Deep Reinforcement Learning with Coalition Action Selection for Online Combinatorial Resource Allocation with Arbitrary Action Space

Tesfay Zemuy Gebrekidan
University of Southampton
Southampton, United Kingdom
tzg1e19@soton.ac.uk

Sebastian Stein
University of Southampton
Southampton, United Kingdom
ss2@ecs.soton.ac.uk

Timothy J.Norman
University of Southampton
Southampton, United Kingdom
t.j.norman@soton.ac.uk

ABSTRACT

Current DRL algorithms typically assume a fixed number of possible actions and sequentially select one action at a time, making them inefficient for resource allocation problems with arbitrarily large action spaces. Sequential action selection requires updating the state for every action selected, which increases the depth of the decision, the state space, the uncertainty, and the number of executions. This affects the convergence of the algorithm and slows the execution speed. Additionally, current DRL algorithms are not efficient for online resource allocation problems with an arbitrary number of task arrivals per time step because they assume a fixed number of actions. To address these challenges, we propose a novel coalition action selection approach that enables the DRL algorithm to simultaneously select a coalition of an arbitrary number of actions from a set with an arbitrary number of possible actions. By making simultaneous decisions at each time step, coalition action selection avoids the computational cost and large state space caused by the sequential decision that updates the state multiple times. We evaluate the performance and complexity of coalition action selection and sequential action selection approaches using an online combinatorial resource allocation problem. The results demonstrate that the coalition action selection approach retains close performance to the offline optimal for various online traffic demand arrival rates of the online combinatorial resource allocation problem, while the performance of the sequential action selection approach decreases as the size of the problem increases. The experiments also demonstrate that coalition action selection has much lower computational complexity than sequential action selection.

KEYWORDS

Dimensionality Reduction; Arbitrary Action Space; Deep Reinforcement Learning; Coalition Action Selection

ACM Reference Format:
Tesfay Zemuy Gebrekidan, Sebastian Stein, and Timothy J.Norman. 2024. Deep Reinforcement Learning with Coalition Action Selection for Online Combinatorial Resource Allocation with Arbitrary Action Space. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), Auckland, New Zealand, May 6 – 10, 2024*, IFAAMAS, 9 pages.

Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024), N. Alechina, V. Dignum, M. Dastani, J.S. Sichman (eds.), May 6 – 10, 2024, Auckland, New Zealand. © 2024 International Foundation for Autonomous Agents and Multiagent Systems (www.ifaamas.org).

1 INTRODUCTION

Combinatorial optimization (CO) problems involve finding the best possible combination of discrete elements from a given set of feasible options. CO problems can be found in many fields, including resource allocation and routing. A resource allocation problem is considered a CO problem when it involves deciding on a combination of tasks to maximize an objective function. The *tasks* can include computational tasks in task offloading that need to be transferred from low-capacity devices to other devices for faster computation [43], or traffic demands or flows that require bandwidth resources over a communication network [17].

Traditionally, mathematical optimization algorithms have been used to solve CO problems [31]. However, if tasks in a CO problem arrive online and in arbitrary numbers, standard optimization algorithms are not efficient because tasks are not known in advance, i.e., online CO problems require online decisions without knowing future arrivals [33], but mathematical optimization algorithms require complete information a priori. DRL is the state of the art for sequential online decisions in dynamic and uncertain contexts [4, 39] with incomplete information because it plans future decisions by learning from experience and minimizes online computational cost [20].

Current DRL algorithms make sequential decisions not only for each time step but also for each available task in a single time step. This comes with many drawbacks, such as the curse of dimensionality [9], large depth of decision, and increased uncertainty, which led to suboptimal results. One of the biggest advantages of applying DRL for resource allocation problems is minimizing online computational costs because it can be trained offline and executed online [20]. However, the sequential decision of the DRL algorithms on CO problems is still inefficient in terms of computational costs. Furthermore, the curse of dimensionality makes training DRL algorithms with sequential action selection slower and more challenging for large problems [22]. Executing the DRL algorithm sequentially for the tasks also incurs a delay in output. Moreover, DRL for arbitrary action space problems, such as combinatorial problems, is understudied. Because the CO problem includes an arbitrary number of tasks in the online setting, the DRL algorithm must work with an arbitrary action space. While these are not negligible challenges, many existing DRL-based online resource allocation algorithms [1, 5, 30] overlook these challenges assuming that only one task arrives at each time step. A DRL-based online resource allocation algorithm by [17] assumes that no new request arrives until the current flow is completed. However, real online resource allocation problems encounter arbitrary numbers of tasks.

For example, [43] considered a scenario in which a cluster of computational units has to make task-offloading decisions on multiple tasks. They proposed a sequential action selection algorithm in which, at each step, the available tasks are selected sequentially until the resource constraint is exhausted. In other words, there are multiple sequential action selections in a single time step to select more tasks. [13] and [14] considered multiple tasks and proposed a deep learning algorithm with a fixed number of outputs to produce binary outputs of fixed size. Other DRL-based combinatorial optimization and resource allocation algorithms [3, 29] apply the concept of sequence-to-sequence modeling using pointer neural networks [38] and transformer networks [36] of natural language processing (NLP) to select combinatorial actions. Nevertheless, all existing multiple-action selection approaches suffer from either the curse of dimensionality or have fixed outputs. Although sequence-to-sequence modeling can be the right option when the order of the output matters, as in the traveling salesman problem (TSP) [3], they are not important when the output order does not matter. Therefore, learning to produce an ordered sequential output for non-orderly sets leads to computational costs in training and execution.

The dimensionality, execution complexity, and training complexity of DRL algorithms in online combinatorial problems can be minimized by changing the way current DRL algorithms select actions. Instead of selecting a sequence of actions one after the other, using sequential execution, until the constraint is met, a coalition of actions can be selected simultaneously with a parallel execution. Coalition formation is a negotiation procedure that aims to resolve conflicts between entities by forming groups that can achieve mutually beneficial outcomes. Recent work by [26] has studied coalition formation and its application to various multiagent systems. In this work, we applied it to combinatorial action selection, where tasks have to find the best coalition by learning different possible coalitions. We model the selection of coalitions as a single task and single coalition formation problem [10], where a group of tasks must form a single coalition to use the limited resources to maximize the combined long-term utility.

In addition to the sequential action selection approach in the output, the representation of arbitrary-sized and orderless (ASO) data in the input is another challenge in applying DRL to online resource allocation problems. I.e. the information in the input to the DRL algorithm can be arbitrary in size, and its order does not matter. However, DRL algorithms, which use a standard neural network as a function approximator, have a predetermined number of inputs that accept a fixed number of inputs in a specific order [34]. Because each input neuron of the neural network is set to accept and sense specific information from the input at a corresponding index, the DRL algorithm will consider different permutations of the same input as different information. This increases the size of the state space and slows down the training of the DRL algorithm. To benefit from DRL approaches in circumstances with such arbitrary inputs whose order does not matter, special architectural components are required [15].

We can classify the existing techniques for the ASO input into two categories: neural network-based input transformations and stationary input transformations as presented in Section 2.2. Stationary transformations map an input of arbitrary length to a fixed-size vector before it is fed into the policy of the DRL approach. Because they can cause a collision of transformations by mapping two or more sets to the same vector, current stationary ASO transformation strategies are less expressive and can cause ambiguity in the DRL algorithm. On the other hand, neural network-based input transformations are more expressive because the original input is directly fed to the policy. The neural network of the policy is used to learn both the input transformation and the policy. Therefore, there is no ambiguity in the DRL. Hence, we adopt the neural network-based input transformation to deal with ASO inputs.

The main contributions of this work are three-fold:

- We propose the first DRL algorithm that selects an arbitrary number of actions simultaneously for combinatorial decisions. This improves the convergence and execution speed of DRL algorithms by minimizing the state space and depth of decision.
- We adopt the transformer neural network to handle ASO inputs and arbitrary action selection.
- We perform a numerical comparison using an online resource allocation problem to evaluate the convergence and complexity of sequential versus coalition action selection.

In the following, Section 2 reviews related work. The problem description is presented in Section 3. The proposed approaches are described in Section 4 and then evaluated in Section 5. The conclusions are presented in Section 6.

2 RELATED WORK

Here, we review the methods related to our proposed coalition action selection and stationary input transformation.

2.1 Task Offloading

The term *sequential action selection* refers to selecting one action at a time, while the term *coalition action selection* refers to selecting a combination of actions at each time step.

Most existing DRL algorithms employ a sequential action selection approach, in which actions are selected one at a time [2]. The curse of dimensionality makes large Markov decision processes (MDPs) intractable without a guarantee of convergence [9] for such sequential action selection approaches. Gu [9] did an in-depth analysis of existing state aggregation and macro-action approaches to reduce state space in MDPs. Delarue et al. [6] explicitly formulated the action selection problem as a mixed-integer optimization problem and used an optimization solver to find the optimal or near-optimal action for the capacitated vehicle routing problem (CVRP). This combinatorial action selection is not suitable for the online combinatorial resource allocation problem for three reasons. First, it accepts only a predefined number of *nearest_M*, inputs at a time in the CVRP. If the number of cities is greater than *nearest_M*, it considers *nearest_M* by distance. There is no reasonable way to choose a fixed number of traffic demands in our online resource allocation problem because they have a complex feature vector as presented in Section 3. Second, it generates a fixed number of actions, whereas the number of actions to be selected in the online combinatorial resource allocation problem is arbitrary in number, depending on the resource constraint. Third, it is not convenient to maximize long-term rewards with bootstrapping. There exist other approaches with binary decisions in resource allocation [13, 14],

but they work with a fixed size of inputs and outputs. Similarly to the limitation discussed for Delarue et al. [6], they are also not convenient for long-term reward maximization. Another work by Yao et al. [41] has proposed an approach that utilizes reversible actions to modify a current solution for combinatorial optimization problems. These actions involve flipping or swapping vertex labels and are encoded using a graph neural network to represent state-action pairs. They have mentioned that permutation invariance is a drawback of their approach. He et al. [11] proposed a combinatorial action selection approach for recommendation systems. However, the action space is designed to be a fixed set of combinations of tasks to be recommended, but combinatorial resource allocation requires selecting an arbitrary number of actions.

Sequence-to-sequence modeling neural networks, such as the pointer neural network [38] and the transformer neural network [36], have shown significant advances in NLP. They are also applied to combinatorial optimization problems such as CVRP [23]. However, even though the encoder part of these sequence-modeling neural networks is processed in parallel, the decoder part is still sequential and gives the output sequentially.

2.2 Handling ASO Inputs in DRL

We classify existing techniques for handling ASO input as neural network-based input transformations and stationary input transformations. One-hot encoding, bag-of-words [44], zero-padding [18], and set-pooling are examples of stationary input transformations. [19] proposed a multiagent DRL (MADRL)-based routing system using one-hot encoding. For example, if we have a network with seven nodes, the number 3 is encoded as 0010000 and the set {4,6} is encoded as 0001010. Although one-hot encoding is used to avoid dependence on the order in which nodes are enumerated, and when nodes are variable in number, it still has two limitations to fully represent ASO input. First, it is a static representation. It cannot represent nodes when they enter and leave the network dynamically. Second, it has a maximum limit on the number of nodes that can be represented. The numerical values that characterize the tasks are also not represented in the one-hot encoding. Bag-of-words is a technique commonly used in NLP to represent a text by the number of occurrences of words. However, it is not suitable for numeric state representation because it only represents the occurrences of words. Zero-padding techniques assume a fixed length of the input. If an input comes with a smaller size than the assumed value, it is zero-padded to make it the set value. The work in [15] has summarized existing approaches to dealing with variable inputs in autonomous driving. In [7] and [42], they proposed elementwise transformation followed by set-pooling to handle the ASO set. First, each element of the set is fed to a DNN of fixed size. Then they are concatenated using a pooling operation *mean*, *sum*, *product*, and *maximum*. Note that although DNN is deployed, the primary technique of handling the ASO input is aggregation of the outputs of the DNN with set-pooling. Lee et al. [16] extended set-pooling to consider the relationship between the elements of the set. These methods have limitations when used for DRL. First, they are used to generate the same output irrespective of the permutation of the input but cannot help in the action representation to select an element of them. In other words, they cannot be used for coalition action selection because their final output is aggregated to a single output vector using set-pooling. Second, although the methods are permutation-invariant and work for any size, they can often transform different sets to the same output. This makes them ineffective for DRL algorithms because they need to interpret different states differently to avoid ambiguity.

Various neural network-based input transformation techniques are available, such as set neural networks [16, 25, 42], permutation-invariant neural networks (PINN) [34], recurrent neural networks (RNN) [12, 32], graph neural networks (GNN) [27], pointer networks [38], and the attention-based transformer neural network known by its title "Attention is all you need" [36]. Set neural networks use a neural network to transform input sets into output sets and then apply maximum pooling or sum pooling to form the context vector. PINN is designed to recognize elements of an ordered input vector, even when there is noise or some elements are missing. RNN processes inputs sequentially, which makes it prone to vanishing gradients and is not suitable for orderless inputs. GNN is designed to work with graph-structured inputs. It uses iterative message passing and embedding to propagate a representation of the input among the elements of the graph. GNN is not important for the ASO input because the inputs have no graph-structured relationship. Despite this, it can be applied by considering the ASO input as a fully connected graph, but this incurs computational cost. Furthermore, the GNN in the fully connected graph is similar to the transformer [37, 40].

The transformer neural network [36], which is also known as a transformer, is the state-of-the-art attention-based neural network that processes input simultaneously. It uses an attention mechanism at every layer of the encoder and decoder network to learn the dependency between elements of a sequence of input. The encoder simultaneously transforms the input and the decoder produces the output sequentially. The transformer is more popular in NLP, but it is also applied to CO problems [23]. In NLP, the input sequence is embedded in a dictionary. However, since the input is numerical data in CO, the embedding layer is replaced by a feedforward neural network [23]. Transformers are more efficient than GNN in representing ASO input elements (traffic demands in our case) because of the attention mechanism, which allows the traffic demands to compute the global context among themselves in parallel. If the order of the input is important, it can use the positional encoder [36], but we do not need it in our case because the traffic demands are independent. We used the encoder part of the transformer to handle ASO input and output the Q-values in parallel from the hidden states for the coalition action selection.

3 PROBLEM DESCRIPTION

The description of the problem is customized from the multiagent learning (MAL) approach for online distributed resource allocation in a network of computing clusters by Zhang et al. [43]. In MAL, tasks with different resource requirements arrive to be processed in a computing cluster from an external environment or are routed internally from neighboring clusters. The offloading of tasks to neighboring clusters aims to maximize global utility by efficiently using resources to process tasks before their deadline. MAL focused on whether to allocate each task locally or forward it to one of the

neighboring clusters, assuming a limit on the number of tasks that can be transferred due to the limited capacity of the communication links. I.e. the number of tasks that can be offloaded are bound by the preset limit. Our work is complementary to MAL, where MAL decides whether tasks are processed locally or offloaded to their neighboring clusters, and our work uses a DRL algorithm to make a combinatorial decision on which of the tasks use the link given the resource constraint and which of them are deferred to the next time step. We customized the problem description of the work by Zhang et al. [43] to consider the bandwidth constraint of a link connecting two clusters as a constraint rather than setting a limit on the number of tasks that can be offloaded. Therefore, the resource allocation in MAL is a computational resource and the resource allocation in our work is a communication resource known as traffic demand [1] [1]. Traffic demands are requests for bandwidth resources to transfer tasks from a source cluster to a neighbor cluster.

Traffic demands are indicated by the identifier k, which ranges from 1 to the total number of traffic demands. To make the identifiers continuous from 1 to the number of traffic demands, the identifier is updated at every time step because new traffic demands can be generated and others can expire. Traffic demand is described by a feature vector that comprises the utility of allocating traffic demand V_k, the bandwidth demand of traffic demand D_k in Optical Data Units (*ODUK*) as used by [1], the time length the bandwidth is needed for traffic demand L_k in time steps, and the maximum allowable waiting time W_k in time steps before it is allocated. Furthermore, the time step at which a traffic demand is generated, indicated by G_k, is recorded for every traffic demand. A set of traffic demands are denoted by $\{k\}$, and hence the set of their V_k, D_k, L_k, and W_k are denoted by $\{V_k\}$, $\{D_k\}$, $\{L_k\}$, and $\{W_k\}$ respectively. At each time step, the total number of traffic demands generated is between 0 and k_{max}. The communication links have a resource constraint of B (in *ODUK*).

3.1 Formulation of the Problem

The problem is to make an online combinatorial resource allocation decision to maximize long-term utility over T time steps, as shown in the objective function of Equation (1a).

$$\max_{X} \quad \sum_{t=1}^{T} \sum_{k=1}^{|\{k\}|} V_{kt} \cdot X_{kt} \cdot G_{kt} \tag{1a}$$

$$\text{s.t.} \quad \sum_{k=1}^{|\{K\}|} D_k \cdot X_{kt} \cdot G_{kt} \leq B_t \quad \forall t \in T \tag{1b}$$

$$\sum_{t=1}^{T} X_{kt} \cdot G_{kt} \in \{0, L_k\} \quad \forall k \in \{k\} \tag{1c}$$

where $|\{k\}|$ is the number of traffic demands, B_t [2] is the link bandwidth constraint at time step t, G_{kt} binary indicator where $G_{kt} = 1$ if $G_k \leq t \leq G_k + W_k + L_k$ or $G_{kt} = 0$ otherwise, $V_{kt} = \frac{V_k}{L_k}$ is an indicator that utility is for the entire length of L_k, and X_{kt} is the binary decision variable for traffic demands where $X_{kt} = 1$ if the traffic demand k is allocated at time step t and 0 otherwise. Equation (1c) ensures that a traffic demand is accepted for the entire length of L_k or rejected. Note that our work is designed to be complementary to MAL. It assumes clusters have decided which tasks to allocate locally, which tasks must be forwarded to the neighboring clusters, and to which neighboring cluster. Because the task routing in the MAL decides for a one-hop distance at a time, the combinatorial resource allocation of the tasks that need to pass through a given link is decided independently of the decision on other links. Therefore, the objective function is from the perspective of a single link.

[1]Traffic demand is a communication resource required to transfer a task.

[2]The value of B_t can be less than or equal to the value of B because previous traffic demands can continue using the link for L_k steps.

3.2 Deployment of the Models

In MAL, the models are deployed as distributed agents in the clusters. In our case, because every link is shared by two clusters, the agent for each link can be deployed in either of the clusters at both ends of the link that has a higher number of processing units.

4 FORMULATING DRL WITH COALITION ACTION SELECTION

A CO problem can be formally defined as a triplet consisting of a set of CO problem instance $\{I\}$, a CO instance to a solution space mapping function S, and an objective function f that maps the solutions in $S(I)$ to real values. This definition is described by [21] and can be used to model the DRL algorithms with sequential and coalition action selection.

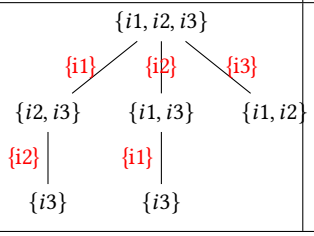
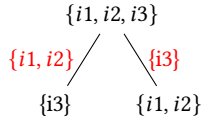
4.1 Modeling the DRL Algorithm with Sequential Action Selection in Combinatorial Optimization

Oren et al. [21] modeled a sequential action selection process for an instance I using an MDP [24] of T time steps. At each time t, the state s_t corresponds to a partial solution and the action $a_t \in A_s$ corresponds to a feasible extension of s_t. A reward $r_{t+1} = r(s_t, a_t) = f(s_{t+1}) - f(s_t)$, transition probability $p(s_{t+1}|s_t, a_t)$, and an action distribution set by a policy $\pi(a_t|s_t)$ are also defined. This leads to a distribution of trajectories $\rho = (s_t, a_t, r_{t+1})$ for $t = 0, \ldots, T-1$, where the transition of the trajectories $p(\rho)$ is calculated as $p(\rho) = p(s_0) \prod_{t=0}^{T-1} \pi(a_t|s_t)\, p(s_{t+1}|s_t, a_t)$. The Q-function is defined as $Q(s_t, a_t) = \mathbb{E}\rho \left[\sum_{i=0}^{T-1} r(s_i, a_i) \middle| s_0 = s_t, a_0 = a_t \right]$. The objective of the DRL agent is to find an optimal policy $\pi^*(a|s) = \text{argmax}_a\, Q(s, a)$.

4.2 Modeling the DRL Algorithm with Coalition Action Selection for Combinatorial and Online Resource Allocation Problems

The description of the coalition action selection problem makes minor changes to the sequential one. Partial solutions are only a subset of those in the sequential formulation. An illustrative example is provided in Table 1. At each time t, the state s_t corresponds to a partial solution. A feasible coalition of actions $\{a_t\} \subseteq A_s$ extends the partial solution to another partial solution. The reward of the coalition is $\bar{r}_{t+1} = r(s_t, \{a_t\}) = f(s_{t+1}) - f(s_t)$, which can be the sum of the rewards of individual actions depending on the objective function. The transition probability $p(s_{t+1}|s_t, \{a_t\})$ and an action distribution set by a policy $\pi(\{a_t\}|s_t)$ are also defined. This leads to a distribution of trajectories $\rho = (s_t, \{a_t\}, r_{t+1})$ for $t = 0, \ldots, T-1$,

Table 1: Illustration of the depth of decision and state space of sequential and coalition action selection for the 0-1 knapsack problem

Sequential Action Selection	Coalition Action Selection
	

where $p(\rho) = p(s_0) \prod_{t=0}^{T-1} \pi(\{a_t\}|s_t)\, p(s_{t+1}|s_t, \{a_t\})$. Then, the Q-function of the DRL algorithm is customized as $Q(s_t, \{a_t\}) = \mathbb{E}\rho \left[\sum_{i=0}^{T-1} \bar{r}(s_i, \{a_i\}) \middle| s_0 = s_t, \{a_0\} = \{a_t\} \right]$. The agent's objective is to find an optimal policy $\pi^*(a|s) = \text{argmax}_a\, Q(s, a)$. Note that the Q-function is computed by the cumulative reward, but the execution and training are run in parallel for the elements of the partial solution to output the Q-value for their best coalition.

4.3 The Depth of Decision and the Size of the State Space of Coalition Action Selection and Sequential Action Selection

DRL-based CO can be illustrated by a decision tree where the root node represents the instance and the child nodes represent the sub-problems after taking an action. An example of the structure of the decision tree is shown in Table 1 with an example of a 0-1 knapsack problem, $I = \{i1, i2, i3\}$ with corresponding weights $\{10, 15, 30\}$, utilities $\{80, 60, 100\}$, and a knapsack capacity of 30. In the trees, the selected elements are represented by red labels at the edges. The leaf nodes represent the terminal states when the knapsack cannot add any more elements. The sequential action selection technique selects only one element at a time. On the contrary, the coalition action selection approach can select any feasible coalition of elements whose sum of weights does not exceed the capacity. In the DRL algorithm, the coalition is formed from the elements with the highest Q-values. The sequential action selection has a depth of 2 and a state space size of 5, while the coalition action selection has a depth of 1 and a state space size of 3.

4.4 The DRL Algorithm with Coalition Action Selection

The online resource allocation problem is a combinatorial optimization problem as shown in Equation (1a). Figure 1 illustrates the interaction between the resource allocation environment and the DRL agent with coalition action selection. At each time step, the DRL algorithm uses a policy π, which is a transformer neural network, that takes a state s containing information about unallocated traffic demands and the resource constraint feature vector B_L [3] for the next L time step as input and outputs corresponding

[3] B_L is the future state of B_t for L time steps, where L is the maximum possible value of L_k, because previous traffic demands occupy the link for L_k time steps.

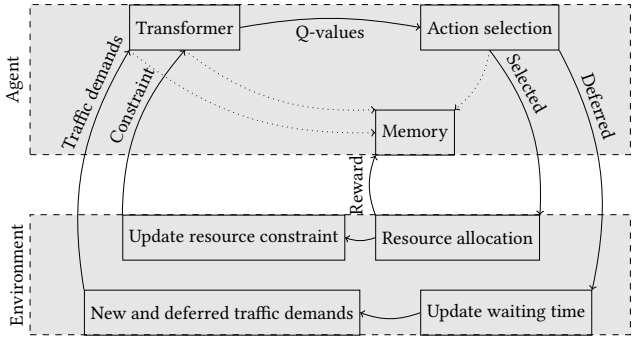


Figure 1: The interaction diagram between the transformer-based DRL agent with coalition action selection and the online combinatorial resource allocation environment

Q-values for the traffic demands in parallel. Then, a coalition of traffic demands is selected based on the order of the Q-values of the traffic demands, considering the resource constraint for the sum of their demands. A reward is computed from the sum of the V_k of the selected traffic demands. B_L is updated after every step because once a traffic demand is allocated, it uses the link until L_k expires. Traffic demands that are not accepted at the current step remain available for decision in later steps as long as their W_k has not expired. The dotted lines indicate that a copy is stored in the replay memory. The state, action, and reward of the DRL algorithm are as follows.

State: The state s includes the feature vectors $[U_k, V_k, D_k, W_k, L_k]$ of the set of traffic demands $\{k\}$ that are waiting to be allocated and the bandwidth constraint vector of the link B_L. Since U_k is used only in the selection of coalition actions, its explanation is available in the description of the action. The state is variable in size because the number of traffic demands can be arbitrarily large. To handle this arbitrary length in the DRL algorithm, we used the encoder part of the transformer neural network to process the input simultaneously because it can handle ASO inputs as explained in Section 2.2.

State Using Transformer: The state transformation is applied by directly entering the set of feature vectors $[U_k, V_k, D_k, W_k, L_k]$ of traffic demands $\{k\}$ and the resource constraint B_L into the encoder as $s = \{B_L, \{k\}\}$. B_L is padded with B to make its feature vector equal to the length of the feature vector of a traffic demand.

Action: The action space $\{a\}$ is the set of traffic demands $\{k\}$. The Q-values of the traffic demands are outputted from the last layer of the encoder-only transformer. Then, the traffic demands with higher demand than the capacity of the link are masked, and the action selection starts for the rest in decreasing order of their Q-value until the resource constraint is exhausted. If a traffic demand has a higher D_L than the resource constraint, the algorithm skips it and checks the next one.

Distinguisher in Coalition Action Selection: DRL outputs the same Q-value for the same input. Therefore, it will be challenging for coalition selection to learn to put the same traffic demands on different coalitions. The U_k, which is 1 by default for all $\{k\}$, is included to distinguish the same traffic demands in the coalition action selection. If two or more traffic demands have the same feature

vector, they are distinguished by indexing them with increasing U_k. The U_k is updated at each time step.

Reward: The set of selected actions receives a joint reward $\bar{r}$, which is the sum of V_k of the selected traffic demands.

Next State: Taking a set of actions $\{a\}$ in state s of the problem transforms the state into a new state s' with reward $\bar{r}$. The selected traffic demands are removed from the set of unallocated demands. The unallocated traffic demands and newly generated traffic demands form the traffic demands of the next state. The resource constraint vector B_L is also updated by subtracting the traffic demands $\{D_k\}$ of the selected actions and also by releasing the occupied resource of traffic demands whose L_k has expired.

Our DRL algorithm with coalition action selection is in Algorithm 1. First, it initializes hyperparameters. Then it runs for E episodes and T steps for every episode. Training is performed at the end of the episode (lines 36 to 42). For the iterations of the steps, it starts by generating initial traffic demands k_{max}, as seen in line 4. After the first step, it generates any number of traffic demands between 0 and k_{max} per step, as seen in line 31. Lines 9 to 14 show exploration and exploitation. Traffic demands are randomly shuffled for exploration. When exploiting, the traffic demands and the resource constraint are fed to the transformer to simultaneously output Q-values for all traffic demands. Then they are sorted so that they are selected accordingly, as seen in lines 18 to 27. Action selection is carried out by iterating on the sorted $\{k\}$ using the index a. In line 19, it checks if the demand D_a of the traffic demand at index a of $\{k\}$ is not greater than the resource constraint. If not, it appends the identifier of the traffic demand to the selected list and adds its utility to the reward, as seen in lines 20 and 21. Then update the resource constraint because traffic demand will use the resource for L_a time steps. This process continues until no more traffic demand can be accepted. The reward is used to update the Q-value of the selected traffic demands in the training. During training, only the selected actions update their Q-values with the target Q-value computed from the shared reward and the best Q-value of the next state, as seen in line 41. Because each action can be explored with different coalitions of actions at different time steps, the DRL algorithm gradually finds the optimal coalition using their Q-values during constrained action selection.

We employ double Q-learning [35], which mitigates overestimation bias, and prioritized experience replay [28], which prioritizes experiences based on their importance to improve training effectiveness. We use the decaying exploration-exploitation probability ϵ which starts at 1 and decays by subtracting $\frac{\epsilon}{5000}$ in every episode.

5 EXPERIMENTAL EVALUATION

In this section, we experimentally evaluate whether coalition action selection, which benefits from reducing the depth of the decision, the state space, and the action space, yields superior performance and lower complexity than sequential selection. We implemented our algorithm for the online resource allocation problem to assess the validity of the hypothesis. First, we introduce the offline optimal.

5.1 Offline Optimal using Integer Programming

To evaluate the performance of the proposed algorithms, we use integer programming (IP) as an offline optimal. We compared the

Algorithm 1 Transformer Neural Network-based DRL Algorithm with Coalition Action Selection

Initialize parameters: primary transformer parameters θ) , target transformer parameters $\theta' = \theta$), discount factor $\gamma = 0.99$, ϵ-greedy $\epsilon = 1$, replay memory $M = []$, minibatch $b = []$, start and maximum episode ($e = 1$, $E = 50,000$), number of time steps $T = 10$

1: **while** $e \leq E$ **do**
2: $\epsilon = \epsilon - \frac{\epsilon}{5000}$
3: Time step $t = 1$
4: Initialize the set of traffic demands $\{k\}$ with k_{max} number of random initial traffic demands
5: Initialize B for L_{max} time steps B_L:$B_l = B$ for $0 \leq l \leq L_{max}$
6: Pad B_L with B to make it same length with the feature vector of k
7: **while** $t \leq T$ **do**
8: Compute state $s = \{\{k\}, B_L\}$
9: **if** rand $\leq \epsilon$ **then**
10: Shuffle $\{k\}$ randomly and form new lists V, D , L, W
11: **else**
12: Get Q-value $Q = Q\,(s \mid \theta)$ in parallel for s
13: Sort $\{k\}$ in descending Q and form new lists V, D , L, W
14: **end if**
15: Reward r = 0
16: Selected = []
17: $a = 0$
18: **while** $a \leq |\{k\}|$ **do**
19: **if** $D_a \leq B_t$ **then**
20: Selected = append(selected,a)
21: r = r + V_a
22: **for** i = 0 to L_a **do**
23: $B_i = B_i - D_a$
24: **end for**
25: **end if**
26: $a = a+ 1$
27: **end while**
28: Exclude the selected traffic demands from $\{k\}$
29: Decrement $\{L_k\}$ and $\{W_k\}$ of the traffic demands
30: Free occupied resources from B_L for all $L_k \leq 0$
31: Generate new traffic demands of size between 0 and k_{max}, and append to $\{k\}$
32: Compute next state s'
33: Store the experience (s, $selected$, r, s') to M
34: Increment t
35: **end while**
36: Sample a minibatch of (s,$selected$,r,s') from M to b
37: Get Q-value $Q' = Q\,(s' \mid \theta')$ for the traffic demands at s'
38: Mask the Q-values of the infeasible ($W_k > B$) traffic demands from Q' and find maximum Q-value $maxQ_i = \max(Q'_i)\ \forall i \in b$
39: Compute target Q-values $y_i = r_i + \gamma maxQ_i\ \forall i \in b$
40: Get current Q-values $currQ = Q\,(s \mid \theta)$ for the traffic demands
41: Update the DQN by minimizing $Loss = \frac{1}{|b|}\sum_{i \in b}\left(y_i - currQ_i\left(s_i\left(selected_i\right)\right)\right)^2$
42: Update the targets: $\theta' \leftarrow \theta$
43: Increment e
44: **end while**

total reward of the coalition action selection and the sequential action selection approaches in each episode as a percentage of the offline optimal. IP is unrealistic for online resource allocation, as it assumes that all information about demand is available upfront. Therefore, we store the traffic demands generated during the time steps of the episode and run the IP at the end of the episode for the objective function shown in Equation (1a).

5.2 Benchmark Selection

Although there exist some techniques that select a fixed number of outputs at a time, as discussed in related work for CVRP [6] and resource allocation [13, 14], they are not suitable for comparison for the mentioned reasons. Therefore, we chose a sequential action selection approach as in [43] and the offline optimal with IP as the lower bound and the upper bound benchmarks, respectively.

5.3 Experimental Setup

Because our work is complementary to MAL as explained in Section 3, we generate a random number of traffic demands in the range of 0 and k_{max} from a uniform distribution at every time step to resemble the traffic demands that arrive at a cluster externally or offloaded from neighboring clusters. First, we run the experiment with a k_{max} value of 10 to analyze convergence and complexity, and then run the experiment for $k_{max} \in \{2, 5, 10, 15, 20\}$, without changing other settings, to evaluate performance with various traffic demand arrival rates. The feature vectors of the traffic demands are generated from a uniform distribution between 1 and V_{max} = 5, L_{max} = 1, and W_{max} = 3, inclusive. Similarly to the work by Almasan et al. [1], we consider three *ODUK* types for the values of $\{D_k\}$ with {*ODU2*, *ODU3*, and *ODU4*}, whose bandwidth requirements are expressed in terms of multiples of *ODU0* signals. Therefore, the values of $\{D_k\}$ are selected as a random choice of {8,32,64} *ODU0* units, and the resource constraint B is 100 (*ODU0*). The time step T is 10.

The hyperparameters of our DRL algorithm are configured as follows. The coalition action selection has five inputs including the distinguisher, and the sequential action selection has four inputs. The numerical inputs are embedded in a single-layer neural network with 8 outputs. This is followed by 6 blocks of feedforward neural network and multihead attention layers. The feedforward neural networks have 32 neurons each. We used a multihead attention value of 8. The final feedforward neural network layer has one output, which will be the Q-value of the corresponding traffic demand at the input. Note that the transformer processes the traffic demands in parallel to compute the Q-values. We used a discount factor of 0.99, a learning rate of 0.001, a replay memory of size 10,000 which stores the transitions in a first-in-first-out order, and a minibatch size of 64. The mentioned number of encoder blocks and their number of neurons are selected because they led to superior convergence after exhaustive trial-and-error experiments with various choices. We ran the experiments for 40 runs. The experiments are implemented with Pytorch. To ensure reproducibility, the experiment environment is initialized with a seed value of 0. The episodes are independent. All experiments are initialized with 50,000 episodes and run for 24 hours.

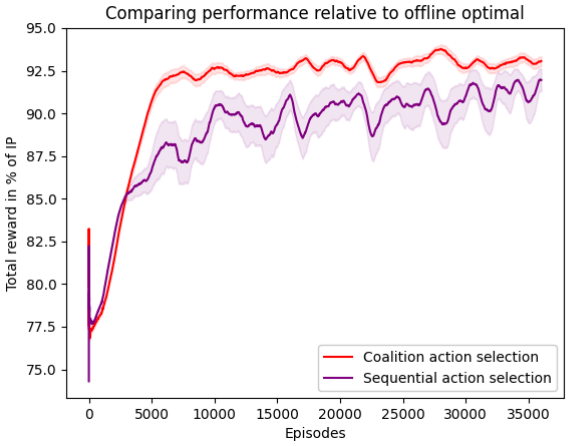


Figure 2: Performance of the coalition action selection and the sequential action selection.

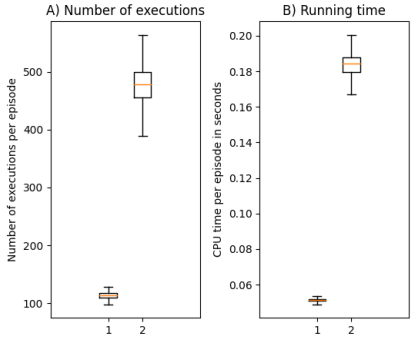


Figure 3: A) Complexity in number of executions for (1) Coalition action selection and (2) Sequential action selection. B) Complexity in CPU time for (1) Coalition action selection and (2) Sequential action selection

5.4 Experimental Comparison

Convergence: First, we evaluate the convergence of the performance of the two algorithms and their complexity. We compared the performance of coalition action selection and sequential action selection as a percentage of offline optimal in each episode. To smooth the curves, the results in Figure 2 are plotted for an averaged moving window of 1000 episodes. Figure 2 shows that the coalition action selection approach converges faster and is superior with an average gap of 2% over sequential action selection. Because the 40 runs end with varying numbers of episodes during the 24-hour training period, we normalize them to the minimum number of episodes to ensure that the 95% confidence interval is computed over an equal number of episodes for Figure 2.

Comparison of Execution Complexity: To compare the computational cost of executing the sequential action selection and coalition action selection approaches, we presented the number of

executions and the CPU time as shown in Figure 3. The box plot is generated from the number of episodes in Figures 2. The number of executions is a count of the number of computing Q-values for the traffic demands. CPU time refers to the sum of the fraction of seconds that the algorithm spent running the DRL algorithm to select actions. The time the algorithm spends training is not considered. As seen in Figure 3 (A), the median number of executions for episodes of the coalition action selection approach is 114 and the number of executions for 50% of the episodes ranges between 110 and 117, while episodes of sequential action selection have a median of 479 executions, and the number of executions of 50% of them ranges between 455 and 499. The sequential action selection performs more executions because it selects one action at a time, resulting in some traffic demands to be executed again in the next decision steps. On the other hand, coalition action selection has a smaller number of executions because it selects multiple actions after a single parallel execution. Figure 3 (B) shows that the CPU time spent executing the DRL in sequential action selection and coalition action selection is proportional to the number of executions in Figure 3 (A). The number of executions and CPU time are averaged over the 40 runs episode-wise before being used for the box plots.

Comparison of Performance and Execution Complexity with Various Problem Sizes: Finally, we repeat the experiment for traffic demand arrival rates of k_{max} = 2, k_{max} = 5, k_{max} = 15, and k_{max} = 20 to compare the algorithms with various sizes of the online combinatorial resource allocation problem. Note that the experiment discussed above is for k_{max} = 10. The results are plotted for the maximum convergence of the best run of 40 runs for a moving window of 5000 episodes, as seen in Figure 4. For example, the maximum convergence for the experiment with k_{max} = 10 shown in Figure 2 with a moving window of 1000 episodes is 93.777% in episode 28036. However, Figure 2 is plotted for the average of 40 runs, while Figure 4 is plotted for the best run of the 40 runs. The reason is that we were unable to compute an average run of 40 runs for k_{max} values of 15 and 20 because some runs failed due to limitations of the academic license of the Gurobi optimizer[4], which is used to compute the IP. Almost all runs failed for k_{max} greater than 20. Note that, with a maximum arrival rate k_{max} of 20, a maximum waiting time of W_{max} of 3 for each traffic demand, and for the time steps per episode T of 10, there are $2^{20*3*10}$ combinations of solutions per episode in the worst case for the Gurobi optimizer.

The coalition action selection approach has shown superior performance, with a convergence of around 95% to offline optimal in the best runs of different problem sizes, but the performance of sequential action selection algorithms decreases, up to 92%, as the size of the problem increases. Note that the two algorithms perform almost the same for very small arrival rates because all traffic demands can be accepted without exhausting the resource constraint. For k_{max} = 10, Figure 2 shows that the coalition action selection converges faster, but Figure 4 shows that they are almost the same because it is plotted only for the maximum convergence.

To compare computational complexity, we present the running time for each corresponding run that led to peak convergence in Figure 4. Unlike Figure 3, which plotted the complexities as box plots in the run episodes, Figure 4 plots the average CPU time in the run episodes because we have to plot a scalar value for every k_{max} on the X-axis. The result shows that the coalition action selection has a lower complexity than the sequential action selection.

[4] https://www.gurobi.com/

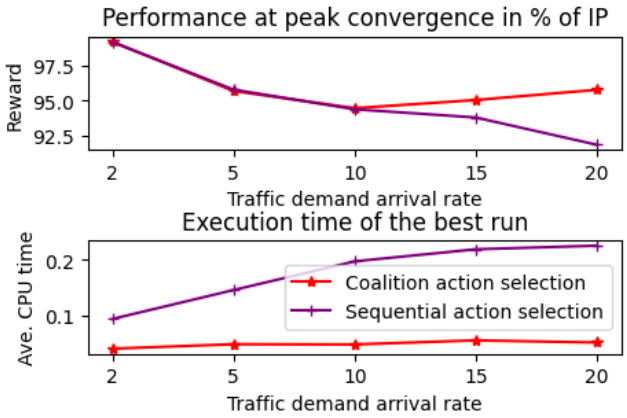


Figure 4: The comparison of the coalition action selection and sequential action selection in terms of performance at peak convergence and average running time, of the episodes of the best run, in CPU time at different numbers of traffic demand arrival rates

6 CONCLUSIONS

We propose a DRL algorithm with coalition action selection for online combinatorial resource allocation with arbitrary action space and experimentally demonstrate that it provides better convergence and lower complexity than sequential action selection. DRL with coalition action provides better performance than sequential action selection because it minimizes the number of iterations, the state space, and the uncertainty of the problem by producing the Q-values for all elements in the input in parallel, unlike sequential action selection, which leads to longer iterations by selecting one action at a time and updating the state for each action selected. The actions of the coalition action selection are coordinated by training with a shared reward. In other recent work, we have applied coalition action selection to a combinatorial client-master MADRL setting [8].

ACKNOWLEDGMENTS

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon. This work was also supported by an EPSRC Turing AI Acceleration Fellowship (EP/V022067/1).

REFERENCES

[1] Paul Almasan, José Suárez-Varela, Krzysztof Rusek, Pere Barlet-Ros, and Albert Cabellos-Aparicio. 2022. Deep reinforcement learning meets graph neural networks: Exploring a routing optimization use case. *Computer Communications* 196 (2022), 184–194.

[2] Tohid Atashbar and Rui Aruhan Shi. 2022. Deep Reinforcement Learning: Emerging Trends in Macroeconomics and Future Prospects. *IMF Working Papers* 2022, 259 (2022).

[3] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. 2016. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940* (2016).

[4] Robert N Boute, Joren Gijsbrechts, Willem Van Jaarsveld, and Nathalie Vanvuchelen. 2022. Deep reinforcement learning for inventory control: A roadmap. *European Journal of Operational Research* 298, 2 (2022), 401–412.

[5] Xiaoliang Chen, Jiannan Guo, Zuqing Zhu, Roberto Proietti, Alberto Castro, and S. J. B. Yoo. 2018. Deep-RMSA: A Deep-Reinforcement-Learning Routing, Modulation and Spectrum Assignment Agent for Elastic Optical Networks. In *2018 Optical Fiber Communications Conference and Exposition (OFC)*.

[6] Arthur Delarue, Ross Anderson, and Christian Tjandraatmadja. 2020. Reinforcement learning with combinatorial actions: An application to vehicle routing. *Advances in Neural Information Processing Systems* 33 (2020), 609–620.

[7] Harrison Edwards and Amos Storkey. 2017. TOWARDS A NEURAL STATISTICIAN. (2017).

[8] Tesfay Zemuy Gebrekidan, Sebastian Stein, and Timothy J.Norman. 2024. Combinatorial Client-Master Multiagent Deep Reinforcement Learning for Task Offloading in Mobile Edge Computing: Extended Abstract. In *Proc. of the 23rd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2024)*. (In press).

[9] Yilan Gu. 2003. Solving Large Markov Decision Processes (depth paper).

[10] Miao Guo, Bin Xin, Jie Chen, and Yipeng Wang. 2020. Multi-agent coalition formation by an efficient genetic algorithm with heuristic initialization and repair strategy. *Swarm and Evolutionary Computation* 55 (2020), 100686.

[11] Ji He, Mari Ostendorf, Xiaodong He, Jianshu Chen, Jianfeng Gao, Lihong Li, and Li Deng. 2016. Deep Reinforcement Learning with a Combinatorial Action Space for Predicting Popular Reddit Threads. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*. 1838–1848.

[12] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9 (1997), 1735–1780.

[13] Liang Huang, Xu Feng, Anqi Feng, Yupin Huang, and Li Ping Qian. 2022. Distributed deep learning-based offloading for mobile edge computing networks. *Mobile networks and applications* 27, 3 (2022), 1123–1130.

[14] Liang Huang, Xu Feng, Luxin Zhang, Liping Qian, and Yuan Wu. 2019. Multi-server multi-user multi-task computation offloading for mobile edge computing networks. *Sensors* 19, 6 (2019), 1446.

[15] Maria Huegle, Gabriel Kalweit, Branka Mirchevska, Moritz Werling, and Joschka Boedecker. 2019. Dynamic input for deep reinforcement learning in autonomous driving. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 7566–7573.

[16] Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. 2019. Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.). PMLR, 3744–3753.

[17] Chenyi Liu, Mingwei Xu, Yuan Yang, and Nan Geng. 2021. DRL-OR: Deep Reinforcement Learning-based Online Routing for Multi-type Service Requirements. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*. 1–10.

[18] Angela Lopez-del Rio, Maria Martin, Alexandre Perera-Lluna, and Rabie Saidi. 2020. Effect of sequence padding on the performance of deep learning models in archaeal protein functional prediction. *Scientific reports* 10, 1 (2020), 1–14.

[19] Dmitry Mukhutdinov, Andrey Filchenkov, Anatoly Shalyto, and Valeriy Vyatkin. 2019. Multi-agent deep learning for simultaneous optimization for time and energy in distributed routing system. *Future Generation Computer Systems* 94 (2019), 587–600.

[20] Rui Nian, Jinfeng Liu, and Biao Huang. 2020. A review On reinforcement learning: Introduction and applications in industrial process control. *Computers & Chemical Engineering* 139 (2020), 106886.

[21] Joel Oren, Chana Ross, Maksym Lefarov, Felix Richter, Ayal Taitler, Zohar Feldman, Dotan Di Castro, and Christian Daniel. 2021. SOLO: search online, learn offline for combinatorial optimization problems. In *Proceedings of the International Symposium on Combinatorial Search*, Vol. 12. 97–105.

[22] Kei Ota, Tomoaki Oiki, Devesh Jha, Toshisada Mariyama, and Daniel Nikovski. 2020. Can increasing input dimensionality improve deep reinforcement learning?. In *International conference on machine learning*. PMLR, 7424–7433.

[23] Bo Peng, Jiahai Wang, and Zizhen Zhang. 2020. A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems. In *Artificial Intelligence Algorithms and Applications: 11th International Symposium, ISICA 2019, Guangzhou, China, November 16–17, 2019, Revised Selected Papers 11*. Springer, 636–650.

[24] Martin L. Puterman. 1994. Markov Decision Processes: Discrete Stochastic Dynamic Programming. In *Wiley Series in Probability and Statistics*.

[25] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. 2017. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 652–660.

[26] Samriddhi Sarkar, Mariana Curado Malta, and Animesh Dutta. 2022. A survey on applications of coalition formation in multi-agent systems. *Concurrency and Computation: Practice and Experience* 34, 11 (2022), e6876.

[27] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. 2009. The Graph Neural Network Model. *IEEE Transactions on Neural Networks* 20, 1 (2009), 61–80.

[28] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2016. Prioritized experience replay. (2016).

[29] Yuxiang Sheng, Huawei Ma, and Wei Xia. 2020. A Pointer Neural Network for the Vehicle Routing Problem with Task Priority and Limited Resources. *Inf. Technol. Control.* 49 (2020), 237–248.

[30] Sebastian Stein, Mateusz Ochal, Ioana-Adriana Moisoiu, Enrico Gerding, Raghu Ganti, Ting He, and Tom La Porta. 2020. Strategyproof reinforcement learning for online resource allocation. (2020), 1296–1304.

[31] Esther M Sundermann, Martin J Lercher, and David Heckmann. 2021. Modeling photosynthetic resource allocation connects physiology with evolutionary environments. *Scientific Reports* 11, 1 (2021), 15979.

[32] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems* 27 (2014).

[33] Xiaoqi Tan, Alberto Leon-Garcia, Yuan Wu, and Danny HK Tsang. 2020. Online combinatorial auctions for resource allocation with supply costs and capacity limits. *IEEE Journal on Selected Areas in Communications* 38, 4 (2020), 655–668.

[34] Yujin Tang and David Ha. 2021. The sensory neuron as a transformer: Permutation-invariant neural networks for reinforcement learning. *Advances in Neural Information Processing Systems* 34 (2021), 22574–22587.

[35] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double q-learning. In *Thirtieth AAAI Conference on Artificial Intelligence*. 2094–2100.

[36] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[37] Petar Veličković. 2023. Everything is connected: Graph neural networks. *Current Opinion in Structural Biology* 79 (2023), 102538.

[38] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. *Advances in neural information processing systems* 28 (2015).

[39] Wenbo Wu, Zhengdong Huang, Jiani Zeng, and Kuan Fan. 2021. A fast decision-making method for process planning with dynamic machining resources via deep reinforcement learning. *Journal of Manufacturing Systems* 58 (2021), 392–411.

[40] Zhanghao Wu, Paras Jain, Matthew Wright, Azalia Mirhoseini, Joseph E Gonzalez, and Ion Stoica. 2021. Representing long-range context for graph neural networks with global attention. *Advances in Neural Information Processing Systems* 34 (2021), 13266–13279.

[41] Fan Yao, Renqin Cai, and Hongning Wang. 2021. Reversible Action Design for Combinatorial Optimization with Reinforcement Learning. *arXiv preprint arXiv:2102.07210* (2021).

[42] Manzil Zaheer, Satwik Kottur, Siamak Ravanbhakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J Smola. 2017. Deep Sets. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 3394–3404.

[43] Chongjie Zhang, Victor R Lesser, Prashant J Shenoy, et al. 2009. A Multi-Agent Learning Approach to Online Distributed Resource Allocation.. In *Ijcai*, Vol. 9. Citeseer, 361–366.

[44] Yin Zhang, Rong Jin, and Zhi-Hua Zhou. 2010. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics* 1, 1-4 (2010), 43–52.