2024 IEEE International Conference on Robotics and Automation (ICRA 2024)
May 13-17, 2024. Yokohama, Japan

Learning Dual-arm Object Rearrangement for Cartesian Robots

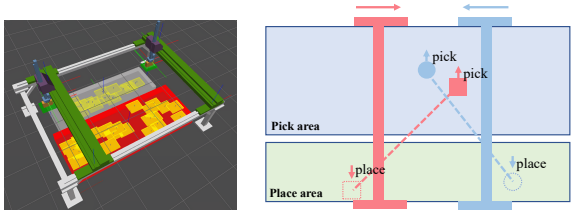
Shishun Zhang[1], Qijin She[1], Wenhao Li[1], Chenyang Zhu[1], Yongjun Wang[1], Ruizhen Hu[3], Kai Xu[1,2,*]
[1]National University of Defense Technology [2]Xiangjiang Laboratory [3]Shenzhen University
[*]Corresponding Author

***Abstract*— This work focuses on the dual-arm object rearrangement problem abstracted from a realistic industrial scenario of Cartesian robots. The goal of this problem is to transfer all the objects from sources to targets with the minimum total completion time. To achieve the goal, the core idea is to develop an effective object-to-arm task assignment strategy for minimizing the cumulative task execution time and maximizing the dual-arm cooperation efficiency. One of the difficulties in the task assignment is the scalability problem. As the number of objects increases, the computation time of traditional offline-search-based methods grows strongly for computational complexity. Encouraged by the adaptability of reinforcement learning (RL) in long-sequence task decisions, we propose an online task assignment decision method based on RL, and the computation time of our method only increases linearly with the number of objects. Further, we design an attention-based network to model the dependencies between the input states during the whole task execution process to help find the most reasonable object-to-arm correspondence in each task assignment round. In the experimental part, we adapt some search-based methods to this specific setting and compare our method with them. Experimental result shows that our approach achieves outperformance over search-based methods in total execution time and computational efficiency, and also verifies the generalization of our method to different numbers of objects. In addition, we show the effectiveness of our method deployed on the real robot in the supplementary video.**

I. Introduction

The object rearrangement problem is commonly encountered in industrial scenarios (product sorting e.g.) and has been studied in the field of task and motion planning (TAMP) [1], [2]. The goal of rearrangement is to transfer every object (mostly using robotic arms) from the source pick position to the target place position with the fewest total completion steps (also known as **makespan**). Utilizing multiple arms can enhance efficiency and accelerate the achievement of the goal, but introduces new higher-level task assignment challenges and lower-level motion planning challenges.

This research primarily addresses the task assignment challenges in the dual-arm object rearrangement while also considering the actual motion planning aspect. To the best of our knowledge, this work is the first one that focuses on object rearrangement using a Cartesian robot (as shown in Figure 1(a)). Compared with high-DOF robots, the movement of the two arms is in a relatively more limited shared space. For example, when there is an interference between the motion trajectories of the two arms (as shown in Figure 1(b)), it cannot be avoided through trajectory optimization, such as one arm circle around another like in normal dual-arm settings, the only way is to let one arm perform the task first and another arm make a concession, which inevitably adds time delays. This limitation requires us to focus more on the higher-level part and aim to reduce the time delays by making a wise task assignment strategy as shown in Figure 2.



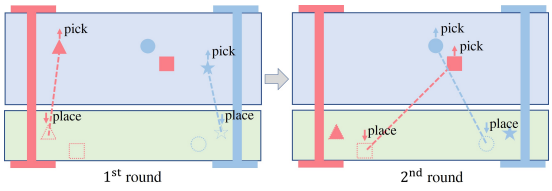
(a) Simulation of the scene (b) Abstraction of the scene

Fig. 1. (a) Two arms in a Cartesian robot are used to transfer the objects (the yellow boards) from the red pick area to the target positions in the white place area. (b) A common motion interference situation: two arms meet, arm r_1 (red) and r_2 (blue) are about to pick up the assigned objects (the same color as the arms). To avoid the collision, r_1 should make a concession for r_2 or vice versa, which inevitably brings time delays.

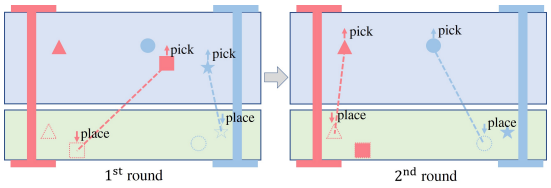
Assigning tasks becomes relatively straightforward when the number of objects is small. However, as the number of objects increases, the complexity of the task assignment problem grows exponentially. This poses challenges in terms of computational efficiency for traditional offline-search-based heuristic methods. Moreover, the potential dynamic changes within the scene, such as the changing positions of objects, require the offline search method to replan the entire task assignment, further adding to the inefficiency.

To address the limitations of traditional methods, we propose to make an online task assignment decision strategy. Taking inspiration from the extensive work on deep learning in online task planning [3]–[7], we recognize the potential of learning-based methods to autonomously discover effective heuristics for optimization tasks [8]. Furthermore, the efficiency demonstrated by reinforcement learning (RL) in long-sequence decision-making motivates us to adopt an online RL framework for making task assignment decisions. Another core insight of this work is utilizing the attention mechanism [9] to discover and model the dependencies between different kinds of input states and find the most likely object-to-arm correspondence under the guide of the dependencies. We have also designed a safe dual-arm motion planning method as the lower-level part, based on it, we adapt some search-based methods to this specific setting as the higher-level task assignment methods and compare our method with them. The experimental results demonstrate the outperformance of our method over other methods in the total completion

979-8-3503-8457-4/24/$31.00 ©2024 IEEE



(a) Unwise assignment: motion interference exists in the 2nd round



(b) Wise assignment: no motion interference in each round

Fig. 2. Comparison of the wise and unwise task assignments. (a) If both arms are assigned the object closest to them in the 1st round, then there will be motion interference in the 2nd round, which will bring time delays caused by concession. (b) In this way, there will be no delay either in the 1st round or in the 2nd round, which will shorten the total completion time.

time (makespan), and verify the acceptable complexity of our method, as it increases linearly with the number of objects.

The contribution of this research can be summarized as 1) A learning-based framework to solve the object rearrangement problems for the motion-limited dual-arm Cartesian robot. 2) An online RL method to make computationally efficient and scalable task assignment strategies for multiple objects. 3) An attention-based neural network to discover dependencies between different states and help achieve better performance in the long-sequence decision-making process.

II. Related Works

Object rearrangement. Numerous efforts have been made to address the issue of object rearrangement. Certain studies concentrate on rearrangement planning when the desired object configuration is provided [10], [11]. Others tackle the challenge of retrieving the target objects and relocating the surrounding objects [12]–[14]. Furthermore, there are investigations into large-scale multiple object rearrangements [15], [16]. Nevertheless, these approaches primarily center around a single robot and do not incorporate multi-robot collaborative planning. In contrast, our research focuses on dual-arm collaboration in the context of object rearrangement.

Multi-agent routing problems. The multi-agent routing or scheduling problem, known for being NP-hard, has been extensively investigated in the field of combinatorial optimization [8], [17], [18]. This problem can also be framed as a multiple traveling salesman problem (mTSP), for which learning-based methods have been widely employed [3]–[5], [19]. However, in many of these studies, the geometric aspects and specific motions of the agents are either overlooked or idealized, despite their significance in real robot scheduling [20]. Similarly, our approach treats the dual-arm rearrangement as a multi-agent routing problem, but unlike previous research, we acknowledge the realistic factors involved, by considering the actual motion of the arm, including factors such as motion interference-induced delays, as illustrated in Figure 1(b).

Task assignment and collaborative planning. Additionally, there are existing works that consider both task assignment and collaborative motion planning [21]–[23]. Many search methods formulate this problem as Mixed-Integer Linear Programs (MILP) or decompose it into sub-problems to mitigate computational complexity. However, as the number of objects increases, the search time escalates significantly. Furthermore, when the number or positions of objects change, a substantial amount of time is required for replanning. In contrast, we propose an online Reinforcement Learning (RL) method that exhibits computation time linearly correlated with the number of objects.

III. Problem Statement

To achieve the objective of transferring all objects with minimal total execution time steps (makespan), our primary focus is on the higher-level task assignment problem, while considering the lower-level dual-arm motion planning. Here, we outline the input and output specifications of the problem:

Input. Suppose that there are n objects $\mathcal{O} = \{o_1, \cdots, o_n\}$, we denote the source pick position and target place position of the object o_i as $pick_i$ and $place_i$ ($i \in [1, n]$). There are two Cartesian robotic arms $\mathcal{R} = \{r_1, r_2\}$, we denote the end effector position of arm r_k as ee_k ($k \in \{1, 2\}$).

Output. 1) Higher-level output: the index i ($i \in [1, n]$) of the object assigned to arm r_k ($k \in \{1, 2\}$) at every task assignment round τ (we use $A^k_\tau = i$ to indicate that). 2) Lower-level output: the collision-free dual-arm motion trajectory when a task is assigned at every τ.

Task assignment. In our real robot system, the control host synchronously publishes the assigned tasks for both arms. Only when both arms have completed their current tasks, the host will publish the new task pair. We denote the set of assignment pairs as $\mathcal{A}$ and the assignment pair at each round τ as $\mathcal{A}_\tau$, we have:

$$\mathcal{A}_\tau = \left\{A^1_\tau, A^2_\tau\right\}_{\tau=1,\cdots,T}. \tag{1}$$

We let $A^1_\tau = i, A^2_\tau = j$ ($i \neq j; i, j \in [1, n]$) represent the index of the object assigned to r_1 and r_2 respectively. For each assignment pair $\mathcal{A}_\tau$, we denote m_τ as the number of time steps required to complete the assignment pair. We need to find the optimal object-to-arm assignment $\mathcal{A}^*$ to minimize the makespan of the whole rearrangement process:

$$\mathcal{A}^* = \underset{\mathcal{A}}{\operatorname{argmin}} \sum_{\tau=1}^{T} m_\tau. \tag{2}$$

Dual-arm motion planning. When the assignment pair $\mathcal{A}_\tau$ is determined, there should be a collision-free dual-arm motion trajectory to complete the transferring process, we denote as $M(\mathcal{A}_\tau)$. Due to the actual discrete motion control, $M(\mathcal{A}_\tau)$ consists of the discrete actions of each arms:

$$M(\mathcal{A}_\tau) = \left\{M^1_\tau, M^2_\tau\right\}_{\tau=1,\cdots,T}. \tag{3}$$

Where M_τ^1 and M_τ^2 are the sets of the discrete steps of arm r_1 and r_2 respectively. As mentioned above, there should be no collision between M_τ^1 and M_τ^2 at every motion step.

IV. Method

We primarily concentrate on addressing the higher-level task assignment problem and present a learning-based framework to achieve it. Additionally, we have developed a heuristic-based approach to execute collision-free dual-arm cooperative motion planning.

A. Higher-level task assignment

Drawing inspiration from previous studies on learning-based approaches for large-scale routing problems [4], [17], [24], and considering the adaptability of online reinforcement learning (RL) in long-sequence decision-making, we model task assignment as a Markov decision process. We then incorporate it into a deep RL framework, treating the two arms as separate agents. We interpret the task assignment process as an active dual-arm task selection process.

MDP formulation. We formulate the online task assignment process as a typical discrete-time Markov decision process, which can be described as a tuple of $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the set of environment states; $\mathcal{A}$ is the action set, which corresponds to the assignment process in our problem; $\mathcal{R}: \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function; $\mathcal{P}: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability function; γ is the discount factor. The policy $\pi: \mathcal{S} \to \mathcal{A}$ is a map from states to probability distributions over actions. We seek for a policy π to maximize the accumulated discounted reward:

$$J(\pi) = E_\pi \left[\sum_{\tau=1}^{T} \gamma^\tau \mathcal{R}\left(s_\tau, a_\tau\right) \right]. \tag{4}$$

Here τ corresponds to the assignment round in our problem, and the action a_τ is the assignment pair $\mathcal{A}_\tau$ that is decided by the policy network at each step.

Observation. The observation consists of three parts: arm states, object states, and a global mask. The arm state $s_k^r = (x_k^r, y_k^r), k \in \{1, 2\}$ contains the dynamic 2-d end-effector coordinates of all arms; The object state $s_i^o = (x_i^s, y_i^s, x_i^t, y_i^t), i \in [1, n]$ contains the source pick and target place position of all objects, and we also update a global mask $M = \{M_1, \cdots, M_n\}$. Each M_i is initially 1, and set to 0 if any object o_i has been transferred, which indicates that the object cannot be assigned in the next round.

Action. At each assignment round τ, based on the current observation (s_τ^r, s_τ^o, M), our network outputs a policy $\pi_\theta\left(\mathcal{A}_\tau\right)$, parameterized by the set of weights θ:

$$\pi_\theta\left(\mathcal{A}_\tau = (i, j) \mid s_\tau^r, s_\tau^o, M\right), i, j \in [1, n]. \tag{5}$$

Where i, j indicate the object index assigned to r_1 and r_2.

Reward Structure. Since our goal is to minimize the makespan (the total completion time) of the whole process (as illustrated in Eq. 2), we follow the most direct and general design, and use the minus of makespan as the sparse reward:

$$R(\pi) = -\sum_{\tau=1}^{T} m_\tau. \tag{6}$$

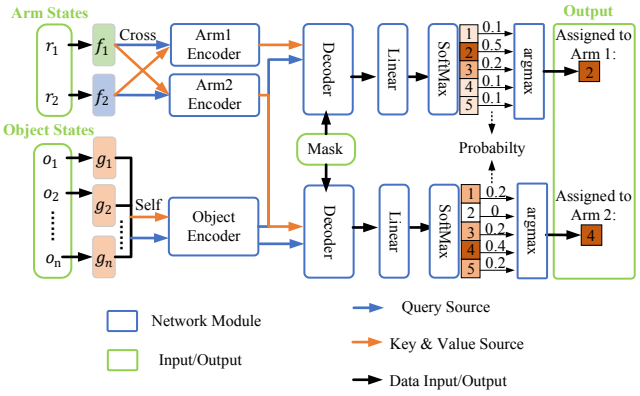


Fig. 3. The structure of the attention-based task assignment network.

Where m_τ is mentioned above as the number of time steps required in one assignment round.

B. Attention-based neural network

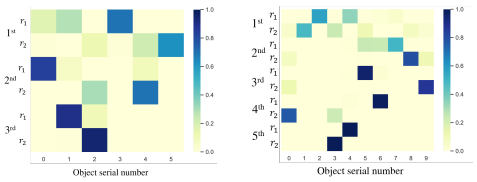
Our main contribution is a centralized attention-based neural network, illustrated in Figure 3, which is shared by both the policy and Q-function networks. The key idea behind our network design is to employ the attention mechanism to identify and capture the relationships between various state information, such as arm-to-arm, object-to-object, and arm-to-object dependencies. These dependencies are then utilized to facilitate the assignment decision-making process. Moreover, the attention mechanism possesses order-invariant and scale-invariant properties, enabling the network to generalize well across different numbers of objects.

Pipeline. The attention-based arm encoders and object encoder receive the arm states and object states as inputs. They model the dependencies among arms and among objects, respectively, and generate arm and object features as outputs. Each arm feature, combined with the shared object feature, forms a pair of (Query, Key-Value) inputs for two attention-based decoder modules. Within each decoder module, the dependency between an arm and all objects is modeled. Guided by the global mask M, attention weights are assigned to each object for the arm. Finally, we use the SoftMax function to output the probability and assign the object with the highest probability to each arm in a greedy manner. The detailed process is further explained in the subsequent section.

Initial state encoder. We use two 2-layer multilayer perceptrons (MLP) to model arm encoding function $\{f_1(\cdot), f_2(\cdot)\}$ and object encoding function $\{g_1(\cdot), g_2(\cdot), \cdots, g_n(\cdot)\}$. Each function takes the state of arms or objects as the input and outputs d-dimensional ($d = 128$ in practice) arm embeddings h_i^r ($i \in \{1, 2\}$) or object embeddings h_j^o ($j \in [1, n]$).

Arm encoder. For each arm, the initial embedding h_i^r is passed through a multi-head attention layer (MHA) as the query source h^q, with the initial embedding of another arm serving as the key-and-value source $h^{k,v}$. By doing so, we achieve the cross-attention of two arm to model dependencies between them. We term the output as $h_{i,j}^{\prime r}$ for each arm.

Object Encoder. Since self-attention achieved good performance to model the message passing of nodes in [25],



(a) object number=6 (b) object number=10

Fig. 4. Visualization of the arm-to-object attention in each assignment round. Each row corresponds to the degree of attention a robot arm paid to different objects in a certain round. (represented as "1st","2nd" round, etc.)

we rely on the same idea, and also use MHA to model the dependencies of all objects in our problem. Specifically, all the n initial object embeddings h_j^o are concated into a $n \times d$ tensor h^O. Then the tensor are served as both query source and key-and-value source of the attention layer, as is commonly done in self-attention mechanisms. We term the output as h'^O.

Decoder. The input of the decoder consists of the common object embedding h'^O (as the query source), an arm embedding $h_i'^r, i \in \{1,2\}$ (as the key-and-value source), and the global mask M. There, we rely on the global mask M to manually set the similarity of each object o_j to each arm r_i as $u_j^i = -\infty, (i \in \{1,2\}, j \in [1,n])$, to ensure that if the object o_j has been rearranged, the attention weight is 0.

$$u_j^i = \begin{cases} \frac{q^T \cdot k_j^i}{\sqrt{d}} & \text{if } M_j = 1 \\ -\infty & \text{otherwise} \end{cases} \tag{7}$$

These similarities are normalized using a Softmax, to finally yield the attention map p of the object o_j for each arm r_i in each round, as shown in Figure 4.

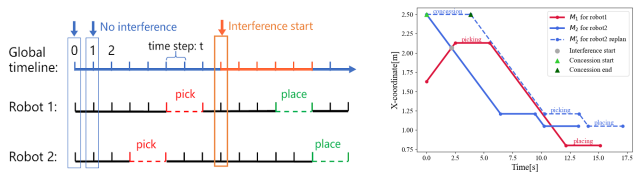
$$p_j^i = \pi_\theta^i \left(A_\tau^i = j \mid h'^O, h_i'^r, M\right) = \frac{e^{u_j^i}}{\sum_{j=1}^n e^{u_j^i}}. \tag{8}$$

Implement details We represent both actor and critic with our weight-sharing network under the Proximal Policy Optimization (PPO; [26]) structure for its stability in training and robustness to the choice of hyperparameters.

C. Lower-level motion planning

The lower-level component is designed to ensure safe dual-arm cooperative motion planning once the assignment pair is determined at each round τ. To accomplish this, we utilize a heuristic-based method to plan a collision-free motion trajectory for both arms before executing the assigned task.

As illustrated in algorithm 1, the lower-level method first plans the initial motion trajectories M_1 and M_2 for r_1 and r_2. Then both trajectories are discretely divided into fixed time interval steps, and the interference is checked at each step, as shown in Figure 5(a); If no interference, both arms will run according to the initial trajectory, and the total time steps will be returned. Otherwise, the interference start time t_{intf_start} will be recorded, then the priority of the two trajectories will be determined to help achieve the minimum concession time. The trajectory with the higher priority will be marked as



(a) Interference detection (b) Motion replanning

Fig. 5. (a) Discretize the trajectories into fixed interval time steps, the solid line indicates going to the pick/place position, while the dashed line indicates picking/placing. The interference is checked at every step. (b) An example of motion replanning when interference exists. The solid lines indicate M_1 and M_2 (minus the safe distance two arms need to keep), and the dashed line indicates M_2', the replanned trajectory for r_2.

Algorithm 1 Lower-level motion planning method

Input: The source end-effector position ee_1^S, ee_2^S; the pick and place position $pick_{1,2}, place_{1,2}$.

Output: The collision-free trajectories M_1, M_2; the total time step T; the final end-effector position ee_1^F, ee_2^F.

```
M_1, M_2 ← InitPlan(ee^S_{1,2}, pick_{1,2}, place_{1,2}) ;
M_1, M_2 ← Discretize(M_1, M_2);
is_interference, t_{intf_start} ← CheckCollision(M_1, M_2);
while is_interference = True do
    M_{1st,2nd}, t_{conc1,conc2} ← PriorityDecide(M_1, M_2);
    M_{2nd} ← Replan(t_{intf_start}, t_{conc1,conc2}, M_1, M_2);
    M_{1st} ← M_{1st}; // no need to replan
    is_interference, t_{intf} ← CheckCollision(M_{1st}, M_{2nd});
end while
T_1 = |M_1|, T_2 = |M_2|;
T = max(T_1, T_2), ee_1^F = M_1(T_1), ee_2^F = M_2(T_2);
return M_1, M_2, T, ee_1^F, ee_2^F
```

M_{1st}, and another as M_{2nd}. Referring to t_{intf_start} and the movement range of M_{1st}, the concession start and end time t_{conc1}, t_{conc2} of M_{2nd} will be calculated, and the trajectory will be replanned as M_{2nd}'. If motion interference still exists between M_{1st} and M_{2nd}' in subsequent steps, then continue the process of check-and-replan, until there is no interference between the two trajectories.

Figure 5(b) shows an example of motion replanning in the x-direction (the direction of the long common rail in Figure 1(a)) of arms that change with time. The figure simply uses straight lines as the acceleration in the real robot is large enough, that the movement process can be approximated as uniform motion at the set speed. It is also worth noting that, in actual control, the potential interference between each arm and surroundings or objects will also be checked.

D. Implementation details

Following the RL training method, in every episode, we randomly sample n objects in the scene as the training data.

Data generation. We manually divide the scene into three parts according to the reachable range of the two arms on the x-axis. Combined with Figure 1(b), we define the two 1/4 size areas on the left and right sides as "exclusive areas" for each arm, and the central 1/2 size areas as "common area" for both arms. If the pick or place position of an object is

located in the exclusive area of an arm, the object can only be operated by this arm. If the pick and place positions are both located in the common area, the object can be operated by both arms. From this, we propose two sampling schemes: 1) **FS**: the pick and place positions of all objects are sampled uniformly in the full space while avoiding the case where the pick and place positions are located in two exclusive areas (to ensure each object is operable by at least one arm) 2) **CA**: all the pick and place positions are sampled only in the narrow common area, which will cause more possibilities for motion interference than **FS**, and brings greater challenges to reducing delay time through task assignment.

Training details. We use both sampling schemes to train our networks. When using the **FS** sampling for training, there could be some "unreachable assignments" since the policy network might assign the object that is only operable by one arm to another arm. To avoid this situation, we add an extra mask to the global mask M and guarantee that the network will not assign unreachable objects to any arm. All networks and learnable parameters are trained with Adam optimizer [27] with a learning rate 10^{-3}, $\beta_1 = 0.9$, $\beta_2 = 0.999$.

V. Experiments

In this section, we evaluate the makespan, delay proportion, and computation time across approaches to demonstrate that our method (i) can help discover efficient long-sequence task assignment strategies that can reduce the overall delay and achieve outperformance in the makespan; (ii) can be generalized to a large number of objects when trained on a relatively small number of objects; (iii) is computationally more efficient than the search-based methods.

A. Experiment setting

We train our networks and run all experiments with an NVIDIA 3080Ti GPU and an i9-10980XE CPU.

Baselines. The performance of our method is compared with three search-based methods and we have also provided two ablations of our method.

1) *Perfect Matching + DP*: We reproduced the method of Shome et al. [22] and adapted it to our problem. The method leverages Perfect Matching [28] and Dynamic Programming (DP) to make offline decisions on task assignments. Based on optimal matching over an undirected graph (transfer graph), the runtime complexity of the method is $O(|\mathcal{E}||\mathcal{V}|\log|\mathcal{V}|) = O\left({}^nP_2 n\log n\right) = O\left(n^3\log n\right)$.

2) *Random Split*: In every round of assignment, we randomly assign objects to the two arms while following the condition of no "unreachable assignment".

3) *Greedy Search*: The "greedy" does not mean simply assigning the object closest to an arm, but assigning the task pair with the least cost currently. In every round of assignment, we calculate the operation steps of every possible assignment pair with the lower-level method and choose the assignment pair with the fewest operation steps for this round. The runtime complexity is $O\left({}^nP_2 n^2\right) = O\left(n^4\right)$.

4) *No_object_encoder*: We use the same architecture as our model, but without the object encoder, and all of the initial object embeddings are just concat together as the query source of two decoders.

5) *No_arm_encoder*: We use the same network model, but without arm encoders, and all of the initial arm embeddings are directly input as the sources to the two decoders.

TABLE I

Results of average makespan on the test set of **CA** sampling (1000 instances each). n denotes the number of objects.

Method	n=4	n=6	n=10	n=14	n=20	n=30
Perfect Matching + DP	45.14	65.76	105.76	145.45	204.22	303.83
Random Split	50.27	75.45	123.97	175.37	246.82	367.29
Greedy Search	45.92	66.78	105.41	143.70	201.06	296.10
No_object_encoder	47.34	67.81	109.45	147.19	212.42	315.02
No_arm_encoder	44.93	64.79	105.32	145.78	205.26	306.97
Ours	**44.37**	**63.13**	**100.57**	**138.69**	**195.79**	**289.26**

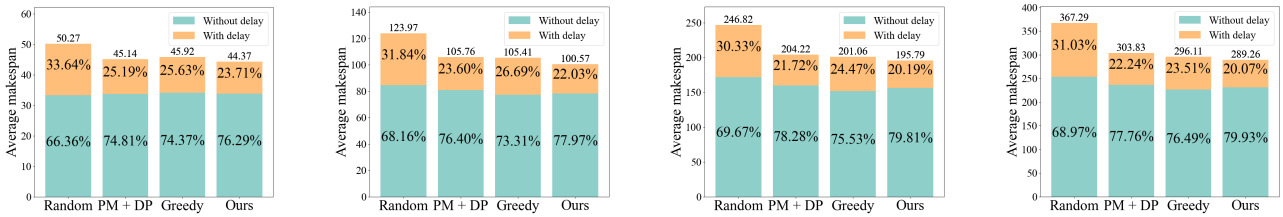
Performance metrics. To evaluate the performance of our method and all baselines, we make two test sets with the sampling scheme of **FS** and **CA**. We sample the object number = 4,6,10,14,20,30 with both schemes and each case with 1000 instances. We train our network, *No_object_encoder* and *No_arm_encoder* with object number = 10, and generalize the trained model to all cases. For heuristic methods, we apply them directly to all cases.

B. Results

In Table I and Table II we report the average Makespan (lower is better) of our method and all baselines on the two test sets we made. In Figure 6 we present the average proportion of delay time within the total makespan for our method and the heuristic approaches, on the **CA** test set, for it sampled only from the narrow center area which significantly increases the probability of motion interference and delay occurring. In Figure 7, we show the comparison of average computation time executing different numbers of objects for *Perfect Matching + DP*, *Greedy Search*, and our method.

From the results of average makespan in Table I and Table II, we first notice that our method can well generalize to all cases despite only trained under the case of n = 10, and it is optimal in most of the cases in the two test sets, except in the case of object number = 20 and 30 in **FS** test set, in which *Greedy Search* performs best. We analyze that, as object numbers increase, *Greedy Search* has advantages over other methods by virtue of a large number of searches and comparisons of assignment pairs in each round. However, this advantage is a trade of computation time, as shown in Figure 7, the search time of which increases dramatically with respect to the scale of the instance, while our method only increases linearly. The computation time of *Perfect Matching + DP* also shows polynomial time growth. From the result, we can say that our method achieves a good balance between computation efficiency and performance.

When we combine the average makespan with the average delay time proportions illustrated in Figure 6, it becomes evident that the key to enhancing efficiency in dual-arm cooperation and ultimately reducing the total makespan lies in minimizing the delay or concession time proportion.



(a) object number=4 (b) object number=10 (c) object number=20 (d) object number=30

Fig. 6. The proportion of delay (or concession) time of *Random Split*, *Perfect Matching + DP*, *Greedy Search*, and *Ours* (lower is better). "With delay" corresponds to the situation where one arm makes a concession for the other, and "Without delay" is the case of both arms at work.

TABLE II

RESULTS OF AVERAGE MAKESPAN ON THE TEST SET OF **FS** SAMPLING(1000 INSTANCES EACH).

Method	n=4	n=6	n=10	n=14	n=20	n=30
Perfect Matching + DP	38.23	55.44	89.05	121.80	171.72	252.30
Random Split	39.65	57.75	96.75	136.41	191.67	285.84
Greedy Search	38.22	55.67	89.13	121.23	**170.08**	**248.59**
No_object_encoder	38.17	55.39	92.80	125.87	177.20	259.57
No_arm_encoder	38.12	55.27	87.43	120.51	170.19	252.09
Ours	**37.84**	**54.64**	**87.05**	**120.04**	170.15	251.22

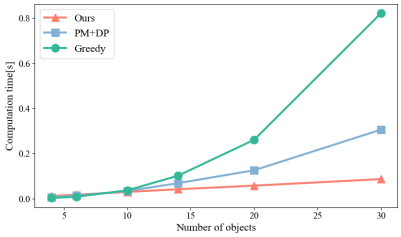
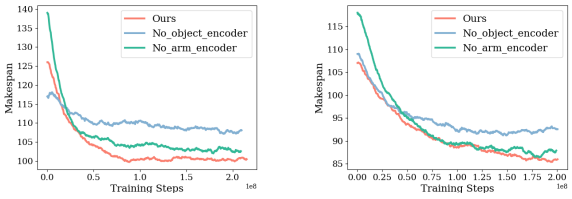
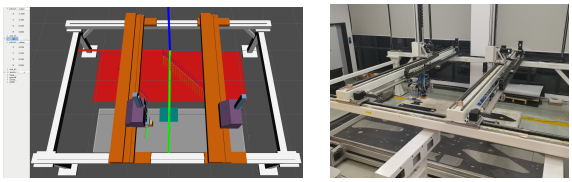


Fig. 7. Comparison of the average computation time.



(a) CA Sampling (b) FS Sampling

Fig. 8. **Training ablation.** Makespan over the training steps. We smoothed all the curves with a smoothing rate of 0.98 to facilitate clearer.



(a) Simulator (b) Real robot

Fig. 9. Simulator and real robot of the rearrangement system.

This essentially translates to maximizing the concurrent engagement of both arms in the task.

Comparing the average makespan results of our method with those of two ablations, we can find that there is a larger gap between *No_object_encoder* and ours in both Table I and Table II. To gain deeper insights, we present the training comparison in Figure 8 and find that *No_object_encoder* tends to converge earliest in both two sampling schemes. We analyze this because the modeling of object-to-object positional association substantially influences the perception of the operational cost of potential assignment pairs, which causes a more significant impact on the overall results.

C. Simulator and real robot

We have built a dual-arm rearrangement system based on ROS [29], which consists of a motion control simulator and a real robot as shown in Figure 9, in which we integrate our task assignment and dual-arm motion planning algorithm.

The pick positions of all objects are obtained in advance by a visual positioning system. Once the assigned objects are determined, the accurate pick position and target place position will be published for both arms, then the motion planning will be made in the simulator before real execution.

The demonstration of motion planning in the simulator and the comparison of rearranging fixed numbers and positions of objects using heuristics and our task assignment method in the real scene are shown in the supplemental video.

VI. CONCLUSION

This paper introduces an online reinforcement learning method that addresses the dual-arm object rearrangement problem by studying an efficient task assignment strategy. Our approach emphasizes combinatorial optimality and leverages an attention-based network to model the dependencies among different states, leading to improved performance in long-sequence decision-making. Experimental results demonstrate the effectiveness and computational efficiency of our method, especially as the number of objects increases. In future work, we plan to extend our approach to more diverse and complex multi-robot cooperation scenarios.

VII. ACKNOWLEDGEMENT

This work was supported in part by the NSFC (62325211, 62132021, 62372457), the National Key Research and Development Program of China (2018AAA0102200), the Major Program of Xiangjiang Laboratory (23XJ01009) and the Huxiang Youth Talent Support Program (2021RC3071).

REFERENCES

[1] F. Beuke, S. Alatartsev, S. Jessen, C. Hanel, and A. Verl, "Online motion planning for dual-arm industrial robots," in *ISR 2018; 50th International Symposium on Robotics*. VDE, 2018, pp. 1–8.

[2] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez, "Integrated task and motion planning," *Annual review of control, robotics, and autonomous systems*, vol. 4, pp. 265–293, 2021.

[3] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *Computers & Operations Research*, vol. 134, p. 105400, 2021.

[4] J. Park, C. Kwon, and J. Park, "Learn to solve the min-max multiple traveling salesmen problem with reinforcement learning," in *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, 2023, pp. 878–886.

[5] S. Paul, P. Ghassemi, and S. Chowdhury, "Learning scalable policies over graphs for multi-robot task allocation using capsule attention networks," in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 8815–8822.

[6] H. Zhao, Q. She, C. Zhu, Y. Yang, and K. Xu, "Online 3d bin packing with constrained deep reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 1, 2021, pp. 741–749.

[7] H. Zhao, C. Zhu, X. Xu, H. Huang, and K. Xu, "Learning practically feasible policies for online 3d bin packing," *Science China Information Sciences*, vol. 65, no. 1, p. 112105, 2022.

[8] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio, "Neural combinatorial optimization with reinforcement learning," *arXiv preprint arXiv:1611.09940*, 2016.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[10] J. E. King, M. Cognetti, and S. S. Srinivasa, "Rearrangement planning using object-centric and robot-centric action spaces," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 3940–3947.

[11] A. Krontiris and K. E. Bekris, "Efficiently solving general rearrangement tasks: A fast extension primitive for an incremental sampling-based planner," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 3924–3931.

[12] S. H. Cheong, B. Y. Cho, J. Lee, C. Kim, and C. Nam, "Where to relocate?: Object rearrangement inside cluttered and confined environments for robotic manipulation," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 7791–7797.

[13] C. Nam, J. Lee, S. H. Cheong, B. Y. Cho, and C. Kim, "Fast and resilient manipulation planning for target retrieval in clutter," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 3777–3783.

[14] M. Danielczuk, A. Kurenkov, A. Balakrishna, M. Matl, D. Wang, R. Martín-Martín, A. Garg, S. Savarese, and K. Goldberg, "Mechanical search: Multi-step retrieval of a target object occluded by clutter," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 1614–1621.

[15] E. Huang, Z. Jia, and M. T. Mason, "Large-scale multi-object rearrangement," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 211–218.

[16] H. Song, J. A. Haustein, W. Yuan, K. Hang, M. Y. Wang, D. Kragic, and J. A. Stork, "Multi-object rearrangement with monte carlo tree search: A case study on planar nonprehensile sorting," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 9433–9440.

[17] Y. Cao, Z. Sun, and G. Sartoretti, "Dan: Decentralized attention-based neural network for the minmax multiple traveling salesman problem," in *International Symposium on Distributed Autonomous Robotic Systems*. Springer, 2022, pp. 202–215.

[18] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[19] R. Li, A. Jabri, T. Darrell, and P. Agrawal, "Towards practical multi-object manipulation using relational reinforcement learning," in *2020 ieee international conference on robotics and automation (icra)*. IEEE, 2020, pp. 4051–4058.

[20] T. Pan, A. M. Wells, R. Shome, and L. E. Kavraki, "A general task and motion planning framework for multiple manipulators," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 3168–3174.

[21] K. Gao and J. Yu, "Toward efficient task planning for dual-arm tabletop object rearrangement," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 10 425–10 431.

[22] R. Shome, K. Solovey, J. Yu, K. Bekris, and D. Halperin, "Fast, high-quality dual-arm rearrangement in synchronous, monotone tabletop setups," in *Algorithmic Foundations of Robotics XIII: Proceedings of the 13th Workshop on the Algorithmic Foundations of Robotics*. Springer, 2020, pp. 778–795.

[23] I. Umay, B. Fidan, and W. Melek, "An integrated task and motion planning technique for multi-robot-systems," in *2019 IEEE International Symposium on Robotic and Sensors Environments (ROSE)*. IEEE, 2019, pp. 1–7.

[24] H. Gao, X. Zhou, X. Xu, Y. Lan, and Y. Xiao, "Amarl: An attention-based multiagent reinforcement learning approach to the min-max multiple traveling salesmen problem," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[25] W. Kool, H. Van Hoof, and M. Welling, "Attention, learn to solve routing problems!" *arXiv preprint arXiv:1803.08475*, 2018.

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[28] Z. Galil, S. Micali, and H. Gabow, "An o(ev\logv) algorithm for finding a maximal weighted matching in general graphs," *SIAM Journal on Computing*, vol. 15, no. 1, pp. 120–130, 1986.

[29] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.