Task Allocation for Human-Robot Collaboration in Dynamic Task Stream Environments

Joe Castor, Matthew Blau, and Zora Zhuang

Abstract— **Human-Robot Collaboration (HRC) is the problem of humans and robots collaborating to complete shared tasks and goals. A key component of HRC is the distribution of tasks between humans and robots, such that both parties can work cohesively and reach maximum productivity in a shared workspace. This process is called task allocation. In previous studies, several frameworks for evaluating task allocation have been proposed with the assumption of static task streams, i.e. the set of tasks that require completion remains constant from the onset of the planning problem. However, in practice, many real-world problems involve dynamic task streams, where the task set changes as time progresses. In this study, to support the development of real-world task allocation systems, we present a framework for evaluating task allocation in dynamic task stream environments. We further demonstrate the utility of our proposed framework, testing commonly used task allocation algorithms on several challenging HRC problems.**

I. INTRODUCTION

Human-Robot Collaboration (HRC) [1] is a research topic of increasing relevance, especially in labour-intensive industries like manufacturing, warehousing and logistics. In a typical HRC scenario, humans and robots collaborate in the same physical workspace to complete a shared set of tasks. Effective HRC enables enhanced productivity by allowing both agents to work on tasks that align with their repsective strengths, compensating for the other party's weaknesses when needed.

Task allocation is a crucial component of HRC. Typically, this involves a task planner assigning actions to humans and robots, enabling them to work together towards a predefined goal. In practice, task planners should consider the capabilities and characteristics of the human [2] to ensure the feasibility of the suggested action-plan.

Previous work has predominantly focused on task planning scenarios with 'static task streams' [2], [3], [4]. In these problems, the task planner knows all the tasks it needs to complete from the onset of the planning problem. This set remains untouched throughout the planning process, only changing when an agent completes a task. However, this problem is not representative of many real-world HRC planning problems.

In the real world, task streams are not always static. Task allocators may need to consider 'dynamic task stream' problems where existing tasks can expire, and new ones can emerge as time progresses. One can imagine this problem in a manufacturing context, where humans and robots must collaborate to remove items from a moving conveyor belt. At any given point, a new part may appear on this conveyor belt. However, the agents may only have visibility over a bounded section of the belt, lacking prior knowledge of what is coming in. The sequence of new tasks may also be highly irregular. In these types of scenarios, effective task allocation ensures that the human and robot collaborate in a way that allows them to adapt to unforeseen tasks.

In this paper, we present a framework for evaluating task allocation systems for HRC in dynamic task stream environments. We rely on the Markov Decision Process (MDP) framework to represent task allocation problems in uncertain environments with changing human characteristics. We then demonstrate the use of this framework by evaluating several Monte-Carlo-based task allocation planning algorithms over a range of planning problems. In doing so, we show that this framework can support the development of advanced task allocation systems for real-world HRC use cases.

II. BACKGROUND AND RELATED WORK

A. *Markov Decision Processes*

Robotic systems designed for real-world scenarios must be able to plan under uncertainty. The Markov Decision Process (MDP) is a modelling framework for decision-making in uncertain environments, where the effects of actions are non-deterministic [5].

An infinite-horizon MDP problem is a tuple of 5 elements $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, where:

- $\mathcal{S}$ represents the state space, which includes all possible problem states.
- $\mathcal{A}$ represents the action space, which includes all possible actions or action combinations the agent(s) can perform.
- $\mathcal{T}$ denotes the transition function, which describes the probability of transitioning to the next state s' after taking an action a from state s.
- $\mathcal{R}$ represents the reward function, which returns the immediate reward taking action a in state s.
- γ represents the discount factor, which controls the diminishing value of future rewards relative to immediate ones.

Solving an MDP problem involves computing an optimal policy $\pi^* : \mathcal{S} \to \mathcal{A}$, an ideal mapping from states to actions. An optimal policy will maximise the following value function:

$$V^*(s) = \max_a \left(\mathcal{R}(s) + \gamma \sum_{s'} \mathcal{T}(s, a, s') V^*(s') \right), \quad (1)$$

which denotes expected future discounted reward.

B. Sampling-based MDP Solvers

A subset of MDP problems can be solved using exact methods like Value Iteration and Policy Iteration [6]. However, these methods suffer from the 'curse of dimensionality', becoming intractable as the state and action space grow. As such, these methods are impractical for many real-world problems.

In practical settings, sampling-based methods are more applicable. These approaches, although forgoing the guarantee of optimality provided by exact solvers, facilitate a marked decrease in problem complexity. The fundamental idea behind these methods is to examine a representative subset of the state space to determine the appropriate action, as opposed to performing an exhaustive search. Provided that the sampling effectively captures the environment's dynamics, this methodology can reduce the complexity of deriving an effective policy.

Monte Carlo Tree Search (MCTS) [7] is a frequently employed algorithm for online planning in Markov Decision Processes. MCTS performs a lookahead search commencing from the agent's present state to approximate the optimal value function (1). This approximation informs the selection of actions at each decision point. The strategy of MCTS is rooted in a sampling-based search, utilising a generative model $\mathcal{G}$, which provides a stochastic sample of the successor state $s' \sim \mathcal{T}(s'|s,a)$ along with the associated reward r, given a current state s and action a. MCTS leverages this model to simulate episodes within the MDP setting, thus refining the value function iteratively. This method aligns the sample complexity with the intrinsic difficulty of the MDP rather than the size of the state space, rendering it more suitable for large MDP problems common to real-world HRC environments.

C. HRC and Task Allocation

Effective task allocation in HRC environments often requires a consideration of the human's capabilities and characteristics. Human characteristics typically refer to innate human qualities, which rarely change as the human completes tasks. On the other hand, human capabilities refer to more dynamic attributes, such as the human's energy level.

Task allocation systems often account for human fatigue when considering human capabilities. Li et al. [3] propose a planning approach that considers human fatigue in its decision-making. They test their methodology in a scenario where humans and robots must collaborate to disassemble end-of-life products as efficiently as possible. In this paper, the authors model human fatigue as a function of time, $F(t) = 1 - e^{-\lambda t}$, where fatigue increases rapidly before gradually converging. λ is a hyperparameter that controls how rapidly human fatigue changes. Other planners extend their consideration of human characteristics by accounting for task complexity. Malik et al. propose a task allocation method for manufacturing assembly problems that consider the complexity of a task in parallel with the human's characteristics and capabilities before allocating it [4]. While both papers provide evaluation frameworks for HRC task allocation systems, both testing environments assume a static task stream.

Aside from assembly/disassembly problems, another common HRC testing scenario is conveyor belt unloading. Gorur et al. propose a framework for HRC in a conveyor belt unloading scenario in a Partially-Observable MDP environment [8]. While this conveyor belt scenario is an example of a dynamic task stream problem, Gorur et al. do not facilitate a 'task allocation' approach. Instead, they assume that humans act according to their free will and plan for robots to work effectively around this behaviour.

In this paper, we aim to fill this gap by presenting a framework for evaluating task allocators within a dynamic task stream conveyor belt scenario. By marrying the consideration of human characteristics and capabilities from [3] with the dynamic task stream nature of [8], we offer a new framework for assessing HRC task allocation algorithms for complex real world use cases.

III. METHODS

A. Conveyor Belt Scenario

In this section, we present our conveyor belt HRC scenario, shown in Figure 1. In this problem, a human and a robot must work together to unload boxes from a moving conveyor belt, and carry them to a designated drop off site. Their goal is to transfer as many boxes as possible before they move away and become unreachable. However, the robot and human cannot share the position or hold more than one thing at a time. There are a fixed number of pick up locations (and corresponding belt locations), with only one drop off location. In each time step, the conveyor belt moves one unit to the left.

B. MDP Model

Denoting the number of pick-up locations as n, we can define an MDP representing the following scenario as follows:

*a) **State**:* The state space can be defined as the Cartesian product of human state, robot state, belt state, and the counter state.

$$\mathcal{S} = \mathcal{S}_{human} \times \mathcal{S}_{robot} \times \mathcal{S}_{belt} \times \mathcal{S}_{counter}$$

where:

- $\mathcal{S}_{human} = position \times tiredness \times lr\_bias \times has\_box$ is the state of the human, including the human position $position \in \mathbf{N}^2$, the tiredness $tiredness \in [0,4]$ (a capability), the human's preference for pick-up locations $lr\_bias \in [0, n-1]$ (a characteristic), and whether or not they are carrying a box $has\_box \in 0, 1$.
- $\mathcal{S}_{robot} = position \times has\_box$ is the state of the robot, including the robot position $position \in \mathbf{N}^2$ and whether or not it is carrying a box $has\_box \in \{0, 1\}$.
- $\mathcal{S}_{belt} \in \{0,1\}^n$ is the state of the belt, representing whether or not there is a box at each belt site.
- $\mathcal{S}_{counter} \in \mathbf{N^4}$ includes counters that record the cumulative statistics of the states:

```
#######################################################
#     # XXX #     #     # XXX #     # XXX # XXX #     #
#     # X X #     #     # X X #     # X X # X X #     #
#     # XXX #     #     # XXX #     # XXX # XXX #     #
#######################################################
.  H  .     .     .     .     .     .     .     .     .
. HXH .     .     .     .     .     .     .     .     .
.  H  .     .     .     .     .     .     .     .     .
.......................................................
.                       .  R  .                       .
.                       . R R .                       .
.                       .  R  .                       .
.                       .......                       .
.                       .$$$$$.                       .
.                       .$$$$$.                       .
.                       .$$$$$.                       .
.......................................................
```

Fig. 1. Conveyor Belt Scenario

– N_{packed} as the number of boxes that have been successfully moved to the drop off site.
– N_{missed} as to the number of missed boxes.
– N_{time_step} as the number of time steps taken so far.
– $N_{time\_since\_rest}$ as the number of times steps since the human took their last rest.

b) ***Action:*** The action space can be defined as the Cartesian product of human action and robot action.

$$\mathcal{A} = \mathcal{A}_{robot} \times \mathcal{A}_{human}$$

where both agents can $goto\_1$, $goto\_2$, $goto\_3$,..., $goto\_n$, $pickup$, $goto\_drop$, $putdown$ and $rest$.

These actions correspond to the human or robot going to a pickup location, picking up a box, going to the drop-off location, dropping off a box and resting.

c) ***Transition Function:*** The transition function for our problem has been defined such that non-deterministic effects of actions result from the human's characteristics, as well as the robot's actuator errors. The consequence of human actions are as follows:

- For any human action, the probability that they fail to do the action and remain in their present state is $\frac{tiredness}{10}$.
- If they did not fail due to tiredness, the human actions $pickup$, $putdown$ and $goto\_drop$ all act deterministically.
- Other human movement actions ($goto\_1$, ..., $goto\_n$) have non determinism stemming from both the $tiredness$ and the lr_bias. That is, the human has a 50% chance of following the assigned movement action, plus a uniform probability of going to a cell in between the assigned cell and their lr_bias cell (inclusive). For example, if their lr_bias was belt 0 and the assigned action was $goto\_2$, there would be a $\frac{1}{2} + \frac{1}{6} = \frac{2}{3}$ chance of landing in front of belt 2, a $\frac{1}{6}$ chance of landing at belt 1, and a $\frac{1}{6}$ chance of landing at belt 0.
- After performing the action, if the action is not $rest$, the human's tiredness is updated using the following formula derived from [3]. In our implementation, the relative rate of accumulation λ is set to 0.2.

$$tiredness = max\_tiredness \times (1 - e^{-\lambda \cdot time\_since\_rest})$$

If the action is $rest$, then the human $tiredness$ is decreased by 3, with the minimum $tiredness$ being 0.

The consequences of robot actions are as follows:

- If the consequence of $\mathcal{A}_{robot}$ and $\mathcal{A}_{human}$ would result in a collision, the robot is punished by instead being moved to its rest location, where it will have to use another action to get back to a desirable location.
- Otherwise, robot actions $pickup$, $putdown$, $rest$ and $goto\_drop$ all act deterministically.
- Robot movement actions ($goto\_1$, ..., $goto\_n$) have an 80% chance of taking the robot to the intended location, and a 10% chance each for instead going one cell to the left or to the right of the intended cell. If the left or the right is off the grid, probability mass is added to the intended cell.

The human action and robot actions take place simultaneously in each time step, but the human action takes precedence over the robot action, as described above. The environment transitions after the actions take place as follows:

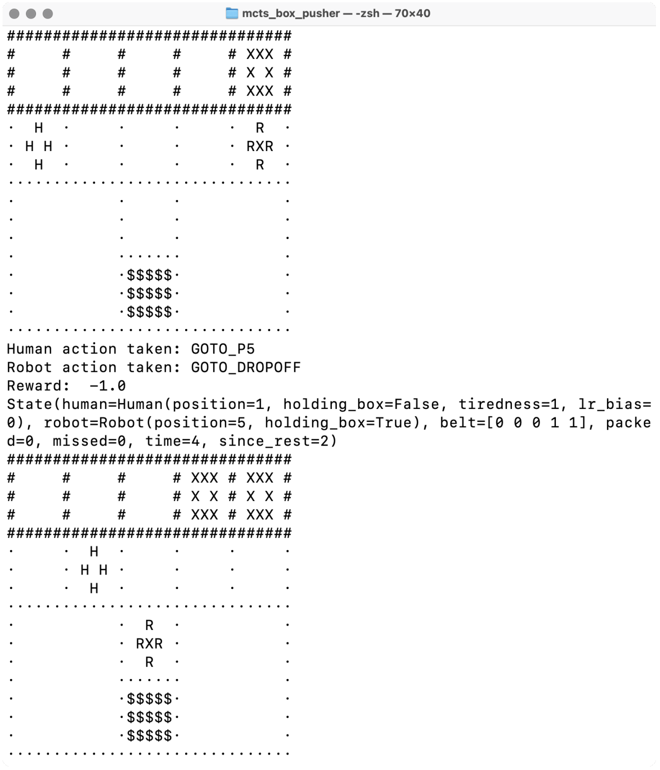
- First, boxes that were picked up are removed from the conveyor belt deterministically.
- The belt always shifts one cell to the left each time step. Boxes that are moved off the edge of the belt are permanently lost and added to the 'missed' count.
- The task stream is dynamic because a new box (task) may appear on the rightmost cell of the belt at any time. This event is non-deterministic, happening with a probability of 80%.
- Boxes that are successfully placed in the drop-off location increment the 'packed' count, which is the metric the agents are trying to optimise.

Figure 2 shows a visualisation of this problem. In the example, the conveyor belt is at the top and the drop-off location is at the bottom. Boxes are represented with 'X', human with 'H' and robot with 'R'. In Figure 2, we see that the robot is holding a box, so the planner assigns the robot action to be $goto\_drop$. The human is not holding a box, so the planner tells the human to go to belt position 5, as this is the action that is most likely to place the human in front of a box. However, since the human's lr_bias is 0 (i.e. biased towards the leftmost cell), the human ends up moving to belt position 2, which had a $\frac{1}{10}$ chance as described by the transition function.

d) ***Reward function:*** The objective of this problem is simply to pack as many boxes as possible, rather than having other considerations such as minimising human cost as seen in [2]. As such, the reward can be modelled simply as:

$$R(s, a, s') = -1 + (10 \times \frac{s'.packed}{s.time\_step})$$

This reward function can be broken into two parts. The first is the -1 cost for any action, and the second part represents the reward for packing boxes. Since $packed$ is a cumulative value, it must be divided by $time\_step$ such that the reward is not skewed based on depth.



```
###############################
#     #     #     #     # XXX #
#     #     #     #     # X X #
#     #     #     #     # XXX #
###############################
·  H  ·     ·     ·     ·  R  ·
· H H ·     ·     ·     · RXR ·
·  H  ·     ·     ·     ·  R  ·
·······························
·           ·     ·           ·
·           ·     ·           ·
·           ·     ·           ·
·           ·······           ·
·           ·$$$$$·           ·
·           ·$$$$$·           ·
·           ·$$$$$·           ·
·······························
Human action taken: GOTO_P5
Robot action taken: GOTO_DROPOFF
Reward:  -1.0
State(human=Human(position=1, holding_box=False, tiredness=1, lr_bias=
0), robot=Robot(position=5, holding_box=True), belt=[0 0 0 1 1], packe
d=0, missed=0, time=4, since_rest=2)
###############################
#     #     #     # XXX # XXX #
#     #     #     # X X # X X #
#     #     #     # XXX # XXX #
###############################
·     ·  H  ·     ·     ·     ·
·     · H H ·     ·     ·     ·
·     ·  H  ·     ·     ·     ·
·······························
·           ·  R  ·           ·
·           · RXR ·           ·
·           ·  R  ·           ·
·           ·······           ·
·           ·$$$$$·           ·
·           ·$$$$$·           ·
·           ·$$$$$·           ·
·······························
```

Fig. 2. Example Transition

Algorithm 1 Selecting Actions For Heuristic Simulation

```
if $human\_tiredness \geq 3$ then
    $human\_action = rest$
else if human can pickup box then
    $human\_action = pickup$
else if human can drop off box then
    $human\_action = putdown$
else if human holding box and not at drop-off location then
    $human\_action = goto\_drop$
else
    iterate over belt, go to location of box most similar to $lr\_bias$
end if
if robot can pickup box then
    $robot\_action = pickup$
else if robot can drop off box then
    $robot\_action = putdown$
else if robot holding box and not at drop-off location then
    $robot\_action = goto\_drop$
else
    iterate over belt, go to any box location that human is not trying to go to. If none, go to random belt location.
end if
return $(robot\_action, human\_action)$
```

C. Implementation

In solving this MDP problem, we designed a task-allocator using a standard MCTS approach with UCB1 [7]. Our first implementation uses a random rollout function, where the simulated value of a node is derived from performing completely random actions. We then extended our MCTS approach to use a heuristic based rollout. In this approach, the simulated value of a node is found by performing intelligent rather than random actions. A simple method was introduced for returning intelligent actions, which prioritises dropping off packages, picking up packages, and moving to cells with the highest chance of having a box in them in next time step.

In the heuristic based approach, we simulate the value of a node by performing 20 actions where the actions are determined by Algorithm 1, rather than by randomly selecting 20 actions.

IV. EVALUATION

To analyse the performance of the MCTS task allocator in our dynamic task stream environments, we conducted three experiments. In the first experiment, we tested the task allocator on a simple $n = 5$ conveyor belt problem over 50 trials. Then, we assessed the potential for performance improvements using the heuristic-based simulation approach described in Section III. Finally, we investigated how the difficulty of the task allocation problem affects the allocator's performance by varying the length of the conveyor belt.

All experiments were performed on a Linux desktop computer using a 6-core Intel Core i5-11500 CPU and 40GB RAM.

A. Initial testing

We first tested our task allocator in a simple dynamic task stream environment using a consistent conveyor belt input sequence over 50 trials, with each trial running 30 steps before termination. The conveyor belt input sequence was randomly generated, with an 80% likelihood of a new box appearing at any particular time step. In the initial state, the conveyor belt is empty. In the first 30 time steps, a total of 26 boxes have been put onto the belt. The complete design of this environment has been described in Section III.

For these experiments, the task allocator was given 3 seconds of planning time. Within our MCTS algorithm, we set a maximum tree depth of 20, an exploration factor of 10 and a discount factor of 0.85.

Table I presents the task allocator's average total reward accumulated by following the policy generated by the MCTS task-allocation algorithm. The $\pm x$ indicates the 95% confidence interval. To benchmark the performance, we repeated the same experiments with a random policy and a greedy policy. The random policy returns randomly generated actions, while the greedy policy returns a greedy action without taking human characteristics or capabilities into account.

From Table I, it is evident that the task allocator significantly outperforms both random policy and greedy policy,

TABLE I
AVERAGE TOTAL REWARD COMPARISON OF TASK ALLOCATORS

Task Allocator	Average Total Reward	Average Total Packed
Random Policy	-28.3 ± 1.1	0.30 ± 0.20
Greedy Policy	34.8 ± 2.7	8.28 ± 0.31
MCTS Task Allocator	44.8 ± 2.6	9.22 ± 0.26

achieving a considerably greater average total reward as well as a greater average number of packed boxes. This outcome validates the usefulness of MCTS as an approach to task allocation in dynamic task stream MDP problems.

B. Improving the task allocator

We also assessed the potential for improvements in the MCTS task allocator using a heuristic-based simulation approach. Instead of the random simulation approach, we ran experiments using the heuristic described in Algorithm 1. Table II presents the result across 50 trials. Additionally, Figure 3 shows the distribution of total reward and total packed for the 4 different allocators (random policy, greedy policy, MCTS with random rollouts, MCTS with heuristic rollouts.) across different trials

TABLE II
AVERAGE TOTAL REWARD COMPARISON OF TASK ALLOCATORS

Task Allocator	Average Total Reward	Average Total Packed
MCTS (Random)	44.8 ± 2.6	9.22 ± 0.26
MCTS (Heuristic)	52.8 ± 1.7	10.26 ± 0.20

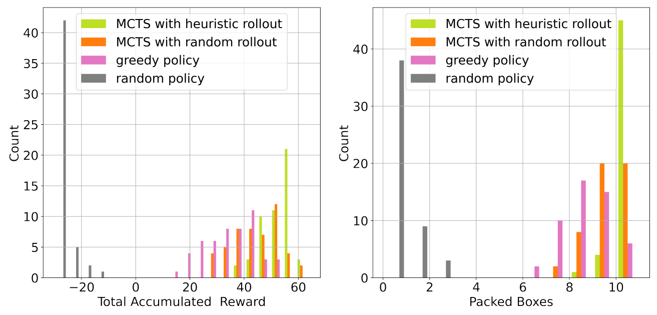


Fig. 3. The distribution of total reward (left) and total packed (right) of different task allocators.

These results show that the heuristic-based task allocator significantly outperforms the original, reaching a much greater average total reward across trials.

We also wanted to investigate how effectively the planner took into account human characteristics and capabilities when assigning tasks. Table III shows that the MCTS (Heuristic) task allocator relies on the *rest* action significantly more than the random policy, which doesn't account for the human's tiredness. This is likely a key contributing factor behind the planner's superior performance.

As well as accounting for the human's tiredness level, the MCTS planner also effectively accounts for the lr_bias of the human. For example, when running trials with lr_bias set to 0, if there is one box on the left of the belt and one box on the right of the belt, the planner will often assign the human to go to the left box location rather than the right. This is because the human is less likely to transition to the assigned cell if it is further from its bias. As such, there is a greater likelihood of being able to pick up a box on the next time step if the allocater assigns the human to go to the left box instead.

TABLE III
AVERAGE REST COUNT COMPARISON OF TASK ALLOCATORS

Task Allocator	Average Rest Count
Random Policy	6.5
MCTS (Heuristic)	3.3

C. Scaling the problem size

To better understand how the difficulty of the dynamic task stream MDP problem scales with the size of the state and action spaces, we tested our task allocator on the same problem with longer conveyor belts. This change involved creating new pickup states corresponding to new conveyor belt positions and new actions to get to said states. All other parameters remained the same as in the previous experiment. Table IV presents the results over 50 trials.

TABLE IV
AVERAGE TOTAL REWARDS FOR A TASK ALLOCATOR ACROSS DIFFERENT PROBLEM SIZES

Task Allocator	Average Total Reward	Average Total Packed
$n = 5$	52.8 ± 1.7	10.26 ± 0.20
$n = 9$	50.6 ± 1.9	10.08 ± 0.18
$n = 13$	46.5 ± 3.0	9.46 ± 0.28

These results suggest that the task allocator performs worse as the complexity of the conveyor belt problem increases. Despite an increased margin for error in n=9 and n=13 conveyor belt problems, given the increased time for the task allocator to handle a box before it is 'missed', the allocator exhibits a steady decrease in performance as the belt size increases.

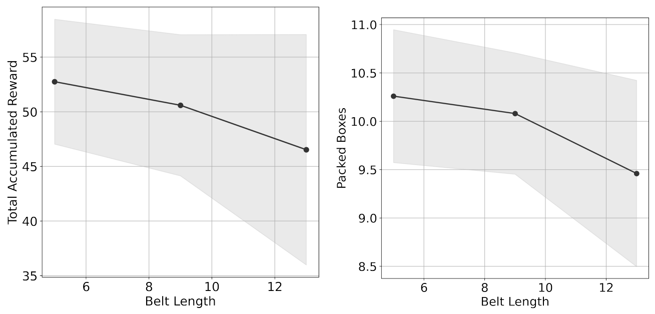


Fig. 4. The average total reward (left) and total packed (right) at different belt lengths. The shadow indicates standard deviation.

This trend, visible in Figure 4, is expected given that sampling-based planning approaches like MCTS may achieve less effective coverage of the state space in larger environments, given a constrained planning time. This planning difficulty could be further excacerbated by our transition function, which creates more uncertainty in larger environments due to the effect of the human's lr_bias.

V. Conclusion and Future Work

In this project, we have presented a framework for evaluating task allocation planners for human-robot collaboration in dynamic task stream environments. We demonstrate the utility of this framework, testing commonly used planning algorithms on challenging planning problems. In our experiments, we show that a heuristic-based MCTS significantly outperforms a random policy, a greedy policy and an MCTS approach with random simulations. Furthermore, the performance of the MCTS solver worsens as the size of the state and action space increases. By developing this framework, we have laid the foundation for improved task allocation algorithms for human-robot collaboration in dynamic task stream environments.

To extend the real-world applicability of our evaluation framework, we could change the problem scenario to contain partial observability. In our current MDP approach, we assume the human's characteristics and capabilities (lr_bias, $tiredness$) are known. However, in the real world, these traits are typically partially observable. To account for this partial observability, we could model the problem as a POMDP, where the robot observes the human's behaviour and uses these observations to inform its belief about the human's characteristics.

A further extension to improve the real-world applicability of our system would be to attach a motion planning component to our evaluation framework. As a task allocation framework, our problem does not consider the series of individual motions required to complete each step in a task. Consequently, the task planner only considers collisions in the robot and human's final positions. By revising the framework to account for the motions of the human and the robot, we could improve the applicability of the evaluation framework to real world scenarios.

Code Availability

Please see BoxPusher to view the code for this project.

Acknowledgment

We would like to express our sincere gratitude to Ruija (Eric) Zhou for his generous guidance and support throughout this research project. We have greatly benefited from his in-depth understanding of Human-Robot Collaboration.

References

[1] A. Bauer, D. Wollherr, and M. Buss, "Human–robot collaboration: a survey," *International Journal of Humanoid Robotics*, vol. 5, no. 01, pp. 47–66, 2008.

[2] Y. Y. Liau and K. Ryu, "Task allocation in human-robot collaboration (hrc) based on task characteristics and agent capability for mold assembly," *Procedia manufacturing*, vol. 51, pp. 179–186, 2020.

[3] K. Li, Q. Liu, W. Xu, J. Liu, Z. Zhou, and H. Feng, "Sequence planning considering human fatigue for human-robot collaboration in disassembly," *Procedia CIRP*, vol. 83, pp. 95–104, 2019.

[4] A. A. Malik and A. Bilberg, "Complexity-based task allocation in human-robot collaborative assembly," *Industrial Robot: the international journal of robotics research and application*, vol. 46, no. 4, pp. 471–480, 2019.

[5] M. L. Puterman, "Markov decision processes," *Handbooks in operations research and management science*, vol. 2, pp. 331–434, 1990.

[6] S. J. Russell and P. Norvig, *Artificial intelligence a modern approach*. London, 2010.

[7] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European conference on machine learning*, pp. 282–293, Springer, 2006.

[8] O. C. Görür, B. Rosman, F. Sivrikaya, and S. Albayrak, "Fabric: A framework for the design and evaluation of collaborative robots with extended human adaptation," *ACM Transactions on Human-Robot Interaction*, vol. 12, no. 3, pp. 1–54, 2023.