programming, which require an exact transition probability model that is often hard to obtain in practice, model-free RL methods (e.g., Temporal difference (TD(0)) learning methods, Q-learning) aim to learn the optimal policy either online or offline without knowing the underlying transition dynamics.

Despite the fact that Q-learning is guaranteed to find the optimal solution, the runtime increases exponentially as the state and action spaces grow since Q-learning is a tabular method. Mnih et al. (2015) develop an approach to train a deep Q-network (DQN), an action-value function approximated by a convolutional neural network on the high-dimensional visual inputs of a variety of Atari games. As part of their enhancements to the original Q-learning algorithm, two key enhancements are made: experience replay buffer and target network freezing. The former is designed to reduce the instability associated with training on highly correlated sequential data. By using a target network whose weights are periodically derived from the main network, the latter addresses the instability caused by chasing a moving target. Since then, DQN has gained wide popularity for solving problems with high-dimensional sensory inputs and actions and excels at a diverse array of challenging tasks. Al-Abbasi, Ghosh, and Aggarwal (2019) develop a DQN model to learn optimal dispatch policies for ride-sharing by interacting with the environment. Gao, Gao, Hu, Jiang, and Su (2020) implement DQN for portfolio management in the stock market and observe that DQN outperforms ten other traditional strategies.

While DQN works well for high-dimensional states with a small action space, it suffers from run-time and memory issues for problems involving very large action space since it increases the size of the output layer. Alternatively, obtaining a single action array as the network output ensures higher efficiency in finding the optimal action. Lillicrap et al. (2015) adapt the idea underlying the success of DQN to complex, high-dimensional action spaces and propose an actor–critic model-free algorithm based on the deterministic policy gradient that can operate over continuous action spaces. This algorithm named deep deterministic policy gradient (DDPG) uses the same learning algorithm and network architecture as DQN. It is able to robustly solve many simulated physics tasks involving large state and action spaces.

2.2.1. RL for combinatorial optimization

Numerous advancements have occurred in solution techniques for combinatorial problems sharing similarities. Machine learning (ML) and reinforcement learning (RL) have been widely employed in addressing a variety of common combinatorial problems, including but not limited to the traveling salesman problem (TSP), vehicle routing problem, maximum cut, and minimum vertex cover problem. Bengio, Lodi, and Prouvost (2020) survey methods in ML and operations research to solve combinatorial problems in general. They review advancements such as imitation learning and experience learning using graph neural networks in the context of combinatorial optimization (CO). Wang and Tang (2021) discuss various recent deep RL methods implemented for common CO problems like capacitated vehicle routing problem (CVRP) and summarize the different deep RL methods categorically based on value and policy, such as DQN, DDPG, and other actor–critic methods. They also compare numerous neural network architectures such as pointer networks, transformers, and LSTMs implemented in conjunction with listed RL algorithms to solve common CO problems.

Delarue, Anderson, and Tjandraatmadja (2020) devise an RL framework tailored for value-based deep RL methods featuring a combinatorial action space. They formulate a CVRP as a sequential decision problem and frame the selection of actions as a mixed-integer optimization problem. Dai, Khalil, Zhang, Dilkina, and Song (2018) propose a combination of RL algorithms with graph embedding to learn greedy heuristics for solving widely known CO problems such as the Traveling Salesman Problem (TSP), Maximum cut, and Minimum Vertex cover. Barrett, Clements, Foerster, and Lvovsky (2020) introduce the ECO-DQN approach, designed to iteratively enhance solutions for the Maximum Cut problem by learning exploration strategies during testing. Their method's adaptability is highlighted by its capability to initiate from any state, demonstrating its flexibility in integration with various search heuristics. Bello, Pham, Le, Norouzi, and Bengio (2017) develop a neural combinatorial optimization algorithm that combines actor–critic methods with recurrent neural networks to solve TSP. They further discuss the possibilities of extending to other problems and emphasize the designing of feasibility for the problems. A similar solution approach was designed by Nazari, Oroojlooy, Snyder, and Takáč (2018), where they replaced a pointer network and added an attention mechanism to their RNN to expand to VRP as well. Kool, van Hoof, and Welling (2019) also implement an attention mechanism along with the REINFORCE methods to learn strong heuristics and solve a wide range of practical problems.

While the aforementioned works provide a thorough literature review of reinforcement learning (RL) applied to combinatorial optimization, our problem stands out due to its distinctive characteristics. With numerous suppliers and customers involved, our problem encompasses multi-dimensional states and actions, with both the action and state spaces expanding exponentially as the number of suppliers and customers grows. This challenge is further compounded by demand uncertainty over the planning horizon. While some cited works tackle infeasibilities, our method adopts a unique approach by proportionally penalizing such solutions and integrating an additional general convex penalty function into the Q-learning update rule, utilizing problem-specific knowledge based on the work by Fox, Pakman, and Tishby (2015). They propose a penalty-based Q-learning algorithm to penalize biased estimates due to the minimum (or maximum) operator in Q-learning. The authors use a prior policy-based penalty function to penalize deterministic policies at the beginning of learning. They show that their method reduces the bias of the value-function estimation, leading to faster convergence to the optimal value and the optimal policy. In this context, we introduce and theoretically validate a regularized Q-learning update rule based on Bertsekas and Tsitsiklis (1996) and Singh, Jaakkola, Littman, and Szepesvári (2000). Furthermore, we extend this methodology to address large-scale problems using deep neural networks, demonstrating near-optimal solutions and enhanced overall performance.

3. Problem setting

We consider a multi-period manufacturing resource matching problem. Let $\mathcal{T} = 1, 2, \dots, T$ denote the decision epochs. Let $\mathcal{D} = 1, 2, \dots, m$ represent the set of demand types for manufacturing resources and $\mathcal{S} = 1, 2, \dots, n$ represent the set of capacity supply type. The sets $\mathcal{D}$ and $\mathcal{S}$ are disjoint. The arc (i, j) represents the matching between demand type i ($i \in \mathcal{D}$) and supply type j ($j \in \mathcal{S}$). We denote the complete set of arcs by $\mathcal{A} = \{(i, j) | 1 \le i \le m, 1 \le j \le n\}$. As an example, different types of manufacturing processes such as 3D printing, extrusion, CNC machining, etc. can be considered as types of demand or capacity. Note that capacity and supply are used interchangeably in this paper.

The system state is denoted by the outstanding demand vector $\mathbf{x} = (x_1, \dots, x_m) \in \mathbb{R}^m_+$. The capacity vector is denoted by $\mathbf{c} = (c_1, \dots, c_n) \in \mathbb{R}^n_+$, where x_i and c_j are the quantity of type i demand and type j capacity available to be matched respectively in time t. We assume that demands arrive randomly at each decision epoch. We also assume that the capacities are fixed in the planning horizon because the capacity of manufacturing resources typically does not change for some time. At the beginning of period t, our state comprises of outstanding (backlogged) demand units of various types, $\mathbf{x}$. We assume that each demand will take the capacity for one period. We further assume an exogenous correlation between the distribution of demand between one period and the next, that is, they are determined outside the model and imposed on the model. Our model does not account for endogenous correlations between a matching decision in a period and the demand distribution in a future period.

We consider two widely used reward structures — the horizontal reward model (Ashlagi & Shi, 2016) and the vertical reward model (Sutton, 1986). Horizontal matching is preference-based and the reward is dependent on the matching of the demand–supply type pair. The unit matching reward, r_{ij} is based on the distance between the demand type i and capacity type j, that is, δ_{ij}. We consider a linear structure reward given by $r_{ij} = R - \delta_{ij}$, where R is the fixed prize for matching. For example, suppose there are two demand types ($i = 1, 2$) and two capacity types ($j = 1, 2$). Let $i = j = 1$ represent the extrusion process and $i = j = 2$ denote the 3D printing process. Demand type 1 will form a higher rewarding pair with capacity type 1 since they have the same preference, and a lower rewarding pair with capacity type 2. We can extend the same understanding to applications with more than two demand–supply types. Consider manufacturing demand and supply types listed in terms of material (product) upgrading as shown in Fig. 1. In this case, matches can be made based on the distance between the demand–supply types, that is, the shorter the distance and thus cost, the higher the reward for matching as we explained previously. On the other hand, vertical matching is quality-based and the reward is dependent on the quality of each demand and capacity type. Vertical matching in manufacturing could be based on aspects such as shop certifications, finishing process, experience, and inspection capabilities which involve differences in terms of quality. For example, the demand–supply types can be lined up from the best demand and supply types to the worst, in terms of quality, and have a reward value associated with each type. Vertical matching a pair would generate a cumulative reward based on the reward of the demand type and the supply type as described above. For simplicity, we consider a linearly additive function. Let $r_{ij} = f(a_i, b_j) = f_d(a_i) + f_s(b_j)$, where a_i represents the quality of demand type i and b_j represents the quality of supply type j, such that f_d and f_s are increasing in a_i and b_j respectively.

At each decision period, a decision maker needs to determine the matching quantities for each supply–demand type, $\mathbf{Q} = (q_{ij}) \in \mathbb{R}^{m \times n}_+$, where q_{ij} denotes the quantity of the ith demand matched with the jth supply type. The reward associated with each arc (i, j) is modeled based on the horizontal or vertical heterogeneity model. The reward values can be written in a matrix form as $\mathbf{R} = (r_{ij}) \in \mathbb{R}^{m \times n}$. The total matching reward is given by $\mathbf{R} \circ \mathbf{Q} = \sum_{i=1}^{m} \sum_{j=1}^{n} r_{ij} q_{ij}$, where "$\circ$" is the sum of elements of the entry-wise product of two matrices.

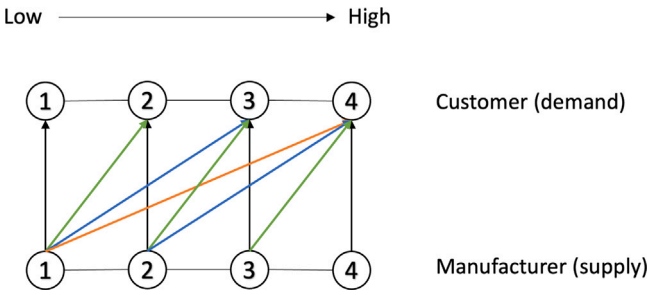


Fig. 1. Material (product) upgrade application of Horizontal differentiation.

3.1. MDP framework

In this section, we first formulate our problem as a standard MDP. Let p_i be the probability distribution of demand type i. The total demand of type i in the next period is the sum of the outstanding demand at the current period x_i and the demand that occurred in the current period d_i. Subtracting the amount of type i demand filled by matching $\bar{q}_i = \sum_j q_{ij}$, we can obtain the outstanding demand in the next period denoted by x'_i given by,

$$x'_i = x_i + d_i - \bar{q}_i. \tag{1}$$

We have illustrated our dynamic resource matching problem in the MDP framework in Fig. 2. Moreover, the matching quantity is constrained by the current state, that is, the demand vector, as well as the capacity vector,

$$\sum_{j=1}^{n} q_{ij} \le x_i, \quad \forall i = 1, \dots, m \tag{2}$$

$$\sum_{i=1}^{m} q_{ij} \le c_j, \quad \forall j = 1, \dots, n \tag{3}$$

The demand type transition probability p_t is given as follows:

$$p_t(x'_i|x_i, \bar{q}_i) = \begin{cases} 0 & \text{if } x'_i < (x_i - \bar{q}_i) \\ p_{x'_i-(x_i-\bar{q}_i)} & \text{if } x'_i \ge (x_i - \bar{q}_i) \end{cases} \tag{4}$$

Our goal is to determine a matching policy $\mathbf{Q}^* = (q^*_{ij})$ that maximizes the expected total discounted reward. Let $\mathbf{x}$ be the current state vector in period t and $V_t(\mathbf{x})$ be the optimal expected total discounted reward. The Bellman equation that can be used to calculate the total expected discounted reward recursively is given below:

$$V_t(\mathbf{x}) = \max_{\mathbf{Q}} [\mathbf{R} \circ \mathbf{Q} + \gamma \sum_{\mathbf{x}'} p_t(\mathbf{x}'|\mathbf{x}, \mathbf{Q}) V_{t+1}(\mathbf{x}')], \tag{5}$$

where γ is the discount factor. The matching quantities cannot exceed the demand or supply levels of the different types. At the end of the horizon, all unmatched demand and supply leave the system and therefore, the boundary condition is $V_{T+1}(x) = 0$ for all $x \in \mathbb{R}^m_+$.

The literature on solving this type of matching for a single-period problem is abundant. The above matching problem, however, cannot be solved as a single-period problem which we illustrate with a simple manufacturing example. Consider a 3-period raw-materials matching problem with two demand types and two supply types. Let type 1 be carbon steel and type 2 be stainless steel, that is, a customer who needs carbon steel is demand type 1 and a supplier for the same is supply type 1. It applies similarly to the second demand–supply type. The outstanding demand is $\mathbf{x} = [x_1, x_2] = [8, 7]$, and the available capacity is $\mathbf{y} = [y_1, y_2] = [6, 5]$. The possible demand quantities for both types are $(0, 1, 2, 3, 4, 5, 6, 7, 8)$ with probabilities $(0.2, 0.2, 0.2, 0.2, 0.2, 0, 0, 0, 0)$ for type 1 and $(0.2, 0, 0.2, 0.2, 0.2, 0, 0, 0, 0.2)$ for type 2 respectively. The discount factor, γ is 0.9 and the reward matrix $\mathbf{R}$ is chosen such that $r_{11} > r_{22} > r_{12} > r_{21}$ and is given by $\mathbf{R} = [10 \quad 7 \quad ; \quad 5 \quad 8]$.

We used CPLEX to solve for the single-period solution and used value iteration as the dynamic programming solution. The matching matrices for both solutions are specified below:

$$\text{Single-period problem: } \mathbf{Q}_{\mathbf{CPLEX}} = [6 \quad 0 \quad ; \quad 0 \quad 5]$$

$$\text{Three-period problem: } \mathbf{Q}_{\mathbf{MDP},t_1} = [4 \quad 0 \quad ; \quad 2 \quad 5],$$

$$\mathbf{Q}_{\mathbf{MDP},t_2} = [4 \quad 0 \quad ; \quad 2 \quad 5],$$

$$\mathbf{Q}_{\mathbf{MDP},t_3} = [6 \quad 0 \quad ; \quad 0 \quad 5]$$

Thus, from the above two solutions, we see that the dynamic programming approach accounts for the incoming demand to obtain an optimal solution as compared to the single-period solution which greedily matches the quantities in every period. Moreover, for an MDP to have a myopic optimum, it requires each transition probability to depend on the action taken but not on the state from which the transition occurs (Sobel, 1981). Thus, optimal matching policies cannot be obtained by solving a single-period problem.

3.2. Domain knowledge-informed Q-learning

The standard MDP problem formulated in Eq. (5) requires explicit modeling of the transition dynamics, which can be hard to obtain in practice. Moreover, even if the dynamics are available, standard MDPs suffer from the curse of dimensionality. Conventional value iteration or policy iteration algorithms can thus, solve only small-size problems due to computing limitations. On the other hand, model-free RL algorithms use experiences (e.g., samples) to obtain the value estimate of a state rather than building a model of the environment. The RL algorithms can further leverage powerful function approximation methods to compactly represent value functions, which enables it to deal with large, high-dimensional state and action spaces. In this section, we enhance the existing Q-learning algorithm based on the characteristics of the problem of our interest. Specifically, we first introduce a function that penalizes deviations of the learned policy from some prior policy. We also remove the constraints by penalizing the violation of constraints in the Q value function. The resulting value-penalty function is then implemented in the update step of Q-learning.

The action-value function (Q-function) is a critical modeling element in any RL algorithm. The Q-value for any state $s \in S$ and $a = \pi(s)$ is given by,

$$Q^\pi(s, a) = \mathrm{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots | s_t = s, a_t = a; \pi] \tag{6}$$

$$= r + \gamma \mathrm{E}[V^\pi(s')|s, a], \tag{7}$$

where s_t, a_t, and r_t are the state, action, and the reward obtained in time step t respectively, π is the policy, and γ is the discount factor.

Q-learning is a model-free, off-policy RL algorithm, where the Q-value is randomly chosen initially for each state–action pair and is then updated iteratively using the following rule

$$Q(s_t, a_t) \longleftarrow (1 - \alpha_t) Q(s_t, a_t) + \alpha_t (r_{t+1} + \gamma \max_a Q(s_{t+1}, a)), \qquad \forall t = 1, 2, \dots, \tag{8}$$

where α_t is the learning rate (or step-size).

As observed in Eq. (8), the Q-learning algorithm updates a state–action pair by maximizing over all those actions possible in the next state. This can lead to the agent learning biased estimates in the initial stages of the algorithm and thus, much of the initial time would be spent on unlearning the biased estimates. To tackle this issue, inspired by the work by Fox et al. (2015), we incorporate a penalty on the learned policy $\pi(a|s)$ for $a \in \mathcal{A}$ and $s \in S$ with respect to some prior policy $\mu(a|s)$. The penalty function is defined as,

$$g^\pi(s, a) = h(\pi(a|s), \mu(a|s)) \tag{9}$$

where $h(\cdot, \cdot)$ is any convex function.

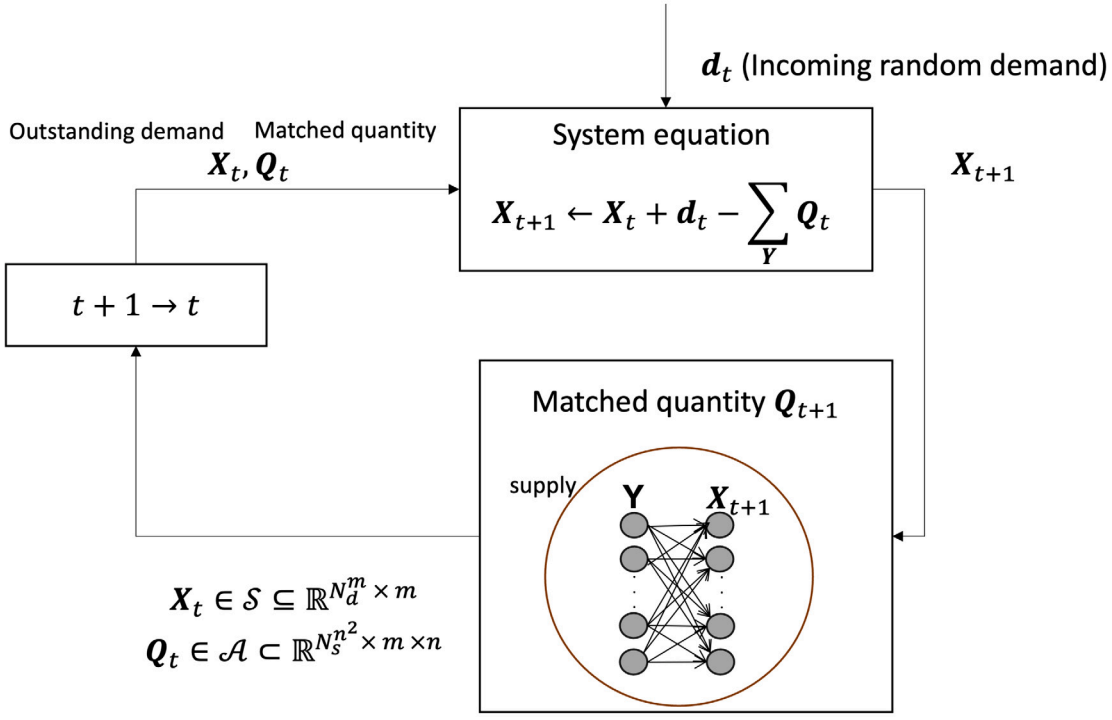


Fig. 2. Dynamic matching problem illustration.

We consider a general convex function as compared to the above-referenced work where the authors specifically use KL-divergence for the penalty function. Other common examples of divergence functions include all types of norms. The penalty in Eq. (9) regularizes the learned policy by penalizing deviations from the prior policy, which is added to the value estimate of the next state. Recall, the value function $V^\pi(s)$ for a given state s with policy π is given by,

$$V^\pi(s) = \sum_{t\geq 0} \gamma^t \mathrm{E}[r_t | s_0 = s] \tag{10}$$

Adding the penalty function to the discounted reward function in the above equation we get,

$$F^\pi(s) = \sum_{t\geq 0} \gamma^t \mathrm{E}\left[\frac{1}{\beta} g^\pi(s_t, a_t) + r_t | s_0 = s\right] \tag{11}$$

We call the above function F^π as the value-penalty function. Here, β is the regularization parameter that sets the weight of the penalty. It controls the effect of the regularization in the update step.

The $V^\pi(s)$ in Eq. (7) can be replaced with the value-penalty function $F^\pi(s)$ defined above to give a new Q-function analogous to Eq. (7),

$$Q^\pi_{DK}(s, a) = r + \gamma \mathrm{E}[F^\pi(s') | s, a], \tag{12}$$

which we call the domain knowledge-informed Q-function. From the above two definitions we obtain

$$F^\pi(s) = \sum_a \pi(a|s)\left[\frac{1}{\beta} g^\pi(s, a) + Q^\pi_{DK}(s, a)\right] \tag{13}$$

The domain knowledge-informed Q-learning update equation is then given by,

$$Q_{DK}(s_t, a_t) \longleftarrow (1 - \alpha_t) Q_{DK}(s_t, a_t) + \alpha_t (r_t + \gamma \max_\pi F^\pi(s_{t+1})). \tag{14}$$

3.2.1. Prior policy μ

Choosing a prior policy is critical since it can either improve convergence or hinder it accordingly. Hence, a prior policy that represents the prior knowledge of the problem has to be carefully chosen. In our problem, we consider a single-period optimal policy as the prior policy, which is obtained by using a standard solver such as CPLEX. The prior policy helps eliminate pathological policies (e.g., actions that match no quantities) that slow down the convergence.

3.2.2. Regularization parameter β

The regularization parameter, β controls divergence from the prior policy. It can either be fixed with respect to time i.e., $\beta_t = \beta$, or scheduled with time, where β_t is a function of time. The penalty function $g^\pi(s, a)$ is modeled in a way that when the value of β increases to a large number, the optimal domain knowledge-informed Q-learning estimate Q^*_{DK} and value-penalty estimate F^* reduce to standard Q-learning estimate Q^* and value function estimate V^*. Also, when β is infinitesimal, the effect of the penalty on the value estimate follows the prior policy μ. Thus, in the initial stages of learning, the prior policy gives an advantage over the greedy Q-value estimate, and in the later stages of learning, the greedy Q-value estimate is a more precise estimate of optimal Q-value, Q^*. Therefore, scheduling β with respect to time would ensure smooth transitioning from Q^μ to Q^π, thereby balancing the benefits of both phases of learning.

3.3. Infeasibility penalty

An infeasible action is one that violates the demand and capacity constraints established in the earlier Section 3.1. This means that the total quantity matched exceeds the respective demand and capacity limits. Hence, it is critical to penalize these actions in order to produce feasible matched quantities. We introduce an infeasibility penalty for violation of each constraint. The penalty associated with violation of the state (outstanding demand) is proportional to the sum of the exceeded matched quantity over all the demand types given by,

$$u(\mathbf{x}, \mathbf{Q}) = k_1 \left(\sum_i^m \mathbb{1}_{\bar{q}_i > x_i} (\bar{q}_i - x_i)\right) \tag{15}$$

where $u(\cdot, \cdot)$ is the demand penalty function whose arguments are demand $\mathbf{x}$ and action (matching quantity) $\mathbf{Q}$, $\bar{q}_i = \sum_{j=1}^n q_{(i,j)}$, and k_1 is the demand penalty proportional constant. Similarly, the penalty associated with violation of the capacity is given by,

$$v(\mathbf{Q}) = k_2 \left(\sum_j^n \mathbb{1}_{\bar{q}_j > c_j} (\bar{q}_j - c_j)\right) \tag{16}$$

where $v(\cdot)$ is the supply penalty function whose argument is action (matching quantity) $\mathbf{Q}$, $\bar{q}_j = \sum_{i=1}^{m} q_{(i,j)}$, and k_2 is the supply penalty proportional constant. We obtain a revised reward formulation by subtracting the penalty functions from the matching reward r_t defined in Section 3.1,

$$r_t = \mathbf{R} \circ \mathbf{Q} - u(\mathbf{x}, \mathbf{Q}) - v(\mathbf{Q}). \tag{17}$$

In the following section, we give theoretical results to prove the convergence of domain knowledge-informed Q-learning algorithm for two cases: fixed β and changing (increasing) β_t. Note, that the matching reward r_t in time t encompasses the net reward after including the infeasibility penalties.

4. Convergence of domain knowledge-informed Q-learning

By introducing a penalty based on a prior policy, we nudge the algorithm away from learning biased estimates, thereby potentially leading to faster convergence than Q-learning. We formalize this intuition here and prove that the optimal Q-value obtained from the domain knowledge-informed Q-learning algorithm converges to the optimal Q value. We develop our proofs based on theoretical results provided by Bertsekas and Tsitsiklis (1996) and Singh et al. (2000). Note, in both the following sections, we denote domain knowledge-informed Q-function as Q instead of Q_{DK} for ease of notation.

4.1. Value-penalty function with fixed β

In this section, we consider a general algorithm based on pseudo-contraction to help prove the convergence of our domain knowledge-informed Q-learning algorithm with fixed weight parameter β.

We first introduce several expressions that are heavily used in the analysis of the convergence behavior of the standard Q-learning algorithm. Starting with some arbitrary estimate $f_0 \in \mathrm{R}^n$, we assume that the ith component $f(i)$ of f is updated according to

$$f_{t+1}(i) = \big(1 - \gamma_t(i)\big) f_t(i) + \gamma_t(i)\big((H f_t)(i) + \omega_t(i)\big), \qquad t = 0, 1, \dots, \tag{18}$$

where states are denoted by $i = 1, 2, \dots, n$, and $\omega_t(i)$ is a random noise term. We denote by $\mathcal{P}_t$ the history of the algorithm until time t, which can be defined as, $\mathcal{P}_t = \{f_0(i), \dots, f_t(i), w_0(i), \dots, w_t(i), \gamma_0(i), \dots, \gamma_t(i)\}$, for $i = \{1, 2, \dots, n\}$, or may include some additional information. We now introduce some assumptions to help prove the following theorem.

Assumption 1 (*Assumption 4.3, Prop. 4.4, Bertsekas et al.*).

(a) The step-sizes $\gamma_t(i)$ are nonnegative and satisfy

$$\sum_{t=0}^{\infty} \gamma_t(i) = \infty, \qquad \sum_{t=0}^{\infty} \gamma_t^2(i) < \infty$$

(b) For every i and t, we have $\mathrm{E}[\omega_t(i) | \mathcal{F}_t] = 0$.

(c) Given any norm $\|\cdot\|$ on R^n, there exist constants A and B such that

$$\mathrm{E}[\omega_t^2(i) | \mathcal{F}_t] \le A + B\|f_t\|^2, \qquad \forall i, t.$$

(d) The mapping H is a weighted maximum norm pseudo-contraction.

Notice that, since $0 \le \gamma_t(i) < 1$, Assumption 1(a) requires that all state–action pairs be visited infinitely often. Parts (b) and (c) provide assumptions on the noise term. Assumption 1(b) states that $\omega_t(i)$ has zero conditional mean and part (c) provides an upper bound on the conditional variance of the noise term. Part (d) implies that if there exists some $r^* \in \mathbb{R}^n$, a positive vector $\xi = (\xi(1), \dots, \xi(n)) \in \mathbb{R}^n$, and a constant $L \in [0, 1)$, then the function $H : \mathbb{R}^n \mapsto \mathbb{R}^n$ satisfies,

$$\|Hr - r^*\|_\xi \le L\|r - r^*\|_\xi, \qquad \forall r.$$

Based on the above assumptions, the proof of convergence of the sequence generated by iteration (18) has been given by Bertsekas and Tsitsiklis (1996). We state the convergence result for the Q-function with penalties on deviations from a prior policy and infeasibilities (Eq. (14)) under a fixed β.

Theorem 1. *Let Q_t be the sequence generated by the iteration (14). We assume that the step-sizes α_t are non-negative and satisfy*

$$\sum_{t=0}^{\infty} \alpha_t = \infty, \qquad \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

Then Q_t converges to Q^* with probability 1.

Theorem 1 shows that Q-learning with penalties on deviation from a priory policy and infeasibilities still converges, in the limit, to the optimal Q-function, under the same mild conditions as traditional Q-learning. We prove this convergence by verifying that the new Q-function in our problem setting also satisfies the conditions provided in Assumption 1. Note that verifying these conditions is not straightforward and requires rigorous analysis of the properties of the new Q-function. Detailed proof for Theorem 1 is provided in Appendix A.

4.2. Value-penalty function with changing β_t

As explained in Section 3.2.2, scheduling β with respect to time ensures a smooth transition of the learning algorithm from Q^μ to Q^π. In this section, we prove the convergence of the domain knowledge-informed Q-learning for an increasing regularization parameter β_t. Note that the penalty-value function is now dependent on β_t instead of β and is therefore denoted by F_t^π. Moreover, the assumptions and results provided previously involve a time-independent mapping H and therefore, cannot be used to prove the convergence for an increasing regularization parameter β_t. Based on this information, we provide Eq. (14) as an iteration,

$$Q_{t+1}(s_t, a_t) = (1 - \alpha_t) Q_t(s_t, a_t) + \alpha_t (r_t + \gamma \max_{\pi} F_t^\pi(s_{t+1})) \tag{19}$$

We now provide the convergence result for the sequence generated by iteration (19).

Theorem 2. *Let Q_t be the sequence generated by the iteration (19). We assume that the step-sizes α_t are non-negative and satisfy*

$$\sum_{t=0}^{\infty} \alpha_t = \infty, \qquad \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

Then Q_t converges to Q^* with probability 1.

Theorem 2 shows that Q-learning with a scheduled regularization parameter β_t with penalties on deviation from a priory policy and infeasibilities still converges, in the limit, to the optimal Q-function. Detailed proof for Theorem 2 is provided in Appendix B.

5. Domain knowledge-informed deep deterministic policy gradient (DKDDPG) algorithm

While Q-learning is a model-free approach, it suffers from the curse of dimensionality as the classical approaches like dynamic programming. To address this issue, Mnih et al. (2015) implement DQN to approximate Q-values in the Q-learning algorithm using neural networks. In the case of dynamic manufacturing resource matching, the action space is high dimensional, which means DQN may suffer from memory problems since the output layer is the size of the action space.

The DDPG algorithm employs an actor–critic network, which is a temporal difference (TD) version of the policy gradient. This approach is inspired by the recent success of DQN in training an RL agent to learn the optimal action to maximize the total reward of matching demand–supply for each state. The algorithm is simple to

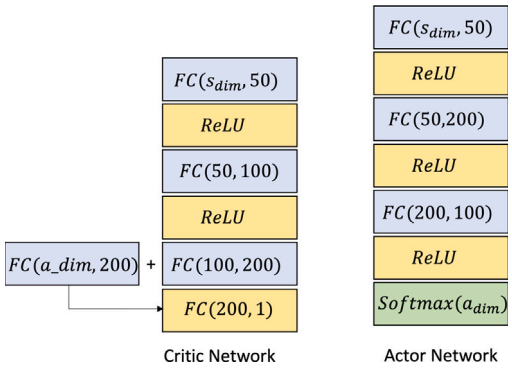


Fig. 3. Neural network architecture of DKDDPG.

implement and scale since it only needs an actor–critic network and a learning algorithm such as Q-learning. Based on DDPG (Lillicrap et al., 2015), we propose domain knowledge-informed DDPG (DKDDPG) which utilizes the domain knowledge-informed Q-learning update equation introduced in Section 3.2, as opposed to DDPG which employs the traditional Q-learning update equation. Our proposed DKDDPG algorithm is presented in Algorithm 1. The details about action exploration, DNN approximator, domain knowledge-informed Q-value function, scheduling of β, action transformation, and the algorithm are discussed below.

Action exploration: In every period, the RL agent matches the demand and supply types, assigning the quantities according to policy π. For action exploration, we employ the $\epsilon-$greedy policy with exponential $\epsilon-$scheduling. During exploration, we inject a normal random noise into the action obtained from the actor network.

DNN: As discussed before, due to the curse of dimensionality, nonlinear functions, and neural network approximators can approximate the Q-values in the Q-learning algorithm. Hence, we employ actor–critic networks based on the success of DQN (Mnih et al., 2015), which solved the issues related to non-stationarity and correlations in the observations by proposing target networks and using experience replay memory. We have implemented two DNNs: an actor network and a critic network. The actor network takes the state as the input and gives an action array as the output. Since the DNN does not provide the actions in a usable format for our problem setting, we perform transformations on the action array which we discuss in detail in the action transformation section. The critic network takes the state and action as input, and provides the optimal Q-values as output, and acts as a Q-function approximator. The DNNs are trained using random mini-batches taken from the experience replay iteratively until all episodes are complete.

We design the architectures of our DNNs similar to the ones implemented by Lillicrap et al. (2015). Considering the state and action spaces, our actor DNN consists of 4 fully connected (FC) layers with shape $[s_{dim}, 50, 200, 100, a_{dim}]$ where s_{dim} is the dimension of the state array and a_{dim} is the flattened-array dimension of the action matrix, along with 3 ReLU activation layers in between and the output FC has softmax activation. We use softmax over tanh for the output layer to obtain a soft matching policy for each demand and supply type. The final layer weights of both the actor and critic were initialized from a uniform distribution $[-0.003, 0.003]$. This is to ensure the initial outputs for the policy and value estimates were near zero. Our critic DNN consists of 3 FC layers with shape $[s_{dim}, 50, 100, 200]$, along with 2 ReLU activation layers in between and a linear activation output. We have provided a simple illustration of our neural network architecture in Fig. 3.

Domain knowledge-informed Q-Value function: We have incorporated the target value of the domain knowledge-informed Q-learning update equation to step 12 of Algorithm 1, where the target prediction is given by,

$$y_i = r_i + \gamma F_i^{\pi^*} = r_i + \gamma \sum_a \pi^*(a|s)\left[\frac{1}{\beta_i} g^{\pi^*}(s,a) + Q'^{\pi^*}(s_{i+1}, \mu(s_{i+1}|\theta^{\mu})|\theta^{Q'})\right]$$

where $\pi^* = \text{argmax}_{\pi} F_i^{\pi}$ and Q' is the target network with weights $\theta^{Q'}$.

Algorithm: Algorithm 1 finds weights θ^Q of the critic DNN network to minimize the Euclidean distance between Q-value $Q(s,a;\theta^Q)$ and y_i. Our approach uses policy gradient to optimize the weights θ^{μ} of the actor DNN network to maximize the Q-value obtained from the critic. The target networks are updated by having them slowly track the learned network as given in the update equation in Algorithm 1. This significantly improves the stability of learning of the networks. The action matrices in each training step of the algorithm are obtained by ϵ-greedy policy.

Scheduling of β: We schedule the relative weight parameter β so that the algorithm penalizes the Q-function and prevents it from choosing deterministic policies initially and then over the length of the episodes, it reduces the penalty, thereby reducing the regularization of the Q-function. We use a linear function to schedule β, i.e. $\beta = kt$, where the hyperparameter k is selected based on a random search from 10 random values followed by a comparison of the total episodic reward over some initial number of episodes.

Algorithm 1 Domain knowledge-Informed DDPG

1: **Input:** Number of demand and supply types, number of time periods T_{max}, total number of episodes, state space
2: Randomly initialize critic network $Q(s,a|\theta^Q)$ and actor $\mu(s|\theta^{\mu})$ with weights θ^Q and θ^{μ}
3: Initialize target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q$ and $\theta^{\mu'} \leftarrow \theta^{\mu}$
4: Initialize replay buffer E
5: **for** episode in total episodes **do**
6: Choose a state s arbitrarily from state space
7: **for** $t = 1 : T_{max}$ **do**
8: Choose action a_t from s_t using ϵ-greedy policy
9: Introduce random demand d and take action a_t, observe reward r_t, next state s_{t+1}
10: Store transition (s_t, a_t, r_t, s_{t+1}) in E
11: Sample a random mini-batch of N transitions (s_i, a_i, r_i, s_{i+1}) from E
12: Set $y_i = r_i + \gamma\left(\sum_a \pi^*(a|s)\left[\frac{1}{\beta_i} g^{\pi^*}(s,a) + Q'^{\pi^*}(s_{i+1}, \mu(s_{i+1}|\theta^{\mu})|\theta^{Q'})\right]\right)$
13: Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$
14: Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^{\mu}} J \approx \frac{1}{N}\sum_i \nabla_a Q(s_i, a_i|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^{\mu}} \mu(s|\theta^{\mu})|_{s_i}$$

15: Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'}$$
$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'}$$

16: **end for**
17: **end for**

Action transformation and infeasibility penalty: When performing action exploitation in the ϵ-greedy policy, we use a softmax activation function in the output layer which outputs a vector of probabilities. We then scale the vector accordingly to obtain the quantities matched

$$\begin{pmatrix}0.5 & 0.2 & 0.2 & 0.1\end{pmatrix} \longrightarrow \begin{pmatrix}0.5 & 0.2\\ 0.2 & 0.1\end{pmatrix} \longrightarrow \begin{pmatrix}0.714 & 0.286\\ 0.667 & 0.333\end{pmatrix}$$
$$\longrightarrow \begin{pmatrix}0.714 & 0.286\\ 0.667 & 0.333\end{pmatrix} * \begin{pmatrix}12\\ 8\end{pmatrix} \longrightarrow \begin{pmatrix}8.57 & 3.43\\ 5.34 & 2.66\end{pmatrix} \longrightarrow \begin{pmatrix}9 & 3\\ 5 & 3\end{pmatrix}$$

Fig. 4. Action transformation for a 2×2 matching example.

between the demand and capacity types. The action vector output from the actor network needs to be normalized and then scaled by performing a row-wise multiplication with the state vector s, to obtain a scaled action matrix. Since the scaled action matrix could be infeasible in terms of exceeding the total capacity quantity, we employ the infeasibility penalty introduced in Section 3.3 into the reward calculation process. This encourages the network to output feasible actions. We demonstrate the above process using a numerical example in our problem setting. Consider a 2×2 matching problem in manufacturing with an outstanding demand vector $x = (12, 8)$. Fig. 4 details the action transformation process, where we transform the vector of probabilities to a 2×2 square matrix. This matrix is then normalized over each row signifying the probability of matching each capacity type to each demand type. Finally, we perform a row-wise scalar multiplication of the outstanding demand vector x to the normalized probability matrix and round it to get the desired action matrix displayed in Fig. 4.

6. Computational study

In this section, we investigate the performance of domain knowled ge-informed DDPG (DKDDPG) with benchmark methods: a solver for LP form of MDP (exact method), Domain Knowledge-informed Q-learning (DKQL), Deep Q-network (DQN), a Deterministic Policy Gradient (DPG) method, and DDPG. We included the tabular version of our modified Q-learning algorithm, DKQL to compare the difference in its performance with its tabular counterpart. Along with the tabular algorithms' comparison, we included two deep RL approaches, namely the DQN and the DPG since the two methods are different in their approach to solving the problem. DQN is a value-based deep RL method that uses a non-linear approximation of the Q-learning update rule as its loss function, and DPG is a policy-based algorithm that learns the near-optimal policy using the principles of policy gradient theorem. To provide an exact solution approach, we have added the linear program formulation and used a solver to obtain the optimal value estimates for small-size problems. Specifically, we first examine the performance of the DKDDPG with that of all the mentioned baselines for small-size problems. For large-size problems that cannot be solved by most of the other methods due to the curse of dimensionality, we compare the performance of the proposed DKDDPG with only the DDPG algorithm.

For all experiments, we consider discrete uniform demands. The lower limits of the demand distributions are set to zero and the upper limits are drawn from a uniform distribution, $U(0, 20)$. The fixed manufacturing capacities are also drawn from this uniform distribution. Since the outstanding demand can go to infinity in theory, we truncate the state space by ignoring states that have little chance of being visited. For simplicity of testing, we assume the number of demand types is the same as that of capacity types (i.e., $m = n$). For each problem size, we consider five problem instances that have different demand distributions. In terms of performance metrics, for small-size problems, we consider the average value function estimates (for LP) and average learned Q-value estimates (for all others) over all states, convergence time and the number of episodes to converge. For large-size problems, we compare the convergence time, the number of episodes to converge, and the average learned Q-value estimates over all states for both DKDDPG and DDPG. Although conventionally, the average episodic reward is reported for comparison of algorithm performance, the learned Q-value estimates showcase more stable results and consistent trends as compared to the undiscounted cumulative reward values obtained in an episode. Therefore, we report the learned Q-value estimates in our experiments.

For the model-free RL methods, we train the agent for 3000 episodes and truncate every episode after 500 timesteps. We perform a random search for hyperparameter tuning for DDPG and DKDDPG and obtain the following values for the hyperparameters. The replay memory E is equal to one million most recent experiences, the batch size is 64, the actor learning rate is 0.0001, the critic learning rate is 0.0005, and the soft target update parameter, τ is 0.0005. We follow an ϵ-greedy policy for action exploration, where ϵ is annealed exponentially from 1 to 0.1 over the first 300 episodes for small-size cases and over the first 1000 episodes for the large-size experiments and is fixed thereafter respectively. The computing infrastructure used is an Intel Xeon with 4 cores and 8 logical processors. We consider the maximum run-time to be 150000 s for all the algorithms.

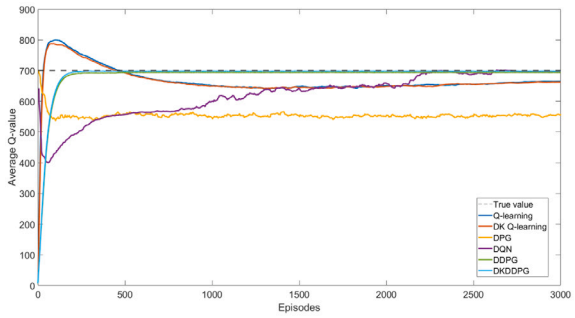
6.1. Performance of DKDDPG in small size problems

We compare the performance of DKDDPG with benchmark algorithms including value iteration, Q-learning, and standard DDPG. We test the performance of the algorithms for m = 2,3,4 and 5. We set the Q-value stopping criterion to be 2%. Along with the metrics mentioned previously, we also record the percentage difference of the maximum Q-value obtained by Q-learning, DDPG, and DKDDPG, with respect to value estimates obtained through Value Iteration.

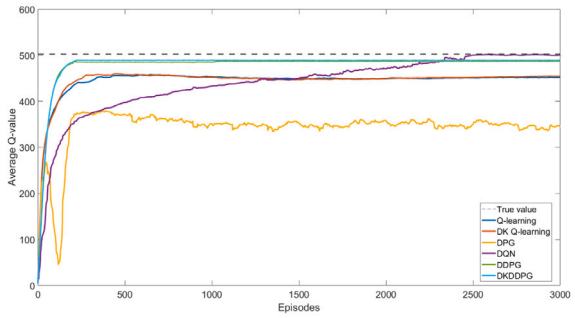
From Table 1, we observe that actor–critic methods such as DPG, DDPG and DKDDPG converge in a much shorter time for all the cases as compared to tabular methods such as Q-learning and DK Q-learning, as well as value-based DRL such as DQN. As the size of the cases increases, the time taken to converge by exact methods such as LP and tabular methods such as Q-learning and DK Q-learning increases significantly since they suffer from the curse of dimensionality. For the 4×4 and 5×5 cases, both LP and the tabular RL methods suffer from memory issues and thus, fail to solve the problem in the designated maximum run times or episodes. Moreover, DQN while able to converge quite close to the true Q-values, suffers from similar issues for 4×4 and 5×5 cases due to the large action spaces. On the contrary, DDPG is able to obtain the solutions for all the cases in an average of 2507 s and 304 episodes, while DKDDPG is able to solve the matching problem for all the cases taking an average of 2300 s and 207 episodes. While DPG also converges at similar times to DDPG and DKDDPG, it obtains much worse values due to correlation and stability issues. Thus, DKDDPG converges approximately 8% faster than DDPG and takes much fewer episodes to converge. Moreover, Q-values generated by the DKDDPG algorithms converge very close to the values of the LPs as can be seen from the error percentages, implying that by leveraging domain knowledge, the DKDDPG algorithm leads to satisfactory policies while accelerating the converging process. We have also illustrated the training process for two of the small-size experiments in Fig. 5 to highlight DKDDPG's superior performance for our dynamic matching problem. Among all the methods, only DQN, DDPG, and DKDDPG closely approximate the true value. While DQN generally converges closest to the true value, it typically takes around 2500 episodes to reach near-optimal results. In contrast, DDPG and DKDDPG achieve reasonably accurate solutions in an average of 200–300 episodes, with DKDDPG slightly outpacing DDPG in convergence and yielding slightly better results for smaller cases.

Table 1
Algorithm Performance in Matching Problem for small size experiments.

Case	Instance	Linear Programming		Q-learning		DK Q-learning		DPG		DQN		DDPG		DKDDPG	
(m x n)		Convergence time (s)	Average value estimates	Convergence time (s) / episodes	Average Q-values / % difference	Convergence time (s) / episodes	Average Q-values / % difference	Convergence time (s) / episodes	Average Q-values / % difference	Convergence time (s) / episodes	Average Q-values / % difference	Convergence time (s) / episodes	Average Q-values / % difference	Convergence time (s) / episodes	Average Q-values / % difference
2 x 2	1	27 662	987.4	1304/2986	917.5/7.1	1333/2767	924.4/6.4	1684/1266	971.5/1.6	84431/2526	969.4/1.8	1195/145	948.2/3.9	1568/138	959.0/2.8
	2	12 988	700.3	2046/2888	659.8/5.8	2057/2794	663.7/5.2	1055/802	450.1/35.7	39287/2355	695.4/0.7	1279/146	687.1/1.9	1725/145	679.0/3.0
	3	22 956	798.6	1189/2967	746.8/6.5	1184/2737	751.2/5.9	1569/1189	699.8/12.4	50311/2968	796.9/0.2	1178/148	763.3/4.4	1513/133	749.8/6.1
	4	23 478	890.4	1281/2742	830.1/6.7	1268/2689	832.2/6.5	2063/1546	583.7/34.4	93936/2816	896.8/0.7	1089/142	865.9/2.7	1555/142	863.7/2.9
	5	19 650	701.3	2040/2966	663.1/5.4	2046/2876	664.7/5.2	942/718	556.4/20.6	26437/2669	700.2/0.1	1172/155	673.2/4.0	1568/146	680.9/2.9
3 x 3	1	27 403	504.3	23413/2925	453.7/10.1	24362/2904	459.2/8.9	2552/1918	353.7/29.9	132051/2887	498.3/1.2	1160/150	478.2/5.2	1590/145	480.3/4.7
	2	10 784	492.5	12792/2796	450.7/8.5	12932/2642	454.6/7.7	2151/961	481.0/2.3	93833/2031	473.8/3.8	1252/158	476.9/3.2	1723/154	464.8/5.6
	3	16 883	502.3	22119/2874	451.4/10.1	23216/2824	455.3/9.3	1284/956	349.5/30.4	127107/2720	500.9/0.3	1215/156	474.9/5.4	1621/147	469.5/6.5
	4	55 706	651.2	26229/3000	596.5/8.4	26732/3000	611.2/6.2	790/587	526.3/19.2	137214/3000	646.1/0.8	1896/242	616.0/5.4	1725/154	600.6/7.8
	5	13 462	491.6	12053/2986	449.3/8.6	12765/2733	448.2/8.8	3203/2426	483.0/1.7	94707/2049	475.3/3.3	1678/152	480.4/2.3	2293/291	474.0/3.6
4 x 4	1	NA		NA		NA		4219/2726	657.2/-	NA		2692/333	1475.1/-	2629/233	1506.0/-
	2							2037/1388	397.1/-			3016/378	767.7/-	2459/219	856.2/-
	3							3964/2647	606.7/-			1683/214	1249.0/-	1689/151	1275.3/-
	4							1698/1186	260.8/-			2787/315	772.4/-	2605/227	1085.8/-
	5							3245/2138	171.5/-			2386/273	835.1/-	2486/202	839.5/-
5x5	1	NA		NA		NA		2796/1964	767.8/-	NA		2269/259	1941.2/-	3588/301	2048.7/-
	2							2254/978	844.5/-			8784/1035	1105.4/-	3533/311	1133.0/-
	3							2526/1565	422.4/-			5739/708	393.3/-	3311/298	1633.2/-
	4							2667/1822	705.9/-			4753/600	1362.2/-	4545/403	1384.5/-
	5							977/662	806.8/-			2914/366	932.6/-	2292/205	1056.7/-

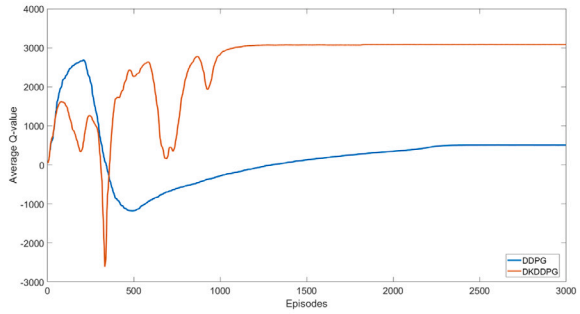


(a) 2x2 Matching Problem

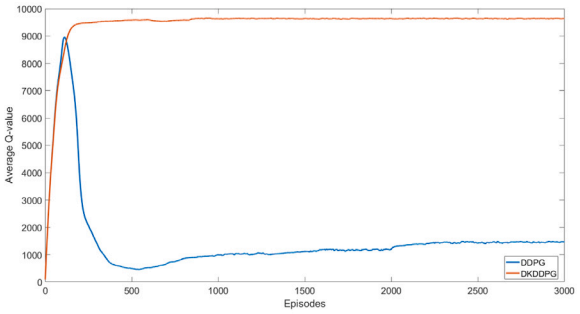


(b) 3x3 Matching problem

Fig. 5. Comparison of different RL approaches for small-size experiments.



(a) 10x10 Matching Problem



(b) 25x25 Matching problem

Fig. 6. Comparison of different RL approaches for large-size experiments.

6.2. Performance of DKDDPG in high-dimensional problems

Next, we compare the performance of the DKDDPG algorithm with the DDPG algorithm for high dimensional state spaces. Specifically, we consider m = 10, 15, 20, 25, and 30. We break this section into two parts: First, we perform the training and report the in-sample performance of DKDDPG and DDPG. Second, we check out-of-sample performance by evaluating the trained models of both algorithms on the manufacturing resource matching problem and report the cumulative reward of both models over 500 timesteps.

6.2.1. In-sample performance of DKDDPG

For the training phase, we set the Q-value stopping criterion to be 3%. We chose the L-2 norm as the penalty function $g^{\pi}(\cdot,\cdot)$. We use the single-period optimal policy discussed in Section 3.2.1 as the prior policy $\mu(a|s)$. Since we consider a linearly scheduled β_t and run the program for ten random values of κ, we record the performance metrics for DKDDPG with the κ value that obtains the highest average learned Q-values after convergence or program completion. Along with the metrics mentioned before, we also calculate the percentage change for the Q-values, convergence time, and the number of episodes to converge for DKDDPG over DDPG.

From Table 2, we observe that the domain knowledge-informed DDPG achieves convergence approximately 30% faster in time than DDPG in all the cases. Moreover, DKDDPG takes approximately 36% lesser number of episodes to solve the matching problem. It is also interesting to note that as the size of the cases increases, the improvement in convergence time and episodes also increases, as observed from the average increment values in time and episode from Table 2. This suggests that for larger problems with a high number of demand and capacity types, DKDDPG is much more efficient as compared to DDPG. This validates our hypothesis of adding a prior policy-based penalty to the Q-learning update rule. Upon observing the average learned Q-values, DKDDPG is able to obtain approximately 345% higher average Q-values as compared to the standard DDPG for all the cases. We note that in some cases, while the DDPG is able to converge sooner than DKDDPG in time, it does so locally and achieves much lower reward values. The accelerated convergence in both time and episode count, along with the increased discounted rewards, indicate that DKDDPG effectively leverages the prior policy-based penalty to mitigate biased estimates and expedite learning. This results in quicker convergence and consistently higher Q-values across all scenarios compared to standard DDPG. Furthermore, in Fig. 6, we illustrate the training process for two large-scale experiments, highlighting DKDDPG's superior performance over DDPG in our dynamic matching problem. In both cases presented, DKDDPG consistently achieves significantly higher average Q-values compared to DDPG, which tends to converge to lower values within our 3000 episode limit.

6.2.2. Out-of-sample performance of DKDDPG

For the evaluation phase, we test our trained models of DKDDPG and DDPG over a single run of the resource matching problem. We run our trained model while simulating the demand distribution over 500 timesteps. Here, we run the models over five instances with different demand distributions for all the demand-capacity cases, that is, m = 10, 15, 20, 25, and 30. These instances are separately generated and are thus, different from the ones generated in the training phase. We record the cumulative reward obtained by both the trained models as well as the percentage change of the cumulative reward of DKDDPG over DDPG.

From Table 3, we observe that the trained model of DKDDPG is able to obtain higher cumulative reward over 500 timesteps as compared to the DDPG model. Specifically, the DKDDPG model achieves an average reward increment of 103% over DDPG amongst the 5 supply–demand cases.

Table 2
DKDDPG vs DDPG comparison for large size experiments during training.

Case	Instance	DDPG			DKDDPG					
($m \times n$)		Average Q-values	Time-to-convergence (s)	Episodes-to-convergence	Average Q-values	Reward Increment/Decrement (%)	Time-to-convergence (s)	Time Increment/Decrement (%)	Episodes-to-convergence	Episodes Increment/Decrement (%)
10 × 10	1	556.2	25 132	2871	2674.3	**380.8**	18 512	**−26.3**	1858	**−35.3**
	2	510.8	12 311	1201	2506.7	**390.7**	14 751	19.8	1099	**−8.5**
	3	411.5	29 712	2156	3072.2	**646.6**	14 414	**−51.5**	1057	**−50.9**
	4	681.3	12 142	1147	3184.5	**367.4**	11 556	**−4.8**	841	**−26.7**
	5	582.3	26 055	1983	3521.4	**504.7**	26 679	2.4	1668	**−15.9**
	Average	548.4	21 070	1872	2991.8	**458.0**	17 182	**−12.1**	1305	**−27.5**
15 × 15	1	886.6	53 512	2227	5189.2	**485.3**	52 043	**−2.7**	1848	**−17.0**
	2	823.9	45 079	2513	6233.4	**656.6**	35 171	**−21.9**	1678	**−33.2**
	3	782.3	16 687	1761	6424.5	**721.2**	4367	**−73.8**	312	**−82.3**
	4	4765.2	32 375	1931	6625.1	**39.0**	31 229	**−3.5**	1575	**−18.4**
	5	976.5	36 994	1771	5243.3	**436.9**	37 934	2.5	1605	**−9.4**
	Average	1646.9	36 929	2041	5943.1	**467.8**	32 149	**−19.8**	1404	**−32.1**
20 × 20	1	22 996.4	8862	598	55 113.2	**139.6**	6026	**−32.0**	329	**−44.9**
	2	9702.2	10 958	647	15 259.6	**57.3**	6424	**−41.4**	352	**−45.6**
	3	4932.1	9083	612	8912.2	**80.7**	8525	**−6.2**	469	**−23.4**
	4	911.7	53 167	1546	7189.3	**688.5**	50 091	**−5.8**	1416	**−8.4**
	5	5488.1	17 405	918	11 016.5	**100.7**	10 789	**−38.0**	542	**−40.9**
	Average	8806.1	19 895	864	19 498.2	**213.4**	16 371	**−24.7**	622	**−32.6**
25 × 25	1	13 276.5	53 347	634	19 693.2	**48.3**	38 001	**−28.7**	544	**−14.2**
	2	4475.4	51 572	631	13 952.6	**211.7**	11 799	**−77.1**	276	**−56.3**
	3	1356.3	58 195	655	9635.9	**610.4**	10 509	**−81.9**	258	**−60.6**
	4	12 324.7	52 751	624	15 097.2	**22.5**	9794	**−81.4**	232	**−62.8**
	5	9546.1	16 227	355	10 828.3	**13.4**	23 543	45.1	415	16.9
	Average	8195.8	46 418	580	13 841.4	**181.3**	18 729	**−44.8**	345	**−35.4**
30 × 30	1	4727.5	55 139	717	13 024.1	**175.5**	16 689	**−69.7**	332	**−53.7**
	2	21 046.2	55 561	720	64 567.3	**206.8**	19 087	**−65.6**	362	**−49.7**
	3	3285.4	48 838	664	19 604.1	**496.7**	13 289	**−72.8**	283	**−57.4**
	4	1751.4	58 465	722	17 785.2	**915.5**	58 252	**−0.36**	692	**−4.2**
	5	5286.5	39 259	583	19 145.3	**262.2**	21 992	**−44.0**	396	**−32.1**
	Average	7219.4	51 452	681	26 825.2	**411.3**	25 862	**−50.5**	413	**−39.4**

Table 3
DKDDPG vs DDPG comparison for large size experiments during evaluation.

Case ($m \times n$)	Instance	DDPG Cumulative Reward	DKDDPG Cumulative Reward	Reward Increment/Decrement (%)
10 × 10	1	57 640	98 790	71.3
	2	17 700	94 229	432.3
	3	20 097	35 020	74.2
	4	16 136	119 297	639.3
	5	65 906	122 830	86.4
	Average	35 496	94 033	260.7
15 × 15	1	27 545	47 625	72.9
	2	37 118	68 237	83.8
	3	28 546	78 356	174.5
	4	13 447	56 321	318.8
	5	53 487	87 612	63.8
	Average	32 029	67 630	142.8
20 × 20	1	28 994	39 554	36.4
	2	36 670	54 933	49.8
	3	30 812	43 090	39.9
	4	47 200	67 478	42.9
	5	41 698	48 933	17.3
	Average	37 075	50 798	37.3
25 × 25	1	379 956	527 663	38.9
	2	843 976	976 238	15.7
	3	538 745	679 832	26.2
	4	306 357	475 368	55.2
	5	481 672	538 135	11.7
	Average	510 141	639 447	29.5
30 × 30	1	685 083	831 725	21.4
	2	721 876	904 137	25.2
	3	377 772	542 611	43.6
	4	432 175	852 467	97.3
	5	483 467	655 734	35.6
	Average	540 075	747 335	44.6

7. Conclusion & future work

In this paper, we considered the problem of dynamically matching the demand-capacity types of resources in manufacturing. We formulated the problem as an MDP where the outstanding demand at the end of a period is considered the state and the quantity of demand and capacity matched is the action matrix. To tackle the issue of biased estimates and infeasible actions, we introduced prior policy-based penalty and infeasibility penalty respectively into the traditional Q-learning algorithm. We considered two cases of the regularization parameter β: constant and scheduling, and further theoretically proved the convergence of our domain knowledge-informed Q-learning algorithm for both β cases. To avoid the curse of dimensionality, we proposed the DKDDPG algorithm which utilizes our modified Q-learning update rule. We investigated the performance of our DKDDPG algorithm with some benchmark RL algorithms for both small and large-size experiments and were able to demonstrate improvement in performance and efficiency over those algorithms.

In this paper, we considered a dynamic resource-matching problem in manufacturing for a matching firm with centralized control. It is worth considering a decentralized, cooperative problem with multiple agents having only local information. While our problem considers a single-period lead time, it will be interesting to model a system with a multi-period lead time to accommodate for delays in production due to numerous reasons. We have not considered any matching costs in our system; including a supply-based cost in our reward framework could aid in minimizing the number of suppliers to fulfill a given demand type. We provided theoretical results for domain knowledge-informed Q-learning which establishes a performance guarantee for small-size problems. Future work will consider deriving convergence results for algorithms with function approximations.

CRediT authorship contribution statement

Saunak Kumar Panda: Conceptualization, Methodology, Writing – original draft, Writing – review & editing. **Yisha Xiang:** Conceptualization, Formal analysis, Investigation, Methodology, Writing – original draft, Writing – review & editing. **Ruiqi Liu:** Methodology, Writing – review & editing.

Acknowledgment

This work is supported in part by the U.S. National Science Foundation under award 2305486.

Appendix A. Proof of Theorem 1

We shall first state the result by Bertsekas and Tsitsiklis (1996) provided as Lemma 1 below.

Lemma 1. *Let f_t be the sequence generated by the iteration (15). Given the conditions in Assumption 1 are satisfied, then f_t converges to f^* with probability 1.*

Notice that at the optimal policy, the optimal domain knowledge-informed Q-function satisfies

$$Q^*(s,a) = r + \gamma \max_{\pi}[F^{\pi}(s')|s,a] \equiv \mathbf{B}^*[Q^*]_{(s,a)}$$

The contraction property of operator $\mathbf{B}^*$ defined above can be proven similarly as done by Fox et al. (2015).

Theorem 1. *Let Q_t be the sequence generated by the iteration (14). We assume that the step-sizes α_t are non-negative and satisfy*

$$\sum_{t=0}^{\infty} \alpha_t = \infty, \qquad \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

Then Q_t converges to Q^* with probability 1.

Proof. We can see that the 1st assumption above is the same as condition (a) stated in Assumption 1. Since $\mathbf{B}^*$ is a contraction mapping, $\mathbf{B}^*$ is automatically a pseudo-contraction, satisfying condition (d) of Assumption 1.

We now verify the assumptions on the noise variable ω_t. Using the definition of $\mathbf{B}^*$, we automatically get

$$\begin{aligned}
\mathrm{E}[\omega_t(r_t, s_{t+1})|\mathcal{F}_t] &= \mathrm{E}\left[-\mathbf{B}^*[Q_t]_{(s_t,a_t)} + r_t + \gamma \mathrm{E}_p[F^\pi(s_{t+1})|s_t, a_t]\middle|\mathcal{F}_t\right] \\
&= -\mathbf{B}^*[Q_t]_{(s_t,a_t)} + \mathrm{E}\left[r_t + \gamma \mathrm{E}_p[F^\pi(s_{t+1})|s_t, a_t]\middle|\mathcal{F}_t\right] \\
&= 0
\end{aligned}$$

Hence, the condition (b) of Assumption 1 is satisfied. Now, we shall verify condition (c).

$$\begin{aligned}
\mathrm{E}[\omega_t^2(r_t, s_{t+1})|\mathcal{F}_t] &= \mathrm{E}\left[\left(-\mathbf{B}^*[Q_t]_{(s_t,a_t)} + r_t + \gamma \mathrm{E}_p[F^\pi(s_{t+1})|s_t, a_t]\right)^2\middle|\mathcal{F}_t\right] \\
&= \mathrm{E}\Big[\left(-\mathbf{B}^*[Q_t]_{(s_t,a_t)}\right)^2 + \left(r_t + \gamma \mathrm{E}_p[F^\pi(s_{t+1})|s_t, a_t]\right)^2 \\
&\quad - 2\mathbf{B}^*[Q_t]_{(s_t,a_t)}\left(r_t + \gamma \mathrm{E}_p[F^\pi(s_{t+1})|s_t, a_t]\right)\Big|\mathcal{F}_t\Big]
\end{aligned}$$

Taking expectation for each term, we get

$$\begin{aligned}
&= \mathrm{E}\left[\left(-\mathbf{B}^*[Q_t]_{(s_t,a_t)}\right)^2\middle|\mathcal{F}_t\right] + \mathrm{E}\left[\left(r_t + \gamma \mathrm{E}_p[F^\pi(s_{t+1})|s_t, a_t]\right)^2\middle|\mathcal{F}_t\right] \\
&\quad - 2\mathrm{E}\left[\left(\mathbf{B}^*[Q_t]_{(s_t,a_t)}\right)\left(r_t + \gamma \mathrm{E}_p[F^\pi(s_{t+1})|s_t, a_t]\right)\middle|\mathcal{F}_t\right] \\
&= \left(-\mathbf{B}^*[Q_t]_{(s_t,a_t)}\right)^2 + \mathrm{E}\left[\left(r_t + \gamma \mathrm{E}_p[F^\pi(s_{t+1})|s_t, a_t]\right)^2\middle|\mathcal{F}_t\right] \\
&\quad - 2\mathrm{E}\left[\left(\mathbf{B}^*[Q_t]_{(s_t,a_t)}\right)\middle|\mathcal{F}_t\right]\mathrm{E}\left[\left(r_t + \gamma \mathrm{E}_p[F^\pi(s_{t+1})|s_t, a_t]\right)\middle|\mathcal{F}_t\right] \\
&= \left(\mathbf{B}^*[Q_t]_{(s_t,a_t)}\right)^2 + \mathrm{E}\left[\left(r_t + \gamma \mathrm{E}_p[F^\pi(s_{t+1})|s_t, a_t]\right)^2\middle|\mathcal{F}_t\right] \\
&\quad - 2\left(\mathbf{B}^*[Q_t]_{(s_t,a_t)}\right)\left(\mathbf{B}^*[Q_t]_{(s_t,a_t)}\right) \\
&= \mathrm{E}\left[\left(r_t + \gamma \mathrm{E}_p[F^\pi(s_{t+1})|s_t, a_t]\right)^2\middle|\mathcal{F}_t\right] - \left(\mathbf{B}^*[Q_t]_{(s_t,a_t)}\right)^2 \\
&\le A + B\left\|Q_t\right\|^2
\end{aligned}$$

where $A \ge \mathrm{E}\left[\left(r_t + \gamma \mathrm{E}_p[F^\pi(s_{t+1})|s_t, a_t]\right)^2\middle|\mathcal{F}_t\right]$ and $B = -1$ are constants. Since, all the conditions of Assumption 1 are satisfied, hence using Lemma 1 we conclude that Q_t converges to Q^* with probability 1. □

Appendix B. Proof of Theorem 2

We use a result provided by Singh et al. (2000) denoted as Lemma 2 to help prove Theorem 2.

Lemma 2. *Consider a stochastic process $(\alpha_t, \Delta_t, H_t), t \ge 0$, where $\alpha_t, \Delta_t, H_t : X \longrightarrow \mathcal{R}$ satisfy the equations*

$$\Delta_{t+1}(x) = (1 - \alpha_t(x))\Delta_t(x) + \alpha_t(x)H_t(x), \qquad x \in X, \qquad t = 0, 1, 2, \ldots$$

Let P_t be a sequence of increasing σ–fields such that the α_0 and Δ_0 are P_0–measurable and α_t, Δ_t and H_{t-1} are P_t–measurable, $t = 1, 2, \ldots$. Assume that the following hold:

1. *the set X is finite.*
2. *$0 \le \alpha_t(x) \le 1, \sum_t \alpha_t(x) = \infty, \sum_t \alpha_t^2(x) < \infty$ w.p.1.*
3. *$\|E\{H_t(\cdot)|P_t\}\|_W \le \kappa\|\Delta_t\|_W + c_t$, where $\kappa \in [0, 1)$ and c_t converges to zero w.p.1.*
4. *$Var\{H_t(X)|P_t\} \le K(1 + \|\Delta_t\|_W)^2$, where K is some constant.*

Then, Δ_t converges to zero with probability one(w.p.1).

Theorem 2. *Let Q_t be the sequence generated by the iteration (19). We assume that the step-sizes α_t are non-negative and satisfy*

$$\sum_{t=0}^{\infty} \alpha_t = \infty, \qquad \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

Then Q_t converges to Q^* with probability 1.

Proof. Subtracting the optimal Q^* on both sides of Eq. (9), we get

$$\Delta_{t+1}(s_t, a_t) = (1 - \alpha_t)\Delta_t(s_t, a_t) + \alpha_t(r_t + \gamma \max_\pi F_t^\pi(s_{t+1}) - Q^*(s_t, a_t))$$

where, $\Delta_t(s_t, a_t) = Q_t(s_t, a_t) - Q^*(s_t, a_t)$. We shall denote $H_t(s_t, a_t) = r_t + \gamma \max_\pi F_t^\pi(s_{t+1}) - Q^*(s_t, a_t)$. Thus, rewriting it, we get

$$\Delta_{t+1}(s_t, a_t) = (1 - \alpha_t)\Delta_t(s_t, a_t) + \alpha_t H_t(s_t, a_t)$$

Notice that the optimal Q-function for a given β_t satisfies

$$\begin{aligned}
{Q^*}_t(s_t, a_t) &= r + \gamma \max_\pi F_t^\pi(s_{t+1}) && \text{(B.1)} \\
&= r + \gamma \max_\pi \sum_{a_{t+1}} \pi(a_{t+1}|s_{t+1})\left[\frac{1}{\beta_t} g^\pi(s_{t+1}, a_{t+1}) + Q_t^\pi(s_{t+1}, a_{t+1})\right] && \text{(B.2)} \\
&\equiv \mathbf{B}_t^*[Q_t]_{(s_t,a_t)} && \text{(B.3)}
\end{aligned}$$

Now, as the value of β_t increases to a large number, $\frac{1}{\beta_t} g^\pi(s_{t+1}, a_{t+1}) \longrightarrow 0$. So, the optimal Q-function as $\beta_t \longrightarrow \infty$ is given by

$$\begin{aligned}
Q^*(s_t, a_t) &= r + \gamma \max_\pi \sum_{a'} \pi(a_{t+1}|s_{t+1}) Q_t^\pi(s_{t+1}, a_{t+1}) && \text{(B.4)} \\
&\equiv \mathbf{B}^*[Q_t]_{(s_t,a_t)} && \text{(B.5)}
\end{aligned}$$

We verify all the assumptions in Lemma 2 to prove the convergence of domain knowledge-inspired Q-learning with scheduled β_t. Assumptions 1 and 2 of Lemma 2 are trivially satisfied for our algorithm. We need to check Assumptions 3 and 4 in Lemma 2. Let $P_t = \{Q_0(s, a), \ldots, Q_t(s, a), H_0(s, a), \ldots, H_t(s, a)\}$, for $s \in S, a \in \mathcal{A}$. Then,

$$\begin{aligned}
\left\|\mathrm{E}\{H_t(s_t, a_t)|P_t\}\right\|_W &= \left\|\mathrm{E}\{r + \gamma \max_\pi F_t^\pi(s_{t+1}) - Q^*(s_t, a_t)|P_t\}\right\|_W \\
&= \left\|\mathrm{E}\{r + \gamma \max_\pi F_t^\pi(s_{t+1})|P_t\} - \mathrm{E}\{Q^*(s_t, a_t)|P_t\}\right\|_W
\end{aligned}$$

Using Equation (B.1) and (B.3)

$$= \left\|\mathbf{B}_t^*[Q_t]_{(s_t,a_t)} - Q^*(s_t, a_t)\right\|_W$$

Adding and subtracting ${Q^*}_t$,

$$\begin{aligned}
&= \left\|\mathbf{B}_t^*[Q_t]_{(s_t,a_t)} - {Q^*}_t(s_t, a_t) + {Q^*}_t(s_t, a_t) - Q^*(s_t, a_t)\right\|_W \\
&\le \left\|\mathbf{B}_t^*[Q_t]_{(s_t,a_t)} - {Q^*}_t(s_t, a_t)\right\|_W + \left\|{Q^*}_t(s_t, a_t) - Q^*(s_t, a_t)\right\|_W
\end{aligned}$$

Applying the definitions from Eq. (B.5) to the first term,

$$\begin{aligned}
&= \left\|\mathbf{B}_t^*[Q_t]_{(s_t,a_t)} - \mathbf{B}_t^*[Q_t]_{(s_t,a_t)}\right\|_W + \left\|{Q^*}_t(s_t, a_t) - Q^*(s_t, a_t)\right\|_W \\
&= \left\|{Q^*}_t(s_t, a_t) - Q^*(s_t, a_t)\right\|_W
\end{aligned}$$

By definitions (B.3) and (B.5),

$$\begin{aligned}
&|{Q^*}_t(s_t, a_t) - Q^*(s_t, a_t)| = |\mathbf{B}_t^*[Q_t]_{(s_t,a_t)} - \mathbf{B}^*[Q_t]_{(s_t,a_t)}| \\
&= \left|\gamma \max_\pi F_t^\pi(s_{t+1}) - \gamma \max_\pi \sum_{a_{t+1}} \pi(a_{t+1}|s_{t+1}) Q_t^\pi(s_{t+1}, a_{t+1})\right| \\
&\le \gamma \max_\pi \left|F_t^\pi(s_{t+1}) - \sum_{a_{t+1}} \pi(a_{t+1}|s_{t+1}) Q_t^\pi(s_{t+1}, a_{t+1})\right| \\
&\le \gamma \max_\pi \sum_{a_{t+1}} \pi(a_{t+1}|s_{t+1})\left|\frac{1}{\beta_t} g^\pi(s_{t+1}, a_{t+1}) + Q_t^\pi(s_{t+1}, a_{t+1}) - Q_t^\pi(s_{t+1}, a_{t+1})\right| \\
&= \gamma \max_\pi \sum_{a_{t+1}} \pi(a_{t+1}|s_{t+1})\left|\frac{1}{\beta_t} g^\pi(s_{t+1}, a_{t+1})\right|
\end{aligned}$$

Assume $g^{\pi}(s_{t+1}, a_{t+1})$ is bounded by a constant C for all π, s_{t+1}, a_{t+1}. It follows that

$$|Q^*{}_t(s_t, a_t) - Q^*(s_t, a_t)| \leq \frac{\gamma}{\beta_t} C$$

Therefore, we have

$$\begin{aligned}\left\|\Delta_t^*(s_t, a_t)\right\|_W &= \left\|Q^*{}_t(s_t, a_t) - Q^*(s_t, a_t)\right\|_W \\ &\leq \frac{\gamma}{\beta_t} C \longrightarrow 0.\end{aligned}$$

Thus,

$$\left\|\mathrm{E}\{H_t(s_t, a_t)|P_t\}\right\|_W \leq \kappa \left\|\Delta_t(s_t, a_t)\right\|_W + c_t$$

where $c_t = \gamma\left\| - \Delta_t^*(s_t, a_t)\right\|_W + \left\|\Delta_t^*(s_t, a_t)\right\|_W$ which converges to zero w.p.1 and $\kappa = 0$ in the 1st term. Thus, assumption 3 is verified. We shall now verify assumption 4 in Lemma 2.

$$\begin{aligned}&Var\{H_t(s_t, a_t)|P_t\} = \mathrm{E}\{H_t^2(s_t, a_t)|P_t\} - \mathrm{E}\{H_t(s_t, a_t)|P_t\}^2 \\ &= \mathrm{E}\Big\{\big(r + \gamma \max_{\pi} F_t^{\pi}(s_{t+1}) - Q^*(s_t, a_t)\big)^2 |P_t\Big\} \\ &\qquad - \mathrm{E}\big\{r + \gamma \max_{\pi} F_t^{\pi}(s_{t+1}) - Q^*(s_t, a_t)|P_t\big\}^2 \\ &= \mathrm{E}\Big\{\big(r + \gamma \max_{\pi} F_t^{\pi}(s_{t+1})\big)^2 + \big(Q^*(s_t, a_t)\big)^2 \\ &\qquad - 2\big(r + \gamma \max_{\pi} F_t^{\pi}(s_{t+1})\big)Q^*(s_t, a_t)|P_t\Big\} \\ &\qquad - \mathrm{E}\big\{r + \gamma \max_{\pi} F_t^{\pi}(s_{t+1}) - Q^*(s_t, a_t)|P_t\big\}^2 \\ &= \mathrm{E}\Big\{\big(r + \gamma \max_{\pi} F_t^{\pi}(s_{t+1})\big)^2 |P_t\Big\} + \mathrm{E}\Big\{\big(Q^*(s_t, a_t)\big)^2 |P_t\Big\} \\ &\qquad - 2\mathrm{E}\Big\{\big(r + \gamma \max_{\pi} F_t^{\pi}(s_{t+1})\big)Q^*(s_t, a_t)|P_t\Big\} \\ &\qquad\quad - \Big(\mathrm{E}\Big\{\big(r + \gamma \max_{\pi} F_t^{\pi}(s_{t+1})\big)|P_t\Big\} \\ &\qquad\quad - \mathrm{E}\big\{Q^*(s_t, a_t)|P_t\big\}\Big)^2\end{aligned}$$

Applying the definitions (B.3) and (B.5),

$$\begin{aligned}&= \big(\mathbf{B}_t^*[Q_t]_{(s_t,a_t)}\big)^2 + \big(\mathbf{B}^*[Q_t]_{(s_t,a_t)}\big)^2 - 2\mathbf{B}_t^*[Q_t]_{(s_t,a_t)}\mathbf{B}^*[Q_t]_{(s_t,a_t)} \\ &\qquad - allour\big(\mathbf{B}_t^*[Q_t]_{(s_t,a_t)} - \mathbf{B}^*[Q_t]_{(s_t,a_t)}\big)^2\end{aligned}$$

Combining the first 3 terms above,

$$\begin{aligned}&= \big(\mathbf{B}_t^*[Q_t]_{(s_t,a_t)} - \mathbf{B}^*[Q_t]_{(s_t,a_t)}\big)^2 - \big(\mathbf{B}_t^*[Q_t]_{(s_t,a_t)} - \mathbf{B}^*[Q_t]_{(s_t,a_t)}\big)^2 \\ &= K(1 + \|\Delta_t\|_W)^2\end{aligned}$$

where $K = 0$, hence making it zero variance. Thus, assumption 4 is verified. Since all the assumptions are verified, $Q_t \longrightarrow Q^*$ w.p.1 by Lemma 2. □

References

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.

Antoniadis, A., Gouleakis, T., Kleer, P., & Kolev, P. (2020). Secretary and online matching problems with machine learned advice. *Advances in Neural Information Processing Systems*, *33*, 7933–7944.

Ashlagi, I., & Shi, P. (2016). Optimal allocation without money: An engineering approach. *Management Science*, *62*(4), 1078–1097. http://dx.doi.org/10.1287/mnsc.2015.2162.

Baïou, M., & Balinski, M. (2000). The stable admissions polytope. *Mathematical Programming*, *87*, 427–439. http://dx.doi.org/10.1007/s101070050004.

Barrett, T. D., Clements, W. R., Foerster, J. N., & Lvovsky, A. I. (2020). Exploratory combinatorial optimization with reinforcement learning. arXiv:1909.04063.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., & Bengio, S. (2017). Neural combinatorial optimization with reinforcement learning. arXiv:1611.09940.

Bengio, Y., Lodi, A., & Prouvost, A. (2020). Machine learning for combinatorial optimization: a methodological tour d'horizon. arXiv:1811.06128.

Bertsekas, D., & Tsitsiklis, J. N. (1996). *Neuro-dynamic programming*. Athena Scientific.

Bloch, F., & Houy, N. (2009). Optimal assignment of durable objects to successive agents. *Economic Theory*, *51*, http://dx.doi.org/10.1007/s00199-011-0616-8.

Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B., & Song, L. (2018). Learning combinatorial optimization algorithms over graphs. arXiv:1704.01665.

Delarue, A., Anderson, R., & Tjandraatmadja, C. (2020). Reinforcement learning with combinatorial actions: An application to vehicle routing. arXiv:2010.12001.

Dulac-Arnold, G., Evans, R., van Hasselt, H., Sunehag, P., Lillicrap, T., Hunt, J., et al. (2016). Deep reinforcement learning in large discrete action spaces. arXiv:1512.07679.

Fox, R., Pakman, A., & Tishby, N. (2015). Taming the noise in reinforcement learning via soft updates. arXiv preprint arXiv:1512.08562.

Gale, D., & Shapley, L. S. (1962). College admissions and the stability of marriage. *American Mathematical Monthly*, *69*(1), 9–15.

Gao, Z., Gao, Y., Hu, Y., Jiang, Z., & Su, J. (2020). Application of deep q-network in portfolio management. In *2020 5th IEEE international conference on big data analytics* (pp. 268–275). IEEE.

Hu, M., & Zhou, Y. (2022). Dynamic type matching. *Manufacturing & Service Operations Management*, *24*(1), 125–142. http://dx.doi.org/10.1287/msom.2020.0952.

Jang, B., Kim, M., Harerimana, G., & Kim, J. W. (2019). Q-learning algorithms: A comprehensive classification and applications. *IEEE Access*, *7*, 133653–133667. http://dx.doi.org/10.1109/ACCESS.2019.2941229.

Jin, C., Allen-Zhu, Z., Bubeck, S., & Jordan, M. I. (2018). Is Q-learning provably efficient?. arXiv:1807.03765.

Karp, R. M., Vazirani, U. V., & Vazirani, V. V. (1990). An optimal algorithm for on-line bipartite matching. In *Proceedings of the twenty-second annual ACM symposium on theory of computing* (pp. 352–358). New York, NY, USA: Association for Computing Machinery, http://dx.doi.org/10.1145/100216.100262.

Knuth, D. E. (1996). Stable marriage and its relation to other combinatorial problems: An introduction to the mathematical analysis of algorithms.

Kool, W., van Hoof, H., & Welling, M. (2019). Attention, learn to solve routing problems!. arXiv:1803.08475.

Kurino, M. (2014). House allocation with overlapping generations. *American Economic Journal: Microeconomics*, *6*(1), 258–289.

Li, Y. (2017). Deep reinforcement learning: An overview. arXiv preprint arXiv:1701.07274.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., et al. (2015). Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.

Liu, Y., Wang, L., & Wang, X. V. (2018). Cloud manufacturing: latest advancements and future trends. *Procedia Manufacturing*, *25*, 62–73.

Mitropoulos, L., Kortsari, A., & Ayfantopoulou, G. (2021). A systematic literature review of ride-sharing platforms, user factors and barriers. *European Transport Research Review*, *13*, 1–22.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533.

Nazari, M., Oroojlooy, A., Snyder, L. V., & Takáč, M. (2018). Reinforcement learning for solving the vehicle routing problem. arXiv:1802.04240.

Ortega, F., & Wolsey, L. (2003). A branch-and-cut algorithm for the single-commodity, uncapacitated, fixed-charge network flow problem. *Networks*, *41*, http://dx.doi.org/10.1002/net.10068.

Pahwa, D., Dur, U., & Starly, B. (2020). Mechanism design for stable matching with contracts in a dynamic Manufacturing-as-a-Service (MaaS) marketplace. http://dx.doi.org/10.48550/ARXIV.2010.12761, URL https://arxiv.org/abs/2010.12761.

Renna, P., & Argoneto, P. (2011). Capacity sharing in a network of independent factories: A cooperative game theory approach. *Robotics and Computer Integrated Manufacturing*, *27*, 405–417. http://dx.doi.org/10.1016/j.rcim.2010.08.009.

Riesen, K., Fankhauser, S., & Bunke, H. (2007). Speeding up graph edit distance computation with a bipartite heuristic. In *Mining and learning with graphs*.

Roth, A. E. (1984). Stability and polarization of interests in job matching. *Econometrica*, 47–57.

Roth, A. E. (1985). Conflict and coincidence of interest in job matching: Some new results and open questions. *Mathematics of Operations Research*, *10*(3), 379–389, URL http://www.jstor.org/stable/3689635.

Roth, A. E., Sönmez, T., & Ünver, M. U. (2004). Kidney exchange. *The Quarterly Journal of Economics*, *119*(2), 457–488.

Roth, A. E., & Sotomayor, M. (1989). The college admissions problem revisited. *Econometrica*, 559–570.

Sethuraman, J., Teo, C., & Qian, L. (2006). Many-to-one stable matching: Geometry and fairness. *Mathematics of Operations Research*, *31*, 581–596. http://dx.doi.org/10.1287/moor.1060.0207.

Singh, S., Jaakkola, T., Littman, M., & Szepesvári, C. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, *38*, 287–308. http://dx.doi.org/10.1023/A:1007678930559.

Sobel, M. J. (1981). Myopic solutions of Markov decision processes and stochastic games. *Operations Research*, *29*(5), 995–1009.

Sutton, J. (1986). Vertical product differentiation: Some basic themes. *American Economic Review*, *76*(2), 393–398, URL https://ideas.repec.org/a/aea/aecrev/v76y1986i2p393-98.html.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT Press.

Tafreshian, A., Masoud, N., & Yin, Y. (2020). Frontiers in service science: Ride matching for peer-to-peer ride sharing: A review and future directions. *Service Science*, *12*, 40–66. http://dx.doi.org/10.1287/serv.2020.0258.

Ünver, M. U. (2010). Dynamic kidney exchange. *Review of Economic Studies*, *77*(1), 372–414.

Vanvuchelen, N., & Boute, R. N. (2022). The use of continuous action representations to scale deep reinforcement learning: An application to inventory control. *SSRN Electronic Journal*, http://dx.doi.org/10.2139/ssrn.4253600.

Wang, Q., & Tang, C. (2021). Deep reinforcement learning for transportation network combinatorial optimization: A survey. *Knowledge-Based Systems*, *233*, Article 107526.

Wu, M.-C., Hsiung, Y., & Hsu, H.-M. (2005). A tool planning approach considering cycle time constraints and demand uncertainty. *International Journal of Advanced Manufacturing Technology*, *26*, 565–571.

Yang, H., Chen, R., & Kumara, S. (2021). Stable matching of customers and manufacturers for sharing economy of additive manufacturing. *Journal of Manufacturing Systems*, *61*, 1–12. http://dx.doi.org/10.1016/j.jmsy.2021.09.013.