

MRBTP: Efficient Multi-Robot Behavior Tree Planning and Collaboration

Yishuai Cai*, Xinglin Chen*, Zhongxuan Cai[†], Yunxin Mao,
Minglong Li[†], Wenjing Yang, Ji Wang

College of Computer Science and Technology, National University of Defense Technology
 {cayishuai, chenxinglin, caizhongxuan, maoyunxin, liminglong10, wenjing.yang, wj}@nudt.edu.cn

Abstract

Multi-robot task planning and collaboration are critical challenges in robotics. While Behavior Trees (BTs) have been established as a popular control architecture and are plannable for a single robot, the development of effective multi-robot BT planning algorithms remains challenging due to the complexity of coordinating diverse action spaces. We propose the Multi-Robot Behavior Tree Planning (MRBTP) algorithm, with theoretical guarantees of both soundness and completeness. MRBTP features cross-tree expansion to coordinate heterogeneous actions across different BTs to achieve the team's goal. For homogeneous actions, we retain backup structures among BTs to ensure robustness and prevent redundant execution through intention sharing. While MRBTP is capable of generating BTs for both homogeneous and heterogeneous robot teams, its efficiency can be further improved. We then propose an optional plugin for MRBTP when Large Language Models (LLMs) are available to reason goal-related actions for each robot. These relevant actions can be pre-planned to form long-horizon subtrees, significantly enhancing the planning speed and collaboration efficiency of MRBTP. We evaluate our algorithm in warehouse management and everyday service scenarios. Results demonstrate MRBTP's robustness and execution efficiency under varying settings, as well as the ability of the pre-trained LLM to generate effective task-specific subtrees for MRBTP.

Code — <https://github.com/DIDS-EI/MRBTP>

Introduction

Multi-robot systems (MRS) that involve robots with diverse capabilities offer the potential for improved performance and fault tolerance compared to single-robot solutions (Colledanchise et al. 2016). Developing an autonomous MRS requires an efficient and robust control architecture, along with methods to adapt them for specific tasks. Behavior Trees (BTs) have emerged as a popular control architecture due to their modularity, interpretability, reactivity, and robustness, making them well-suited for both single- and multi-robot systems (Heppner et al. 2024; Heppner et al. 2023; Neupane and Goodrich 2019; Colledan-

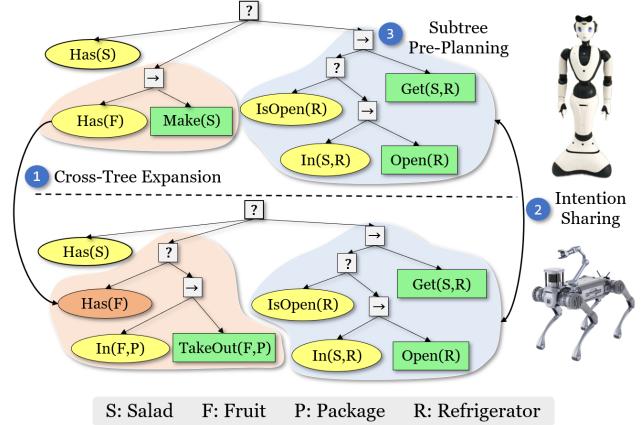


Figure 1: An example of two BTs planned by MRBTP: (1) Cross-tree expansion, (2) Intention sharing, (3) Optional plugin: subtree pre-planning.

chise et al. 2016). As the potential of BTs gains more attention, various methods for automatically generating BTs have been proposed, including evolutionary computing (Neupane and Goodrich 2019; Colledanchise, Parasuraman, and Ögren 2019), reinforcement learning (Banerjee 2018; Pereira and Engel 2015), and BT synthesis (Tadewos, Newaz, and Karimoddini 2022; Neupane and Goodrich 2023). Among these methods, BT planning (Chen et al. 2024a,b; Cai et al. 2021; Colledanchise, Almeida, and Ogren 2019) has advantages in leveraging interpretable action models and producing reliable BTs to achieve goals, which make it a promising approach to generate BTs for autonomous robot systems.

However, current BT planning focuses primarily on a single robot, and the development of effective multi-robot BT planning algorithms remains challenging. The challenges mainly arise from two aspects:

- For heterogeneous actions, how to coordinate them across different BTs to accomplish the team's goal.
- For homogeneous actions, how to use them to improve fault tolerance without redundant execution.

In this paper, we propose Multi-Robot Behavior Tree Planning (MRBTP), the first sound and complete algorithm for generating reliable and robust BTs for MRS. MRBTP

*These authors contributed equally.

[†]Corresponding Author

Copyright © 2025, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

addresses the above challenges as follows:

- We employ cross-tree expansion, where the condition expanded in one BT will be further expanded by all BTs. This means one robot may take an action to satisfy another's precondition, enabling multi-tree collaboration.
- We allow backup structures to be expanded by robots with homogeneous actions to ensure fault tolerance, while using intention sharing to avoid redundant execution. During execution, each robot broadcasts its current action so that others can predict its effects and avoid performing actions with the same effects.

As shown in Figure 1, in an everyday service scenario, the team's goal is to prepare a Salad. The humanoid robot can perform the action `Make(Fruit, Salad)`, if the precondition `Has(Fruit)` is satisfied. Although it cannot do `Unload(Fruit, Package)`, it can push `Has(Fruit)` to the planning queue, and another quadruped robot will expand this action through cross-tree expansion, enabling multi-tree coordination.

In another case where `In(Salad, Refrigerator)` is satisfied, and both robots can do `Open(Refrigerator)` and `Get(Salad)`. In this case, MRBTP will expand the same structure in both trees to ensure failure tolerance. If both robots are available, the humanoid robot with higher priority will do `Open(Refrigerator)` and share its intention. The quadruped robot will then assume `IsOpen(Refrigerator)` is true and walk to the Refrigerator, waiting to do `Get(Salad, Refrigerator)` as long as the Refrigerator is truly open. The intention sharing ensures parallelization and improves execution efficiency of the robot team.

Although MRBTP is a domain-independent algorithm, it is possible to enhance the planning and execution efficiency if Large Language Models (LLMs) are available for domain-dependent reasoning. Therefore, we further propose an optional plugin named subtree pre-planning. Assuming the LLM can reason some useful actions for each robot according to its capabilities, we can use these actions to plan useful subtree structures quickly before the long-horizon planning process. These subtrees can not only increase the planning speed, but can also decrease the communication expenses during execution. Experiments in warehouse management and everyday service scenarios demonstrate MRBTP's robustness and execution efficiency under varying settings, as well as the ability of pre-trained LLMs to generate effective task-specific subtrees for MRBTP.

Background

Behavior Tree. A BT is a directed rooted tree where the execution nodes interact with the environment and the control flow nodes handle the triggering logic of their children(Colledanchise and Ögren 2018). At each time step, the BT initiates a tick that goes through control nodes, determining the action that the robot will execute according to the environmental state. This paper mainly focuses on four typical BT nodes:

- **Condition** : An execution node that checks whether

the environment state satisfies the specified condition, returning either success or failure accordingly.

- **Action** : An execution node that controls the robot to perform an action, returning success, failure, running depending upon the outcome of execution.
- **Sequence** : A control flow node that only returns success if all its children succeed. Otherwise, it ticks its children from left to right, and the first child to return failure or running will determine its return status.
- **Fallback** : A control flow node with logic opposite to the sequence node. It returns failure only if all of its children fail. If not, the first occurrence of success or running during ticking becomes its return status.

BT Planning. In BT planning for a single robot (Cai et al. 2021), we represent a BT as a three-tuple $\mathcal{T} = \langle f, r, \Delta t \rangle$. $f : 2^n \rightarrow 2^n$ is its effect on the environment state, Δt is the time step, and $r : 2^n \mapsto \{S, R, F\}$ partitions states into three regions, where S returns success, running, failure, respectively.

Then the BT planning problem can be described as: $\langle S, \mathcal{L}, \mathcal{A}, \mathcal{M}, s_0, g \rangle$, where S is the finite set of environment states, \mathcal{L} is the finite set of literals that form states, \mathcal{A} is the finite set of actions, \mathcal{M} is the action model, s_0 is the initial state, g is the goal condition.

A condition c in BT is usually a subset of a state s . If $c \subseteq s$, it is said condition c holds in that state s . The state transition affected by action $a \in \mathcal{A}$ can be defined as a triplet $\mathcal{M}(a) = \langle \text{pre}(a), \text{add}(a), \text{del}(a) \rangle$, comprising the precondition, add effects, and delete effects of the action. If a is finished after k time step, the subsequent state $s_{t'}$ will be:

$$s_{t'} = f_a(s_t) = s_t \cup \text{add}(a) \setminus \text{del}(a), t' = t + k \quad (1)$$

Problem Formulation

We first extend the BT representation from a single robot to a multi-robot system.

Definition 1 (Multi-BT System). A n -robot BT system is a four-tuple $\langle \Phi, f_\Phi, r_\Phi, \Delta t_\Phi \rangle$, where $\Phi = \{\mathcal{T}_i\}_{i=1}^n$ is the set of BTs, $f_\Phi : S \mapsto S$ is the team state transition function, Δt_Φ is the team time step, $r_\Phi : S \mapsto \{S, R, F\}$ is the team region partition.

Due to variability in hardware performance, we allow each robot's BT to have a different response frequency, with Δt_Φ representing the common minimum response interval. The state transition can be calculated as follows:

$$s_{t+\Delta t_\Phi} = f_\Phi(s_t) = s_t \cup \bigcup_{i=1}^n (\text{add}(a_i) \setminus \text{del}(a_i)) \quad (2)$$

where a_i is the action of robot i in time t . If robot i does not have an action or its action is running, we let $\text{add}(a_i) = \text{del}(a_i) = \emptyset$.

The team region partition can be calculated as follows:

$$r_\Phi(s) = \begin{cases} R & \text{if } \exists i, r_i(s) = R \\ S & \text{if } \forall i, r_i(s) \neq R \text{ and } \exists i, r_i(s) = S \\ F & \text{if } \forall i, r_i(s) = F \end{cases} \quad (3)$$

The status of Φ is R if any BT is still running, S if some BT returns success and no one is running, and F if all BT fails.

Algorithm 1: One-step cross-tree expansion

```

1: function ExpandOneRobot ( $\mathcal{T}, \mathcal{A}, c$ )
2:    $\mathcal{T}_{new} \leftarrow c$                                  $\triangleright$  newly expanded subtree
3:    $\mathcal{C}_{new} \leftarrow \emptyset$                           $\triangleright$  newly expanded conditions
4:   for each action  $a \in \mathcal{A}$  do
5:     if  $c \cap (pre(a) \cup add(a) \setminus del(a)) \neq \emptyset$  and  $c \setminus del(a) = c$  then
6:        $c_a \leftarrow pre(a) \cup c \setminus add(a)$ 
7:        $\mathcal{T}_a \leftarrow Sequence(c_a, a)$ 
8:        $\mathcal{T}_{new} \leftarrow Fallback(\mathcal{T}_{new}, \mathcal{T}_a)$ 
9:        $\mathcal{C}_{new} \leftarrow \mathcal{C}_{new} \cup \{c_a\}$ 
10:    if  $\mathcal{C}_{new} \neq \emptyset$  then
11:      if ConditionInTree( $c, \mathcal{T}$ ) then
12:        Replace  $c$  with  $\mathcal{T}_{new}$  in  $\mathcal{T}$            $\triangleright$  in-tree expand
13:      else if  $\mathcal{T}_{new} \neq c$  then
14:         $\mathcal{T} \leftarrow Fallback(\mathcal{T}, \mathcal{T}_{new})$      $\triangleright$  cross-tree expand
15:    return  $\mathcal{T}, \mathcal{C}_{new}$ 

```

Definition 2 (Finite Time Successful). Φ is finite time successful (FTS) from region of attraction (ROA) R to condition c , if $\forall s_0 \in R$ there is a finite time τ such that for any $t < \tau$, $r_\Phi(s_t) = R$, and for any $t \geq \tau$, $r_\Phi(s_t) = S$, $c \in s_t$.

With definitions above, the multi-robot BT planning problem can finally be defined.

Problem 1 (Multi-Robot BT Planning). *The problem is a tuple $\langle \mathcal{S}, \mathcal{L}, \{\mathcal{A}_i\}_{i=1}^n, \mathcal{M}, s_0, g \rangle$, where \mathcal{S} is the finite set of environment states, \mathcal{L} is the finite set literals that form states and conditions, \mathcal{A}_i is the finite action set of robot i , \mathcal{M} is the action model, s_0 is the initial state, g is the goal condition. A solution to this problem is a BT set $\Phi = \{\mathcal{T}_i\}_{i=1}^n$ built with $\{\mathcal{A}_i\}_{i=1}^n$, such that Φ is FTS from $R \ni s_0$ to g .*

Methods

We first detail MRBTP, analyzing its soundness, completeness, and computational complexity. Then, we demonstrate how intention sharing functions among BTs during execution. Finally, we introduce the optional plugin, subtree pre-planning, to further enhance efficiency.

Multi-Robot Behavior Tree Planning

One-Step Cross-Tree Expansion Algorithm 1 gives the pseudocode of one-step cross-tree expansion for one robot. Given its current BT \mathcal{T} , action space \mathcal{A} and the condition to expand c (line 1), the function returns an expanded BT \mathcal{T} along with the newly expanded condition set \mathcal{C}_{new} (line 15). Similar to one-step expansion for a single robot (Cai et al. 2021), the expansion begins with $\mathcal{T}_{new} = c$ and $\mathcal{C}_{new} = \emptyset$ (line 2-3). Then we go through the action space \mathcal{A} to find all premise actions that can lead to c (line 4-5). For each premise action a , we calculate its corresponding precondition c_a (line 6), form a sequence structure \mathcal{T}_a (line 7), and add \mathcal{T}_a to the tail of the root fallback node of \mathcal{T}_{new} (line 8). Now \mathcal{T}_{new} can achieve c using these expanded actions if their precondition are met. We store these preconditions in \mathcal{C}_{new} (line 9).

Algorithm 2: MRBTP

```

Input: problem  $\langle \mathcal{S}, \mathcal{L}, \{\mathcal{A}_i\}_{i=1}^n, \mathcal{M}, s_0, c \rangle$ 
Output: solution  $\Phi = \{\mathcal{T}_i\}_{i=1}^n$ 
1:  $\mathcal{C}_U \leftarrow \{g\}$                                  $\triangleright$  conditions to be explored
2:  $\mathcal{C}_E \leftarrow \emptyset$                            $\triangleright$  expanded conditions
3: for  $i = 1$  to  $n$  do
4:    $\mathcal{T}_i \leftarrow Fallback(g)$                    $\triangleright$  init the BTs
5:   while  $\mathcal{C}_U \neq \emptyset$  do
6:      $c \leftarrow \text{Pop}(\mathcal{C}_U)$                  $\triangleright$  explore  $c$ 
7:     if HasSubSet ( $c, \mathcal{C}_E$ ) then continue     $\triangleright$  prune
8:     for  $i = 1$  to  $n$  do
9:        $\mathcal{T}_i, \mathcal{C}_{new} \leftarrow \text{ExpandOneRobot} (\mathcal{T}_i, \mathcal{A}_i, c)$ 
10:      if HasSubSet ( $s_0, \mathcal{C}_{new}$ ) then
11:        return  $\Phi = \{\mathcal{T}_i\}_{i=1}^n$              $\triangleright$  return a solution
12:       $\mathcal{C}_E \leftarrow \mathcal{C}_E \cup \mathcal{C}_{new}$ 
13:       $\mathcal{C}_U \leftarrow \mathcal{C}_U \cup \mathcal{C}_{new}$            $\triangleright$  add new conditions
14:    return Unsolvable

```

If \mathcal{T}_{new} is expanded (line 10), we need to decide where in \mathcal{T} to place it. There are two cases: (1) c is in \mathcal{T} , which means it was previously expanded by \mathcal{T} itself. So we replace c with \mathcal{T}_{new} in \mathcal{T} just like in single-robot BT expansion (line 11-12); (2) c is not in \mathcal{T} , which means it was expanded by other BTs. To allow this BT to take actions to fulfill c , we add it to the tail of the root fallback node of \mathcal{T} (line 13-14).

Proposition 1. *Given \mathcal{T} is FTS from R to g , if \mathcal{T} is expanded by Algorithm 1 to \mathcal{T}' given c , c is in \mathcal{T} and $\mathcal{C}_{new} \neq \emptyset$, then \mathcal{T}' is FTS from $R' = R \cup \{s \in \mathcal{S} | c_a \subseteq s, c_a \in \mathcal{C}_{new}\}$ to g .*

Proposition 2. *If \mathcal{T} is expanded by Algorithm 1 to \mathcal{T}' given c , c is not in \mathcal{T} and $\mathcal{C}_{new} \neq \emptyset$, then \mathcal{T}' is FTS from $S_{new} = \{s \in \mathcal{S} | c_a \subseteq s, c_a \in \mathcal{C}_{new}\}$ to c .*

The above two propositions state the changes in the ROA after one-step cross-tree expansion. If c is in \mathcal{T} (Proposition 1), the ROA of \mathcal{T} will be expanded by \mathcal{C}_{new} to achieve g . If c is not in \mathcal{T} (Proposition 2), then c will be treated as a new sub-goal for \mathcal{T} to be achieved from S_{new} .¹

MRBTP Algorithm 4 gives the pseudocode of MRBTP to plan BTs for the whole robot team. The algorithm initializes a set of conditions to be explored $\mathcal{C}_U = \{g\}$ and a set of expanded conditions $\mathcal{C}_E = \emptyset$ (line 1-2). The BT for each robot i is initialized as $\mathcal{T}_i = Fallback(g)$ (line 4), which is FTS from \emptyset to g . Then the algorithm continually explores conditions in \mathcal{C}_U (line 5-6) until a solution is found, otherwise it returns *Unsolvable* (line 14). For each explored c , it is either pruned if $\exists c' \in \mathcal{C}_E, c' \subseteq c$ (line 7), or expanded by all robots through one-step cross-tree expansion (line 8-9). After the one-step expansion for each robot, the newly expanded conditions \mathcal{C}_{new} will be appended to \mathcal{C}_E and \mathcal{C}_U (line 12-13). If at that time $\exists c' \in \mathcal{C}_{new}, c' \subseteq s_0$, which means a solution is found, the algorithm returns $\Phi = \{\mathcal{T}_i\}_{i=1}^n$ as the solution (line 10-11).

Proposition 3. : *After the k -th ($k \geq 1$) iteration of the while loop in Algorithm 4, where the explored condition is*

¹All the formal proofs are in the Appendix.

$c^k, \Phi^k = \{\mathcal{T}_i^k\}_{i=1}^n$ is FTS from ROA $R^k = R^{k-1} \cup \bigcup_{i=1}^n \mathcal{S}_i^k$ to goal g , where $\mathcal{S}_i^k = \{s \in \mathcal{S} | c_a \subseteq s, c_a \in C_{i,new}^k\}$.

Note that Proposition 3 cannot be naturally extended from the single BT planning as it might seem. This proposition requires the assumption that robots execute in an appropriate order (at any time step, only the robot with the highest priority can execute an action if its precondition is satisfied); otherwise, deadlocks or departures from the ROA could occur. Fortunately, we can always use mechanisms such as deadlock detection during execution, ensuring that this serial execution is only employed in exceptional cases. In the vast majority of cases, robots can safely execute in parallel, so there is no need to worry that this assumption will reduce the execution efficiency of the robot team.

Proposition 4. *Algorithm 4 is sound, i.e. if it returns a result Φ rather than `Unsolvable`, then Φ is a solution of Problem 1.*

Proposition 5. *Algorithm 4 is complete, i.e., if Problem 1 is solvable, the algorithm returns a Φ which is a solution.*

Proposition 9 can be proven by strong induction based on Proposition 8, and Proposition 10 can be proven based on Proposition 9. These two propositions state the soundness and completeness of MRBTP, which makes it an effective algorithm to solve the multi-robot BT planning problem.

The time complexity of MRBTP in the worst case is $O(|\bigcup_{i=1}^n \mathcal{A}_i| |\mathcal{S}| |\mathcal{L}|)$, which is polynomial to the system size. In this case, the algorithm has to explore all states $s \in \mathcal{S}$ to find a solution. And in each exploration, the actions of all robots will be checked, with the checking complexity of $O(|\mathcal{L}|)$.

Intention Sharing for Multi-BT Execution

From the MRBTP planning process, we can observe that if multiple robots have identical actions (or similar actions with the same effect), MRBTP will expand them simultaneously in different BTs. This could lead to backup structures. These structures are beneficial for fault tolerance because if one robot fails, others can take over and complete the action. However, when multiple robots are available, backup structures can result in redundant execution. To avoid this, we introduce the multi-BT intention sharing method based on communication.

Intention Queue During execution, each robot i maintains an intention queue $\mathcal{I}_i = (a_1, a_2, \dots, a_m)$ that indicates the actions being performed by other robots. In a situation with good communication, all robots' intention queues should remain consistent. Therefore, in the following text, we use \mathcal{I} to refer to the intention queues of all robots. Based on the intention queue, we can calculate the belief success space \mathcal{B}_i^S and the belief failure space \mathcal{B}_i^F for robot i :

$$\mathcal{B}_i^S = \bigcup_{k=1}^{j-1} (\text{add}(a_k) \setminus \text{del}(a_k)) \quad (4)$$

$$\mathcal{B}_i^F = \bigcup_{k=1}^{j-1} (\text{del}(a_k) \setminus \text{add}(a_k)) \quad (5)$$

where j is the index of its own action a_j in the intention queue \mathcal{I} . If $j = 1$, then $\mathcal{B}_i^S = \mathcal{B}_i^F = \emptyset$, which means the action is not dependent on any other's intention. If the robot currently has no action, it will be treated as $j = m + 1$ when calculating belief spaces.

\mathcal{B}^S and \mathcal{B}^F will be used during the ticks of each BT. For each atomic condition node represented by a single literal $c = l$, it will first check if l is in the belief spaces when ticked. If $l \in \mathcal{B}^S$, it returns `S` without interacting with the environment, and returns `F` when $l \in \mathcal{B}^F$.

Whenever a robot i exits an action or enters a new one, it will be broadcast to every other robot. Each robot then removes the old action of robot i from the intention queue \mathcal{I} (if it exists) and pushes the new action into it (if applicable). After this, each robot will update its belief spaces \mathcal{B}^S and \mathcal{B}^F to adjust its actions reactively. Note that an action exiting or entering may be due to two cases: (1) the environment state has changed, or (2) the belief spaces have changed. As a result, any addition or removal of actions in the intention queue \mathcal{I} may lead to adjustments in other actions, creating a chain reaction. In other words, our intention-sharing method maintains the reactivity and robustness of BTs in response to uncertain environments.

Parallelism and Blocking While intention sharing can avoid redundant execution, it also enhances action parallelism within the robot team. For example, as shown in Figure 2, in a warehouse management scenario, there are two robots capable of opening doors and transporting packages. They have expanded almost identical tree structures, sequentially executing `Open(Door)`, `Walk(Package)`, and `Move(Package)`. However, since `IsClose(Door)` $\in s_0$, both robots satisfy the precondition to execute `Open(Door)`. Without intention sharing, they would perform this action simultaneously, causing redundancy. With intention sharing, however, if robot 1 ticks its BT \mathcal{T}_1 first, it will execute `Open(Door)` and send this intention to robot 2. For robot 2, `IsOpen(Door)` $\in \mathcal{B}_2^S$ after updating the intention queue \mathcal{I} , so the corresponding condition node for `IsOpen(Door)` will return `S`, allowing the BT \mathcal{T}_2 to continue ticking and start executing `Walk(Package)`. This transforms a serial BT structure into parallel execution.

However, when robot 2 attempts to execute `Move(Package)`, which relies on the precondition `IsOpen(Door)`, robot 2 will wait until the door is actually opened by robot 1. Formally speaking, if $l \in \mathcal{B}_i^S$ but not in the current state $l \notin s$, when robot i attempts to perform an action a where $l \in \text{pre}(a)$, a will be blocked. In this case, robot i shares the intention of a , and a returns `R` as if it were executing, but it is actually doing nothing. The blocking mechanism prevents actions from being executed under incorrect preconditions, while also enabling the parallel execution of subsequent actions, thereby further enhancing the execution efficiency of the robot team.

Optional Plugin: Subtree Pre-planning

While MRBTP with intention sharing is proven to be an effective and efficient algorithm for the multi-robot BT planning problem, there is still room for further improving its

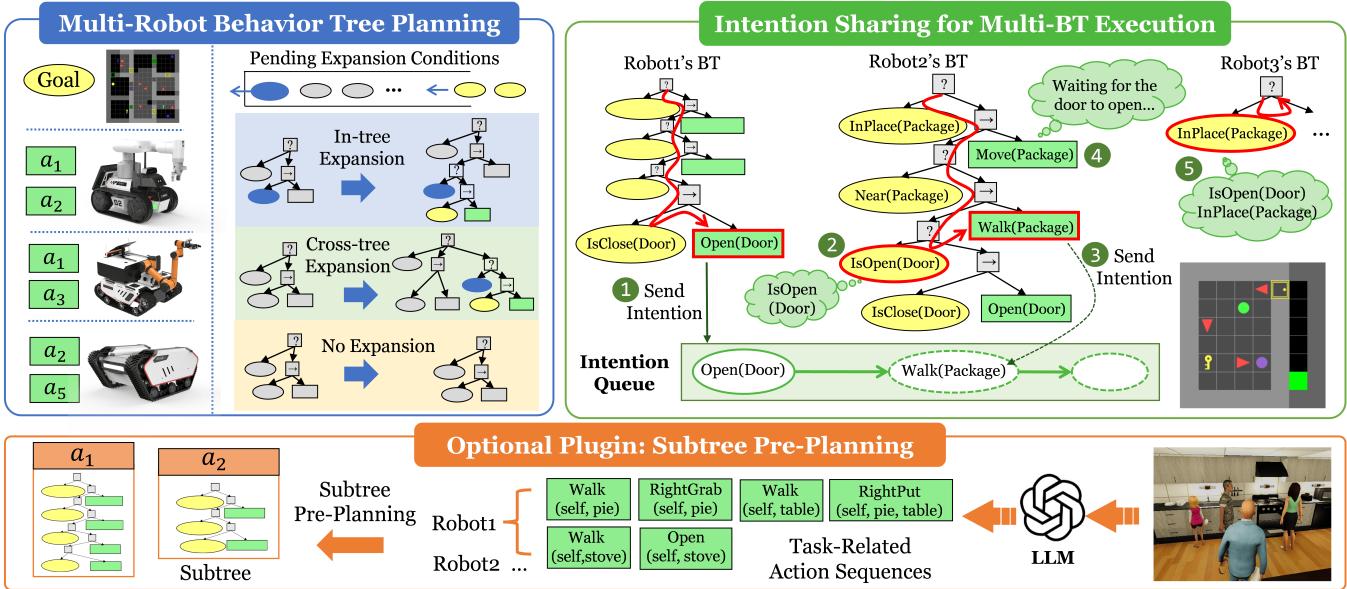


Figure 2: The framework of our paper. (1) MRBTP. A sound and complete algorithm for the multi-robot BT planning problem, capable of coordinating diverse actions across different BTs through cross-tree expansion. (2) Intention Sharing. Robots share intentions with each other during execution, enabling multi-BT parallelization without compromising failure tolerance. (3) Optional Plugin: Subtree Pre-planning. This plugin utilizes LLMs to pre-plan task-specific subtrees, establishing long-horizon action sequences to enhance MRBTP’s planning and execution efficiency.

efficiency. To achieve this, we begin by considering the following observations.

- During planning, the same tree structure might be generated multiple times, especially when multiple robots have overlapping action spaces.
- During execution, sharing the intentions of every short-horizon atomic action not only increases the communication overhead but is also ineffective for long-term task scheduling.

A natural idea is that if we can obtain some long-horizon actions for each robot that are beneficial to the task, which we call subtrees, and add these actions to the corresponding robot’s action space. During planning, we let these subtrees be prioritized over atomic actions, thereby speeding up the search for solutions and avoiding redundant planning. During execution, we only share the intentions of these subtrees. If the subtrees are well-designed, this approach can reduce communication overhead while also improving the efficiency of parallel execution.

Subtree Pre-planning Let’s first assume that we have obtained an action sequence $A = (a_1, a_2, \dots, a_m)$ for planning the subtree, and then consider how to use LLMs to generate task-related action sequences for each robot. Due to the modularity of the BT, we can treat the action sequence A as a long-horizon action, and its action model can be calculated:

$$pre(A) = \bigcup_{j=1}^m \left(pre(a_j) \setminus \bigcup_{k=1}^j add(a_k) \right) \quad (6)$$

$$add(A) = \bigcup_{j=1}^m (add(a_j) \setminus del(a_j)) - pre(A) \quad (7)$$

$$del(A) = \bigcup_{j=1}^m (del(a_j) \setminus add(a_j)) \quad (8)$$

We can obtain the tree structure of execution actions in A sequentially by running a single-robot BT planning algorithm, with constraints on the order of actions to be expanded, a process we call subtree pre-planning.

However, to make a subtree behave like an atomic action, i.e., not to exit the precondition of A while running in intermediate states, we need to introduce an additional subtree control structure, as illustrated in Figure 3. The subtree T_A has the preconditions `Close(Door)` and `Empty(Hand)`, but after `Get(Key)`, the `Empty(Hand)` condition is no longer satisfied. In the conventional BT planning algorithm, this would result in the subsequent actions not being ticked, causing the entire subtree T_A to fail. To address this issue, we introduce three subtree control nodes: `EnterSubtree`, `ExitSubtree`, and `RunningSubtree`. If $pre(A)$ is satisfied and the robot is not currently running this subtree T_A , then `EnterSubtree` will be executed. This action will change the status of the subtree to running. The `RunningSubtree` will return `S` until `ExitSubtree` is executed, or the BT begins executing a new action due to a change in the environment state. The parameter for the three nodes can be any identifier of the subtree. A simple way is to use the add effect $add(a_m)$ of the last action in the action sequence A as the identifier.

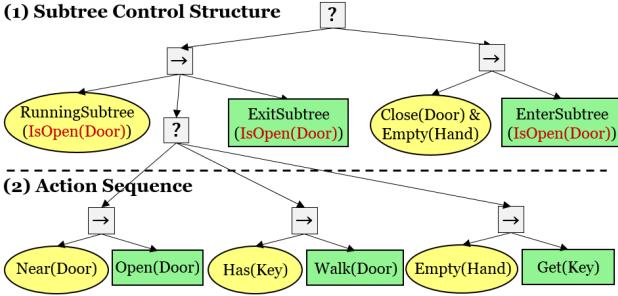


Figure 3: An example of a pre-planned subtree structure for open a door.

Prompt and Feedback for LLMs Appropriate subtrees for each robot can significantly improve the efficiency of planning and execution. However, obtaining these subtrees before planning is a very tricky task. Fortunately, pre-trained LLMs have been proven to possess task reasoning capabilities (Liu et al. 2023; Song et al. 2023). When such models are available, subtree pre-planning can be a highly effective plugin for MRBTP.

To obtain suitable action sequences for each robot from pre-trained LLMs, the model’s prompt should include: (1) task information, including the initial environment state and goal; (2) objects in the environment; (3) the action space for each robot; (4) few-shot demonstrations; and (5) a system prompt to guide the model to output correctly.

After the LLM produces an output, we designed a checker to automatically verify it. We provide feedback to the LLM in three cases: (1) the output has grammar errors; (2) the action sequences cannot be pre-planned into a subtree, meaning they are not coherent; (3) the number of action sequences generated for each robot is insufficient. Once the output is good enough or the maximum number of feedback attempts has been reached, we begin subtree pre-planning. After pre-planning is completed, we add each subtree to the action spaces of all robots that contain all of the actions in the subtree, to fully utilize the subtrees.

Experiments

We evaluate the performance of MRBTP in two simulated scenarios: (1) Warehouse management with coarse action granularity and a smaller action space, and (2) Home service with finer granularity and a larger action space. First, we assess the robustness of the MRBTP method under varying levels of homogeneity by introducing a failure probability for each action. Next, we conduct an ablation study on intention sharing to verify its contribution to the execution efficiency of multi-robot BTs. Then, given the finer action granularity in the home service scenario, we perform an ablation study to evaluate subtree pre-planning, examining the effectiveness of pre-trained LLMs in generating task-related action sequences and their impact on the overall efficiency of the MRBTP. All experiments were conducted on a system equipped with an AMD Ryzen 9 5900X 12-core processor with a 3.70 GHz base clock and 128 GB of DDR4 RAM.

Method	$\alpha = 1$			$\alpha \approx 0.5$	$\alpha = 0$
	SR(%)	TS	RS	SR(%)	SR(%)
BT-Expansion	100	8.8	33.8	12.4	4.6
MRBTP	100	5.8	15.3	100	100

Table 1: Performance comparison with baseline in warehouse management (4 robots, averaged over 500 trials).

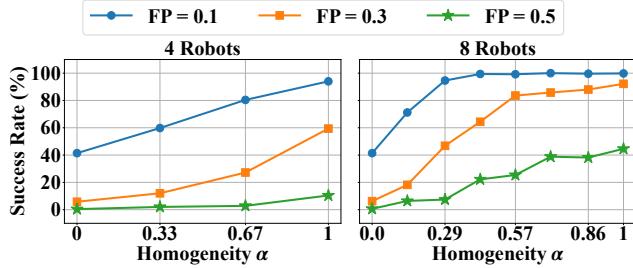
Experimental Setup

Scenarios (a) Warehouse Management. We extend the Minigrid (Chevalier-Boisvert et al. 2023) environment for multi-robot simulations with 4-8 robots in 4 rooms containing randomly placed packages. Robots have diverse action spaces, including room inspection and package relocation, with some possessing specialized capabilities or restricted access. The goal is to optimize warehouse space utilization. (b) Home Service. In the VirtualHome (Puig et al. 2018) environment, 2-4 robots interact with dozens of objects and perform hundreds of potential actions. Each robot’s action space is diverse, aiming to complete complex household tasks, such as setting the table or preparing a meal.

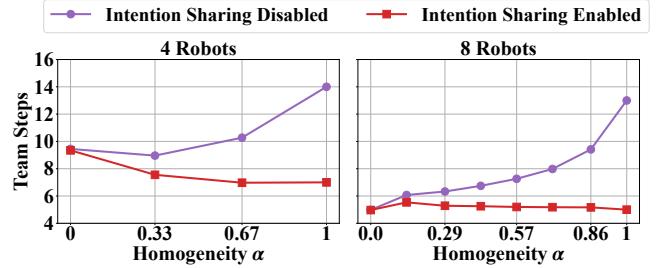
Evaluation Metrics The algorithm’s performance was evaluated using the following metrics: (a) Success Rate (SR): The percentage of successfully completed tasks across multiple trials, accounting for action failure probabilities. (b) Team Steps (TS): The total number of steps required for all robots to complete their tasks in parallel. (c) Total Robot Steps (RS): The sum of steps taken by each robot independently. (d) Communication Overhead (Comm.): The number of broadcast communications between robots due to intention sharing. (e) Number of Expanded Conditions (EC): The number of condition nodes expanded during the multi-robot BTs planning process, including those from subtree pre-planning if available. (f) Planning Time (PT): The time taken for multi-robot BT planning, including subtree pre-planning when available.

Settings (a) Homogeneity (α): The proportion of redundant actions assigned to robots, where $\alpha = 1$ denotes complete heterogeneity (no overlap in action spaces) and $\alpha = 0$ denotes complete homogeneity (identical action spaces). (b) Action Failure Probability (FP): The probability that a robot fails to execute an action. (c) Subtree Intention Sharing (Subtree IS) and Atomic Action Nodes Intention Sharing (Atomic IS): These terms refer to the application of Intention Sharing either among subtrees or at the level of individual atomic action nodes. (d) Feedback (F) and No Feedback (NF): This setting distinguishes between LLMs that use feedback during subtree generation and those that do not. In the Feedback condition, the LLM receives up to 3 feedback iterations, while in the No Feedback condition, no feedback is provided.

Baselines BT planning algorithms typically utilize action models for planning. To ensure consistency under the same problem assumptions, we propose directly adapting the BT-Expansion (Cai et al. 2021) algorithm, which has



(a) The impact of homogeneity on robustness



(b) The impact of intention sharing on execution efficiency

Figure 4: Comparison of success rate and team steps under different conditions for 4 and 8 robots. Each data point represents the average of 500 trials.

Homogeneity		$\alpha = 1$							$\alpha \approx 0.5$							$\alpha = 0$						
Subtree		✓							✓							✓						
Subtree IS		-							-							-						
Atomic IS		-							-							-						
TS	NF	161	159.4	114.4	109.6	78.9	79.8	139.7	137.5	126.2	119.6	86.9	102.5	73.7	68.5	96.1	94.8	75.6	78.0			
	F	-	-	116.7	114.2	77.1	79.16	-	-	124.6	126.0	80.8	96.2	-	-	107.1	106.4	70.4	76.8			
RS	NF	570.8	557.3	374	359.4	217.4	219.1	385.2	380.1	345.8	326.6	209.6	222	128.6	128.6	128.6	128.6	128.6	128.6			
	F	-	-	377	370.9	205.2	208	-	-	380.7	348.2	192.2	209	-	-	128.6	128.6	128.6	128.6			
Comm.	NF	0.0	63.8	0.0	7.1	6.7	14.1	0.0	43.5	0.0	8.0	7.1	20.9	0.0	15.2	0.0	2.8	6.5	9.3			
	F	-	-	0.0	4.8	6.6	12.2	-	-	0.0	4.4	6.4	13.2	-	-	0.0	0.7	5.2	6.0			

Table 2: Execution efficiency with subtree pre-planning and intention sharing.

Homogeneity	Subtree	Feedback	EC	PT (s)
$\alpha = 1$	-	-	8033.3	Timeout
	✓	-	998.1	12.4
	✓	✓	384.3	3.7
$\alpha \approx 0.5$	-	-	7882.5	Timeout
	✓	-	623.8	7.2
	✓	✓	267.9	2.6
$\alpha = 0$	-	-	2695.5	20.2
	✓	-	576.6	5.6
	✓	✓	146.8	1.4

Table 3: Planning efficiency with pre-planned subtrees. The average response time per LLM invocation is 4.2 seconds.

been proven sound and complete in single-robot settings, to multi-robot scenarios as our baseline. In BT-Expansion, each robot independently performs backward planning towards the team’s goal, without incorporating cross-tree expansion or intention sharing.

Experimental Results

Performance Comparison We randomly generated solvable multi-robot BT planning problems under various settings. Table 1 shows a significant drop in BT-Expansion’s success rate as homogeneity decreases. In contrast, MRBTP maintains a perfect success rate of 100% across all settings due to its cross-tree expansion. To avoid bias in exe-

cution efficiency (TS, RS) caused by planning failures, we only compared cases where both algorithms succeeded. Notably, even under full homogeneity, MRBTP outperforms BT-Expansion in execution efficiency due to intention sharing.

Robustness As shown in Figure 4a, the robustness of our algorithm improves with increasing homogeneity and is further enhanced by a larger number of robots. This improvement results from the increased likelihood of other robots compensating for action failures. Even with a 50% failure probability per action, the system retains approximately a 50% chance of achieving the goal with 8 robots and complete action space homogeneity.

Execution Efficiency As shown in Figure 4b, in fully heterogeneous scenarios, enabling intention sharing results in fewer team steps, indicating that our MRBTP algorithm inherently maintains superior execution efficiency under these conditions. Additionally, as homogeneity increases, the likelihood of robots performing redundant actions rises, reducing the probability of parallel task execution. However, with intention sharing, redundant actions are significantly minimized, preventing further efficiency loss. In this context, increased homogeneity brings more backup structures, further improving execution efficiency.

Effectiveness of Task-Specific Subtree Pre-Planning We constructed a dataset of 75 instances across three levels of homogeneity. The model used to generate subtrees

Models	No Feedback			Feedback		
	TS	RS	Comm.	TS	RS	Comm.
GPT-3.5-turbo	81.6	223.6	5.1	80.0	219.0	5.1
GPT-4o-mini	78.9	217.4	6.7	77.1	205.2	6.6
GPT-4o	77.4	200.9	6.3	74.9	190.7	6.3

Table 4: Execution efficiency across different LLMs under $\alpha = 1$ with subtree and subtree intention sharing.

is *gpt-4o-mini-2024-07-18* (OpenAI 2023). Table 2 provides a comparative analysis of the impact of introducing task-specific subtrees on execution efficiency, and communication overhead. The improvement in execution efficiency brought about by intention sharing increases with higher homogeneity. Execution efficiency is highest when subtrees are combined with intention sharing. During execution, the long-horizon subtrees facilitate forward planning, leading to more efficient and less frequent communication compared to finer-grained atomic actions. Table 3 shows that subtree pre-planning significantly reduces BTs planning time under a 60-second constraint by minimizing redundancy through subtree reuse and similar robot action spaces. Additionally, both Table 2 and Table 3 demonstrate that feedback effectively enhances both planning and execution efficiency, especially when integrated with the sharing of the subtree and intentions.

Execution Efficiency across Different LLMs We tested different versions of LLMs, including *gpt-3.5-turbo* (2024.12) and *gpt-4o-2024-08-06* (OpenAI 2023), for assisting in subtree pre-planning. As shown in Table 4, with the increased reasoning capability of the LLMs, there is a slight improvement in execution efficiency, while communication overhead remains largely unchanged. This can be attributed to the fact that subtree pre-planning becomes more appropriate and effective as the model’s reasoning ability improves. Additionally, the results further demonstrate that the feedback mechanism enhances execution efficiency across all LLMs.

Related Work

BT Planning. Many works have focused on automatically generating BTs to perform tasks, such as evolutionary computing (Neupane and Goodrich 2019; Colledanchise, Parasuraman, and Ögren 2019; Lim, Baumgarten, and Colton 2010), reinforcement learning (Banerjee 2018; Pereira and Engel 2015), imitation learning (French et al. 2019), MCTS (Scheide, Best, and Hollinger 2021), and formal synthesis (Li et al. 2021; Tadewos, Newaz, and Karimoddini 2022; Neupane and Goodrich 2023). Recently, some works directly generate BTs using LLMs (Lykov and Tsetserukou 2023; Lykov et al. 2023). However, the above methods either require complex environment modeling or cannot guarantee the reliability of BTs. In contrast, BT planning (Cai et al. 2021; Chen et al. 2024a) based on STRIPS-style modeling (Fikes and Nilsson 1971) not only offers intuitive environment modeling but also ensures the reliability and robustness

of the generated BTs.

BT in MRS. BT generation for Multi-Robot Systems (MRS) has been investigated using various methodologies. Evolutionary computing (Neupane and Goodrich 2019) is a general heuristic search method applied to BT generation in MRS. While versatile, this approach often suffers from slow search efficiency due to its lack of integration with the action model. Given the modular nature of BT systems, the action model is not difficult to obtain (Arora et al. 2018), enabling the development of methods that can yield more efficient solutions. MRS BT generation methods based on LLMs (Lykov et al. 2023) or other machine learning techniques (Fu, Qin, and Yin 2016) have also been explored. These methods require substantial training data, making data collection and model training resource-intensive. Moreover, the aforementioned methods lack guarantees for the completeness and correctness of the generated BTs. Auction-based methods (Dahlquist et al. 2023; Heppner et al. 2024; Colledanchise et al. 2016), some of which incorporate action model planning, rely on the assumption of reliable communication and low transmission delay to ensure efficient task completion. However, such conditions are not always guaranteed, rendering these approaches less robust in environments with unreliable communication. In contrast, our method generates BTs before the robot team begins execution, ensuring task completion even in the absence of communication during execution. Communication during execution serves only to improve coordination efficiency, rather than being a necessary assumption.

LLM for Task Reasoning. Recently, significant progress has been made in using LLMs for task reasoning (Song et al. 2023; Liu et al. 2023; Ahn et al. 2022; Chen et al. 2023), such as progprompt (Singh et al. 2022), PlanBench (Valmeekam et al. 2023), and Voyager (Wang et al. 2023). Furthermore, the LLM has shown the ability to decompose the task into subgoals (Gao et al. 2024; Singh, Traum, and Thomason 2024), which is closely related to our subtree pre-planning for multi-robot BT planning. As the task reasoning abilities of LLMs continue to evolve and strengthen, our subtree pre-planning technique is poised to become increasingly relevant and effective.

Conclusion

We propose MRBTP, the first sound and complete algorithm for solving the multi-robot BT planning problem. The cross-tree expansion coordinates BTs for achieving goals, while intention sharing improves execution efficiency and robustness. The LLM plugin further enhances planning speed and reduces communication overhead. These contributions represent a key step forward in scalable, reliable multi-robot systems. Future research will refine the algorithm’s performance and extend its application to more complex, dynamic environments, solidifying MRBTP as a foundational approach in multi-robot planning. Furthermore, the potential deployment of the algorithm on actual robotic systems will be explored, evaluating its effectiveness, scalability, and practicality in real-world scenarios.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (Grant Nos. 62106278, 62032024).

References

- Ahn, M.; Brohan, A.; Brown, N.; Chebotar, Y.; Cortes, O.; David, B.; Finn, C.; Fu, C.; Gopalakrishnan, K.; Hausman, K.; et al. 2022. Do as i Can, Not as i Say: Grounding Language in Robotic Affordances. *arXiv preprint arXiv:2204.01691*.
- Arora, A.; Fiorino, H.; Pellier, D.; Métivier, M.; and Pesty, S. 2018. A review of learning planning action models. *The Knowledge Engineering Review*, 33: e20.
- Banerjee, B. 2018. Autonomous Acquisition of Behavior Trees for Robot Control. 3460–3467.
- Cai, Z.; Li, M.; Huang, W.; and Yang, W. 2021. BT Expansion: A Sound and Complete Algorithm for Behavior Planning of Intelligent Robots with Behavior Trees. 6058–6065. AAAI Press.
- Chen, X.; Cai, Y.; Mao, Y.; Li, M.; Yang, W.; Xu, W.; and Wang, J. 2024a. Integrating Intent Understanding and Optimal Behavior Planning for Behavior Tree Generation from Human Instructions.
- Chen, X.; Cai, Y.; Mao, Y.; Li, M.; Yang, Z.; Shanghua, W.; Yang, W.; Xu, W.; and Wang, J. 2024b. Efficient Behavior Tree Planning with Commonsense Pruning and Heuristic. *arXiv preprint arXiv:2406.00965*.
- Chen, Y.; Cui, W.; Chen, Y.; Tan, M.; Zhang, X.; Zhao, D.; and Wang, H. 2023. RoboGPT: An Intelligent Agent of Making Embodied Long-Term Decisions for Daily Instruction Tasks. *arXiv preprint arXiv:2311.15649*.
- Chevalier-Boisvert, M.; Dai, B.; Towers, M.; de Lazcano, R.; Willems, L.; Lahlou, S.; Pal, S.; Castro, P. S.; and Terry, J. 2023. Minigrid & Miniworld: Modular & Customizable Reinforcement Learning Environments for Goal-Oriented Tasks. *CoRR*, abs/2306.13831.
- Colledanchise, M.; Almeida, D.; and Ogren, P. 2019. Towards Blended Reactive Planning and Acting Using Behavior Trees. 8839–8845. Montreal, QC, Canada: IEEE. ISBN 978-1-5386-6027-0.
- Colledanchise, M.; Marzinotto, A.; Dimarogonas, D. V.; and Oegren, P. 2016. The Advantages of Using Behavior Trees in Multi-Robot Systems. In *Proceedings of ISR 2016: 47st International Symposium on Robotics*, 1–8. VDE.
- Colledanchise, M.; and Ögren, P. 2018. *Behavior Trees in Robotics and AI: An Introduction*. CRC Press.
- Colledanchise, M.; Parasuraman, R.; and Ögren, P. 2019. Learning of Behavior Trees for Autonomous Agents. *IEEE Transactions on Games*, 11(2): 183–189.
- Dahlquist, N.; Lindqvist, B.; Saradagi, A.; and Nikolakopoulos, G. 2023. Reactive Multi-agent Coordination Using Auction-based Task Allocation and Behavior Trees. In *2023 IEEE Conference on Control Technology and Applications (CCTA)*, 829–834. IEEE.
- Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. 2(3-4): 189–208.
- French, K.; Wu, S.; Pan, T.; Zhou, Z.; and Jenkins, O. C. 2019. Learning Behavior Trees from Demonstration. In *2019 International Conference on Robotics and Automation (ICRA)*, 7791–7797. IEEE.
- Fu, Y.; Qin, L.; and Yin, Q. 2016. A reinforcement learning behavior tree framework for game AI. In *2016 International Conference on Economics, Social Science, Arts, Education and Management Engineering*, 573–579. Atlantis Press.
- Gao, Z.; Mu, Y.; Qu, J.; Hu, M.; Guo, L.; Luo, P.; and Lu, Y. 2024. DAG-Plan: Generating Directed Acyclic Dependency Graphs for Dual-Arm Cooperative Planning. *arXiv preprint arXiv:2406.09953*.
- Heppner, G.; Oberacker, D.; Roennau, A.; and Dillmann, R. 2024. Behavior Tree Capabilities for Dynamic Multi-Robot Task Allocation with Heterogeneous Robot Teams. *arXiv preprint arXiv:2402.02833*.
- Heppnerl, G.; Berg, N.; Oberacker, D.; Spielbauer, N.; Roennau, A.; and Dillmann, R. 2023. Distributed Behavior Trees for Heterogeneous Robot Teams. In *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*, 1–8. IEEE.
- Li, S.; Park, D.; Sung, Y.; Shah, J. A.; and Roy, N. 2021. Reactive Task and Motion Planning under Temporal Logic Specifications. 12618–12624. IEEE.
- Lim, C.-U.; Baumgarten, R.; and Colton, S. 2010. Evolving Behaviour Trees for the Commercial Game DEFCON. In *Applications of Evolutionary Computation: EvoApplications 2010: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Istanbul, Turkey, April 7-9, 2010, Proceedings, Part I*, 100–110. Springer.
- Liu, B.; Jiang, Y.; Zhang, X.; Liu, Q.; Zhang, S.; Biswas, J.; and Stone, P. 2023. LLM+P: Empowering Large Language Models with Optimal Planning Proficiency. *arXiv*.
- Lykov, A.; Dronova, M.; Naglov, N.; Litvinov, M.; Satsevich, S.; Bazhenov, A.; Berman, V.; Shcherbak, A.; and Tsetserukou, D. 2023. LLM-MARS: Large Language Model for Behavior Tree Generation and NLP-enhanced Dialogue in Multi-Agent Robot Systems. *arXiv preprint arXiv:2312.09348*.
- Lykov, A.; and Tsetserukou, D. 2023. LLM-BRAIn: AI-driven Fast Generation of Robot Behaviour Tree Based on Large Language Model. *arXiv preprint arXiv:2305.19352*.
- Neupane, A.; and Goodrich, M. 2019. Learning Swarm Behaviors Using Grammatical Evolution and Behavior Trees. 513–520. IJCAI Organization.
- Neupane, A.; and Goodrich, M. A. 2023. Designing Behavior Trees from Goal-Oriented LTLf Formulas. *arXiv preprint arXiv:2307.06399*.
- OpenAI. 2023. GPT-4 Technical Report. *ArXiv*.
- Pereira, R. d. P.; and Engel, P. M. 2015. A Framework for Constrained and Adaptive Behavior-Based Agents. *arXiv preprint arXiv:1506.02312*.

Puig, X.; Ra, K.; Boben, M.; Li, J.; Wang, T.; Fidler, S.; and Torralba, A. 2018. Virtualhome: Simulating Household Activities via Programs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 8494–8502.

Scheide, E.; Best, G.; and Hollinger, G. A. 2021. Behavior Tree Learning for Robotic Task Planning through Monte Carlo DAG Search over a Formal Grammar. 4837–4843. Xi'an, China: IEEE. ISBN 978-1-72819-077-8.

Singh, I.; Blukis, V.; Mousavian, A.; Goyal, A.; Xu, D.; Tremblay, J.; Fox, D.; Thomason, J.; and Garg, A. 2022. ProgPrompt: Generating Situated Robot Task Plans Using Large Language Models.

Singh, I.; Traum, D.; and Thomason, J. 2024. Twostep: Multi-agent Task Planning Using Classical Planners and Large Language Models. *arXiv preprint arXiv:2403.17246*.

Song, C. H.; Wu, J.; Washington, C.; Sadler, B. M.; Chao, W.-L.; and Su, Y. 2023. LLM-Planner: Few-Shot Grounded Planning for Embodied Agents with Large Language Models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2998–3009.

Tadewos, T. G.; Newaz, A. A. R.; and Karimoddini, A. 2022. Specification-Guided Behavior Tree Synthesis and Execution for Coordination of Autonomous Systems. *Expert Systems with Applications*, 201: 117022.

Valmeekam, K.; Marquez, M.; Olmo, A.; Sreedharan, S.; and Kambhampati, S. 2023. PlanBench: An Extensible Benchmark for Evaluating Large Language Models on Planning and Reasoning about Change. In Oh, A.; Naumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; and Levine, S., eds., *Advances in Neural Information Processing Systems*, volume 36, 38975–38987. Curran Associates, Inc.

Wang, G.; Xie, Y.; Jiang, Y.; Mandlekar, A.; Xiao, C.; Zhu, Y.; Fan, L.; and Anandkumar, A. 2023. Voyager: An Open-Ended Embodied Agent with Large Language Models. In *NeurIPS 2023 Workshop IMOL*.

Appendix

A. Proofs of MRBTP

In this section, we present a comprehensive version of the formalizations and proofs for Multi-Robot Behavior Tree Planning (MRBTP).

Definitions

Definition 3 (Behavior Tree). A behavior tree (BT) is a three-tuple $\mathcal{T} = \langle f, r, \Delta t \rangle$. $f : 2^n \rightarrow 2^n$ is its effect on the environment state, Δt is the time step, and $r : 2^n \mapsto \{S, R, F\}$ partitions states into three regions, where \mathcal{T} returns success, running, failure, respectively.

In BT planning for a single robot (Cai et al. 2021), we represent the problem as: $\langle \mathcal{S}, \mathcal{L}, \mathcal{A}, \mathcal{M}, s_0, g \rangle$, where \mathcal{S} is the finite set of environment states, \mathcal{L} is the finite set of literals that form states, \mathcal{A} is the finite set of actions, \mathcal{M} is the action model, s_0 is the initial state, g is the goal condition.

A condition c in BT is usually a subset of a state s . If $c \subseteq s$, it is said condition c holds in that state s . The state transition affected by action $a \in \mathcal{A}$ can be defined as a triplet $\mathcal{M}(a) = \langle \text{pre}(a), \text{add}(a), \text{del}(a) \rangle$, comprising the precondition, add effects, and delete effects of the action. We assume that an action always finishes in finite time. If a is finished after k time step, the subsequent state $s_{t'}$ will be:

$$s_{t'} = f_a(s_t) = s_t \cup \text{add}(a) \setminus \text{del}(a), t' = t + k \quad (9)$$

The following property holds for $\forall a \in \mathcal{A}$:

$$\text{add}(a) \cap \text{del}(a) = \emptyset \quad (10)$$

$$\text{add}(a) \cap \text{pre}(a) = \emptyset \quad (11)$$

We then extend the BT representation from a single robot to a multi-robot system.

Definition 4 (Multi-BT System). A n -robot BT system is a four-tuple $\langle \Phi, f_\Phi, r_\Phi, \Delta t_\Phi \rangle$, where $\Phi = \{\mathcal{T}_i\}_{i=1}^n$ is the set of BTs, $f_\Phi : \mathcal{S} \mapsto \mathcal{S}$ is the team state transition function, Δt_Φ is the team time step, $r_\Phi : \mathcal{S} \mapsto \{S, R, F\}$ is the team region partition.

Due to variability in hardware performance, we allow each robot's BT to have a different response frequency, with Δt_Φ representing the common minimum response interval. The state transition can be calculated as follows:

$$s_{t+\Delta t_\Phi} = f_\Phi(s_t) = s_t \cup \bigcup_{i=1}^n (\text{add}(a_i) \setminus \text{del}(a_i)) \quad (12)$$

where a_i is the action of robot i in time t . If robot i do not have an action or its action is running, we let $\text{add}(a_i) = \text{del}(a_i) = \emptyset$.

The team region partition can be calculated as follows:

$$r_\Phi(s) = \begin{cases} R & \text{if } \exists i, r_i(s) = R \\ S & \text{if } \forall i, r_i(s) \neq R \text{ and } \exists i, r_i(s) = S \\ F & \text{if } \forall i, r_i(s) = F \end{cases} \quad (13)$$

The status of Φ is R if any BT is still running, S if some BT returns success and no one is running, and F if all BT fails.

Definition 5 (Finite Time Successful). Φ is finite time successful (FTS) from region of attraction (ROA) R to condition c , if $\forall s_0 \in R$ there is a finite time τ such that for any $t < \tau$, $r_\Phi(s_t) = R$, and for any $t \geq \tau$, $r_\Phi(s_t) = S$, $c \subseteq s_t$.

With definitions above, the multi-robot BT planning problem can finally be defined.

Problem 2 (Multi-Robot BT Planning). The problem is a tuple $\langle \mathcal{S}, \mathcal{L}, \{\mathcal{A}_i\}_{i=1}^n, \mathcal{M}, s_0, g \rangle$, where \mathcal{S} is the finite set of environment states, \mathcal{L} is the finite set literals that form states and conditions, \mathcal{A}_i is the finite action set of robot i , \mathcal{M} is the action model, s_0 is the initial state, g is the goal condition. A solution to this problem is a BT set $\Phi = \{\mathcal{T}_i\}_{i=1}^n$ built with $\{\mathcal{A}_i\}_{i=1}^n$, such that Φ is FTS from $R \ni s_0$ to g .

Propositions and Proofs

Lemma 1. Given a condition c , the sequence structure $\mathcal{T}_a = \text{sequence}(c_a, a)$ expanded in Algorithm 3 (line 7) is FTS from $\mathcal{S}_a = \{s \in \mathcal{S} | c_a \subseteq s\}$ to c .

Proof. Starting from any $s_t \in \mathcal{S}_a = \{s \in \mathcal{S} | c_a \subseteq s\}$, $\exists c_a, c_a \subseteq s_t$. According to Equation 9, there exists a finite k such that the action returns success and $s_{t+k} = s_t \cup \text{add}(a) \setminus \text{del}(a) \supseteq c_a \cup \text{add}(a) \setminus \text{del}(a) = \text{pre}(a) \cup c \setminus \text{del}(a)$. Since action selection (line 5) ensures that $c \setminus \text{del}(a) = c$, we have $s_{t+k} \supseteq \text{pre}(a) \cup c \supseteq c$. Therefore \mathcal{T}_a is FTS from \mathcal{S}_a to c . \square

Lemma 2. Given a condition c and $\mathcal{C}_{\text{new}} \neq \emptyset$, \mathcal{T}_{new} expanded in Algorithm 3 (line 8) is FTS from $\mathcal{S}_{\text{new}} = \{s \in \mathcal{S} | c_a \subseteq s, c_a \in \mathcal{C}_{\text{new}}\}$ to c .

Algorithm 3: One-step cross-tree expansion

```

1: function ExpandOneRobot ( $\mathcal{T}, \mathcal{A}, c$ )
2:    $\mathcal{T}_{new} \leftarrow c$                                  $\triangleright$  newly expanded subtree
3:    $\mathcal{C}_{new} \leftarrow \emptyset$                        $\triangleright$  newly expanded conditions
4:   for each action  $a \in \mathcal{A}$  do
5:     if  $c \cap (pre(a) \cup add(a) \setminus del(a)) \neq \emptyset$  and  $c \setminus del(a) = c$  then
6:        $c_a \leftarrow pre(a) \cup c \setminus add(a)$ 
7:        $\mathcal{T}_a \leftarrow Sequence(c_a, a)$ 
8:        $\mathcal{T}_{new} \leftarrow Fallback(\mathcal{T}_{new}, \mathcal{T}_a)$ 
9:        $\mathcal{C}_{new} \leftarrow \mathcal{C}_{new} \cup \{c_a\}$ 
10:    if  $\mathcal{C}_{new} \neq \emptyset$  then
11:      if ConditionInTree( $c, \mathcal{T}$ ) then
12:        Replace  $c$  with  $\mathcal{T}_{new}$  in  $\mathcal{T}$                                  $\triangleright$  in-tree expand
13:      else if  $\mathcal{T}_{new} \neq c$  then
14:         $\mathcal{T} \leftarrow Fallback(\mathcal{T}, \mathcal{T}_{new})$                                  $\triangleright$  cross-tree expand
15:    return  $\mathcal{T}, \mathcal{C}_{new}$ 

```

Proof. Since $\mathcal{C}_{new} \neq \emptyset$, \mathcal{T}_{new} has and only has one fallback node as the root, given $Fallback(\mathcal{T}_1, Fallback(\mathcal{T}_2, \mathcal{T}_3)) = Fallback(\mathcal{T}_1, \mathcal{T}_2, \mathcal{T}_3)$. The fallback node returns success as long as any child returns success. According to Lemma 1, $\forall c_a \in \mathcal{C}_{new}$, the corresponding \mathcal{T}_a is FTS from \mathcal{S}_a to c , and \mathcal{C}_{new} is a finite set as \mathcal{A} is a finite set. Therefore \mathcal{T}_{new} is FTS from $\mathcal{S}_{new} = \bigcup_{c_a \in \mathcal{C}_{new}} \mathcal{S}_a = \{s \in \mathcal{S} | c_a \subseteq s, c_a \in \mathcal{C}_{new}\}$ to c . \square

Proposition 6. Given \mathcal{T} is FTS from R to g , if \mathcal{T} is expanded by Algorithm 3 to \mathcal{T}' given c , c is in \mathcal{T} and $\mathcal{C}_{new} \neq \emptyset$, then \mathcal{T}' is FTS from $R' = R \cup \{s \in \mathcal{S} | c_a \subseteq s, c_a \in \mathcal{C}_{new}\}$ to g .

Proof. For any $s \in R'$, we consider two cases:

(1) $s \in R$. In this case, either \mathcal{T}_{new} in the BT \mathcal{T}' is not ticked ($c \not\subseteq s$), or \mathcal{T}_{new} returns success ($c \subseteq s$). In either situation, the execution logic of the BT is the same as that of the original \mathcal{T} . Since \mathcal{T} is FTS from R to g , the expanded \mathcal{T}' is also FTS from R to g .

(2) $s \notin R$ and $s \in \mathcal{S}_{new} = \{s \in \mathcal{S} | c_a \subseteq s, c_a \in \mathcal{C}_{new}\}$. In this case, the expended \mathcal{T}_{new} in BT \mathcal{T}' will be ticked. According to Lemma 2, \mathcal{T}_{new} is FTS from \mathcal{S}_{new} to c , which means there is a finite time τ_1 such that for any $t < \tau_1$, $r_{\mathcal{T}'}(s_t) = R$, and for $t = \tau_1$, $r_{\mathcal{T}'}(s_{\tau_1}) = S$, $c \subseteq s_{\tau_1}$. Since $s_{\tau_1} \in R$, there is a finite time $t = \tau_2$ such that for any $t < \tau_1 + \tau_2$, $r_{\mathcal{T}'}(s_t) = R$, and for any $t \geq \tau_1 + \tau_2$, $r_{\mathcal{T}'}(s_t) = S$, $g \subseteq s_t$. Therefore \mathcal{T}' is FTS from $\mathcal{S}_{new} \setminus R$ to g .

According (1) and (2), \mathcal{T}' is FTS from $R' = R \cup \{s \in \mathcal{S} | c_a \subseteq s, c_a \in \mathcal{C}_{new}\}$ to g . \square

Proposition 7. If \mathcal{T} is expanded by Algorithm 3 to \mathcal{T}' given c , c is not in \mathcal{T} and $\mathcal{C}_{new} \neq \emptyset$, then \mathcal{T}' is FTS from $\mathcal{S}_{new} = \{s \in \mathcal{S} | c_a \subseteq s, c_a \in \mathcal{C}_{new}\}$ to c .

Proof. Because c is not in \mathcal{T} , the expanded subtree \mathcal{T}_{new} is added to the tail of the root fallback node of \mathcal{T} (line 14). $\forall s \in \mathcal{S}_{new}$, \mathcal{T}_{new} in \mathcal{T}' will be ticked. According to Lemma 2, \mathcal{T}_{new} is FTS from \mathcal{S}_{new} to c , which means \mathcal{T}' is also FTS from \mathcal{S}_{new} to c . \square

Lemma 3. After the one-step expansion of the multi-BT system $\Phi = \{\mathcal{T}_i\}_{i=1}^n$ with respect to condition c , let $\Phi_{new} = \{\mathcal{T}_{i,new}\}_{i=1}^n$ represent the set of all extended subtrees. Φ_{new} is FTS from $\bigcup_{i=1}^n \mathcal{S}_{i,new}$ to c , where $\mathcal{S}_{i,new} = \{s \in \mathcal{S} | c_a \subseteq s, c_a \in C_{i,new}\}$.

Proof. According to Lemma 2, $\forall \mathcal{T}_{i,new} \in \Phi_{new}$, $\mathcal{T}_{i,new}$ is FTS from $\mathcal{S}_{i,new}$ to c . Therefore, for any $s \in \bigcup_{i=1}^n \mathcal{S}_{i,new}$, at least one subtree can run in s . Since we assume that at each time, only one robot with the highest priority can execute an action if its precondition is satisfied, we designate the executed subtree as $\mathcal{T}_{j,new}$. Because $\mathcal{T}_{j,new}$ is FTS to c , we have that Φ_{new} is FTS from $\bigcup_{i=1}^n \mathcal{S}_{i,new}$ to c . \square

Proposition 8. : After the k -th ($k \geq 1$) iteration of the while loop in Algorithm 4, where the explored condition is c^k , $\Phi^k = \{\mathcal{T}_i^k\}_{i=1}^n$ is FTS from ROA $R^k = R^{k-1} \cup \bigcup_{i=1}^n \mathcal{S}_i^k$ to goal g , where $\mathcal{S}_i^k = \{s \in \mathcal{S} | c_a \subseteq s, c_a \in C_{i,new}^k\}$.

Proof. This proposition can be proved by strong induction. In the basis step (before the first expansion), all the BT $\mathcal{T}_i^0 \in \Phi^0$ is $Fallback(g)$ (line 4). According to 13, $\Phi^0 = \{\mathcal{T}_i^0\}_{i=1}^n$ if FTS from $R^0 = \{s \in \mathcal{S} | g \subseteq s\}$ to g .

For the inductive step, we assume after the k -th ($k \geq 1$) iteration of the while loop, Φ^k is FTS from R^k to g . Then after the $(k+1)$ -th iteration, for $s \in R^k \cup \bigcup_{i=1}^n \mathcal{S}_i^{k+1}$, there are two cases:

Algorithm 4: MABTP

Input: problem $\langle \mathcal{S}, \mathcal{L}, \{\mathcal{A}_i\}_{i=1}^n, \mathcal{M}, s_0, c \rangle$

Output: solution $\Phi = \{\mathcal{T}_i\}_{i=1}^n$

```

1:  $\mathcal{C}_U \leftarrow \{g\}$                                  $\triangleright$  conditions to be explored
2:  $\mathcal{C}_E \leftarrow \emptyset$                              $\triangleright$  expanded conditions
3: for  $i = 1$  to  $n$  do                                 $\triangleright$  init the BTs
4:    $\mathcal{T}_i \leftarrow \text{Fallback}(g)$ 
5: while  $\mathcal{C}_U \neq \emptyset$  do                                 $\triangleright$  explore  $c$ 
6:    $c \leftarrow \text{Pop}(\mathcal{C}_U)$                           $\triangleright$  prune
7:   if  $\text{HasSubSet}(c, \mathcal{C}_E)$  then continue
8:   for  $i = 1$  to  $n$  do                                 $\triangleright$  return a solution
9:      $\mathcal{T}_i, \mathcal{C}_{new} \leftarrow \text{ExpandOneRobot}(\mathcal{T}_i, \mathcal{A}_i, c)$ 
10:    if  $\text{HasSubSet}(s_0, \mathcal{C}_{new})$  then
11:      return  $\Phi = \{\mathcal{T}_i\}_{i=1}^n$ 
12:     $\mathcal{C}_E \leftarrow \mathcal{C}_E \cup \mathcal{C}_{new}$                                  $\triangleright$  add new conditions
13:     $\mathcal{C}_U \leftarrow \mathcal{C}_U \cup \mathcal{C}_{new}$ 
14: return Unsolvable

```

(1) $s \in R^k$. In this case, Φ^{k+1} has the same execution logic as Φ^k , just similar to case (1) in Proposition 6. Therefore Φ^{k+1} is FTS from R^k to g .

(2) $s \notin R^k$ and $s \in \bigcup_{i=1}^n \mathcal{S}_i^{k+1}$. In this case, one of the expended $\mathcal{T}_{i,new}^{k+1} \in \Phi_{new}^{k+1}$ will be ticked and the state will go to some $s' \supseteq c$ in finite time according to Lemma 3. Since $s' \in R^k$, Φ^{k+1} will succeed to g in finite time according to (1).

According (1) and (2), after the k -th iteration, Φ^k is FTS from ROA $R^k = R^{k-1} \cup \bigcup_{i=1}^n \mathcal{S}_i^k$ to goal g . \square

Proposition 9. Algorithm ?? is sound, i.e. if it returns a result Φ rather than *Unsolvable*, then Φ is a solution of Problem 1.

Proof. If Algorithm ?? returns a result Φ^k after k -th iteration, that means $\exists c \in \mathcal{C}_{new}^k, c \subseteq s_0$ (line 10). So s_0 is in the ROA of Φ^k : $R^k = R^{k-1} \cup \bigcup_{i=1}^n \mathcal{S}_i^k$. According to Proposition 8, Φ^k is FTS from R^k to g . That makes Φ^k a solution of Problem 2. \square

Lemma 4. Algorithm ?? terminates in finite time.

Proof. This is because the total number of states $|\mathcal{S}|$ is finite, therefore the number of conditions is also finite. The algorithm either returns prematurely (line 11) or terminates when $\mathcal{C}_U = \emptyset$ (line 14). Within the while loop, the loop for one-step expansion also terminates in finite time because the total number of actions $|\{\mathcal{A}_i\}_{i=1}^n|$ is finite. Therefore, Algorithm ?? terminates in finite time. \square

Proposition 10. Algorithm ?? is complete, i.e., if Problem 2 is solvable, the algorithm returns a Φ which is a solution.

Proof. According to Lemma 4, Algorithm ?? either returns prematurely or terminates when $\mathcal{C}_U = \emptyset$ in finite time.

If it returns a Φ prematurely, according to Proposition 9, Φ is a solution of Problem 2.

If it terminates when $\mathcal{C}_U = \emptyset$, it implies that all conditions reachable from the goal condition g have been explored, but no viable path from the goal g to the initial state s_0 has been found. This suggests that Problem 2 is not solvable.

This can be proved by contradiction. We assume that there exists a solution Φ but our algorithm fails. Φ is FTS from $R \ni s_0$ to g , so we can utilize a finite sequence to represent the state transition of the solution BT: $(s_0, a_1, s_1, a_2, \dots, s_{m-1}, a_m, s_m)$ where the goal condition $g \subseteq s_m$. We can prove, via mathematical induction from s_m to s_0 in this sequence, that every state in this sequence must satisfy some condition that was traversed during the while loop.

For the basis step, we consider the state s_{m-k} with $k = 0$, i.e. $s_m \supseteq g$. Obviously g is in all BTs of Φ .

For the inductive step, we consider a transition $(s_{m-k-1}, a_{m-k}, s_{m-k})$ and there exists a condition $c \subseteq s_{m-k}$ in some BT $\mathcal{T}_i \in \Phi$ as the inductive premise. We need to prove that there also exists a condition satisfied by s_{m-k-1} . The inductive premise and the transition gives:

$$s_{m-k-1} \cup \text{add}(a_{m-k}) \setminus \text{del}(a_{m-k}) = s_{m-k} \supseteq c \quad (14)$$

which can be deduced to:

$$s_{m-k-1} \supseteq c \setminus \text{add}(a_{m-k}) \quad (15)$$

We then analyze in two cases, whether $c \cap (\text{pre}(a_{m-k}) \cup \text{add}(a_{m-k}) \setminus \text{del}(a_{m-k})) = \emptyset$.

(1) $c \cap (\text{pre}(a_{m-k}) \cup \text{add}(a_{m-k}) \setminus \text{del}(a_{m-k})) = \emptyset$: From the case premise, $c \cap \text{add}(a_{m-k}) = \emptyset$ holds because the add effects and delete effects has no common member (Equation 10). Therefore Equation 15 can further be deduced to $s_{m-k-1} \supseteq c$, which means s_{m-k-1} satisfies the condition c of the BT.

(2) $c \cap (pre(a_{m-k}) \cup add(a_{m-k}) \setminus del(a_{m-k})) \neq \emptyset$: The transition shown by eqn. 14 indicate $s_{m-k} \setminus del(a_{m-k}) = s_{m-k}$. Therefore when expanding c , action a_{m-k} will be explored because the selective condition holds (line 5), which creates a condition node $c_{a_{m-k}} = pre(a_{m-k}) \cup c \setminus add(a_{m-k})$. The state transition also provides that $s_{m-k-1} \supseteq pre(a_{m-k})$, then with Equation 15 we can deduce:

$$s_{m-k-1} \supseteq pre(a_{m-k}) \cup c \setminus add(a_{m-k}) \quad (16)$$

Therefore s_{m-k-1} satisfies the condition $c_{a_{m-k}}$. Assume a_{m-k} is in the action space of robot i , $a_{m-k} \in \mathcal{A}_i$, then $c_{a_{m-k}}$ will be expanded to its BT $\mathcal{T}_i \in \Phi$.

The two cases complete the proof of the inductive step.

The mathematical induction proves that any state in the sequence must satisfy some condition node expanded in the while-loop. Therefore s_0 will satisfy some condition node $c \in \mathcal{C}_{new}$ after one-step expansion, then the algorithm will return a Φ , which contradicts the initial assumption that a solution Φ exists but our algorithm fails.

Therefore if Problem 2 is solvable, the algorithm returns a solution Φ . \square

B.Limitations and Future Work

Assumption

As mentioned above, we assume that at each time, only one robot with the highest priority can execute an action if its precondition is satisfied. Otherwise, two exceptions may occur:

(1) Deadlocks. For example, at time t , the robot 1 takes an action a_1 that changes the state from s_1 to s_2 , but the robot 2 then takes an action a_2 and changes s_2 back to s_1 . This loop will continue indefinitely.

(2) Departures from the ROA. For example, the state is $\{1, 2\}$ at time t , the robot 1 attempts to take a_1 and the robot 2 attempts to take a_2 . The action models are:

$$\begin{aligned} pre(a_1) &= \{1, 2\}, add(a_1) = \{3\}, del(a_1) = \{1\} \\ pre(a_2) &= \{1, 2\}, add(a_2) = \{4\}, del(a_2) = \{2\} \end{aligned}$$

If only a_1 is applied, the state will transfer to $\{2, 3\}$. If only a_2 is applied, the state will transfer to $\{1, 4\}$. Both $\{2, 3\}$ and $\{1, 4\}$ is in the ROA of Φ . However, if a_1 and a_2 are applied simultaneously, the state will transfer to $\{3, 4\}$, which is out of the ROA.

As we demonstrated in the paper, we can always use mechanisms such as deadlock detection and intention sharing during execution, ensuring that this serial execution is only employed in exceptional cases. In the vast majority of cases, robots can execute in parallel safely, so there is no need to worry that this assumption will reduce the efficiency of the robot team.

Nevertheless, we believe it is valuable to explore in future work whether it is possible to design an improved algorithm that addresses the aforementioned issues during the planning process.

Optimality

MRBTP can guarantee finding a solution for Problem 2; however, since the cost of actions is not considered, MRBTP does not account for the optimality of the solution obtained. In the future, designing an algorithm for solving the optimal multi-robot BT planning problem is also a valuable research field.

Parallelization

MRBTP can achieve a certain degree of multi-robot parallel execution through intention sharing during execution. However, this parallelism is not carefully scheduled. In the future, we may explore the development of an algorithm that can automatically identify parallelizable subtrees and combine it with a runtime scheduling algorithm to achieve more efficient multi-robot parallelism.

Subtree Generation

Currently, we rely on the task reasoning capabilities of LLMs to generate task-specific subtrees. This approach becomes ineffective when LLMs are unavailable. Therefore, we might explore the development of an efficient subtree pre-planning algorithm that does not depend on LLMs, to quickly obtain useful subtree structures before the formal planning process of MRBTP.

C.Experiments

Computing Infrastructure

All experiments were conducted on a system equipped with an AMD Ryzen 9 5900X 12-Core Processor (3.70 GHz base clock), 128 GB DDR4 RAM, and running Windows 10, 64-bit.

Simulation Scenarios

The experimental design encompasses two distinct scenarios, each with two simulation modes: computational and scenario simulations. In the Warehouse Management scenario, up to 8 robots are tasked with organizing supplies across 5 designated rooms. This setup utilizes two action predicates, `OpenRoom` and `MovePackage`, and four condition predicates: `IsOpen`, `IsClose`, `IsHolding`, and `IsInRoom`. Similarly, the Home Service scenario involves up to 4 robots assigned to household tasks, employing the same set of action and condition predicates across both simulation modes. Figure 5 depicts the two simulation scenarios, and Table 5 details the action and condition predicates used in the scenario simulation mode.

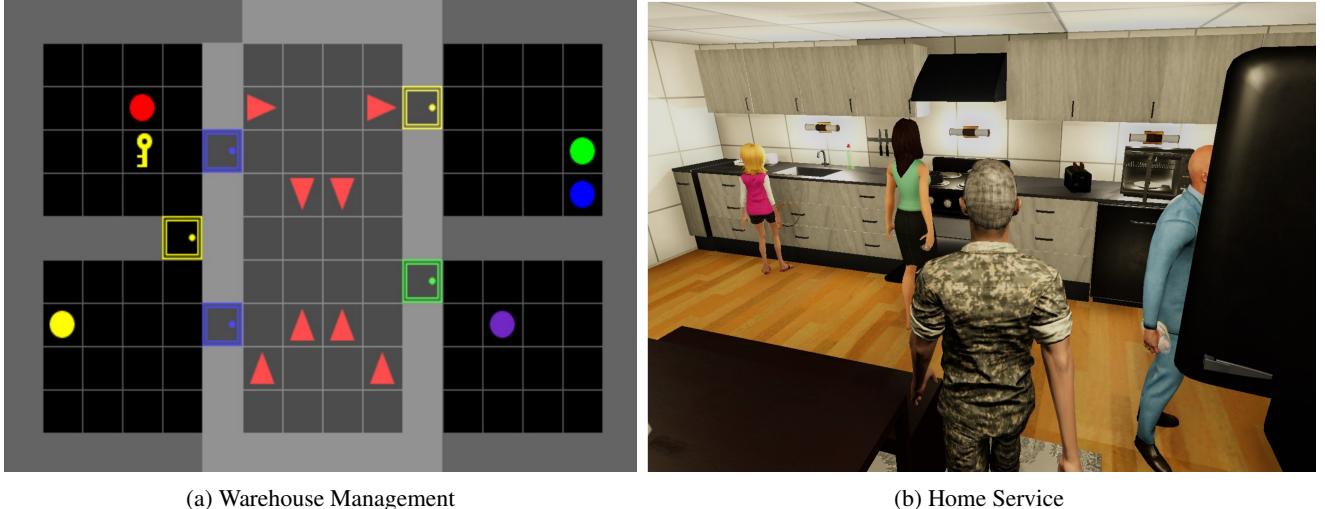


Figure 5: Simulation Scenario Overview

Table 5: Action predicates for different scenarios

Scenarios	Included Action Predicates
Warehouse Mangement	<code>GoToInRoom</code> , <code>GoBtwRoom</code> , <code>PickUp</code> , <code>PutInRoom</code> , <code>PutNearInRoom</code> , <code>Toggle</code>
Home Service	<code>Walk</code> , <code>LeftPut</code> , <code>LeftPutIn</code> , <code>LeftGrab</code> , <code>RightGrab</code> , <code>Open</code> , <code>Close</code> <code>RightPut</code> , <code>RightPutIn</code> , <code>SwitchOn</code> , <code>SwitchOff</code>

Goal Distribution in Datasets

In the *Effectiveness of Task-Specific Subtree Pre-Planning* experiments, the datasets involved goals that were composed of 1 to 5 conditions. The overall distribution of these conditions across the datasets is presented in Table 6.

Table 6: Overall distribution of conditions

Condition	Overall Percentage
<code>IsOpen/Close(obj)</code>	11.9%
<code>IsSwitchOn/Off(obj)</code>	19.5%
<code>In(obj,container)</code>	27.0%
<code>On(obj,surface)</code>	41.6%

Action Allocation Process Based on Homogeneity (α) Parameter Let α represent the Homogeneity parameter, where $0 \leq \alpha \leq 1$. Assume there are n robots, and let \mathcal{A} denote the set of actions, with each action $a \in \mathcal{A}$. The cardinality of \mathcal{A} is denoted by $|\mathcal{A}| = m$.

The action allocation process is described as follows:

- Initially, each action $a_i \in \mathcal{A}$ is randomly assigned to a unique robot $r_j \in \{r_1, r_2, \dots, r_n\}$, ensuring that each action is allocated to exactly one robot.
- The number of additional robots to which each action will be redundantly assigned is determined by $k = \text{int}(m \times \alpha)$.
- Each action a_i is then randomly reallocated to k additional robots, ensuring that these robots do not include the one initially assigned to the action.

LLM

Prompt The objective of this study is to employ the GPT-4 model to generate goal-oriented actions for each robot, utilizing detailed task information to support the pre-planning of subtrees. The input prompt is systematically structured in JSON format, encompassing condition predicates, action predicates, and comprehensive task details, including goals, initial states, objects, and the actions available to each robot. The model is programmed to produce a list of dictionaries, also in JSON format, where each dictionary corresponds to a specific robot and contains multiple sets of goal-related actions derived from the model's inference. This study takes advantage of GPT-4's capability to handle JSON-structured inputs and outputs, allowing for the definition of a precise JSON schema that ensures strict adherence to the predefined format. This method enhances consistency and predictability, while also streamlining subsequent data processing. For a detailed example of the input prompt, refer to .

Prompt

[Condition]

IsNear_self_!ALL $_i$, IsOn_!GRABBABLE $_i$ _!SURFACES $_i$, IsIn_!GRABBABLE $_i$ _!CONTAINERS $_i$,
IsOpen_!CAN_OPEN $_i$, IsClose_!CAN_OPEN $_i$, IsSwitchedOn_!HAS_SWITCH $_i$, IsSwitchedOff_!HAS_SWITCH $_i$

[Action]

Walk_!ALL $_i$, RightGrab_!GRABBABLE $_i$, LeftGrab_!GRABBABLE $_i$, RightPut_!GRABBABLE $_i$ _!SURFACES $_i$,
LeftPut_!GRABBABLE $_i$ _!SURFACES $_i$, RightPutIn_!GRABBABLE $_i$ _!CONTAINERS $_i$,
LeftPutIn_!GRABBABLE $_i$ _!CONTAINERS $_i$, RightGrabFrom_!GRABBABLE $_i$ _!CONTAINERS $_i$,
LeftGrabFrom_!GRABBABLE $_i$ _!CONTAINERS $_i$, Open_!CAN_OPEN $_i$, Close_!CAN_OPEN $_i$,
SwitchOn_!HAS_SWITCH $_i$, SwitchOff_!HAS_SWITCH $_i$

[Example]

[Task Information]

”goal”: [”IsOn(mug,nightstand)”, ”IsSwitchedOn(tablelamp)”, ”IsOpen(book)”),
”init_state”: [”IsSwitchedOff(tablelamp)”, ”IsClose(book)”, ”IsClose(nightstand)”),
”objects”: [”mug”, ”nightstand”, ”tablelamp”, ”book”, ”remotecontrol”, ”drawer”],
”action_space”: [
[”Walk”, ”SwitchOn”, ”Open”, ”Close”],
[”Walk”, ”RightGrab”, ”RightPut”, ”SwitchOn”, ”RightPutIn”],
[”Walk”, ”RightGrab”, ”RightPut”, ”SwitchOn”, ”Open”, ”Close”, ”RightPutIn”]]

[Output]

”multi_robot_subtree_ls”: [{
”WalkToSwitchOntablelamp”: [”Walk(self,tablelamp)”, ”SwitchOn(self,tablelamp)”],
”WalkToOpenBook”: [”Walk(self,book)”, ”Open(self,book)”],
”WalkToOpenNightstand”: [”Walk(self,nightstand)”, ”Open(self,nightstand)”]
},
{
”WalkToPutMugOnNightstand”: [”Walk(self,mug)”, ”RightGrab(self,mug)”, ”Walk(self,nightstand)”,
”RightPut(self,mug,nightstand) ”],
”WalkToSwitchOntablelamp”: [”Walk(self,tablelamp)”, ”SwitchOn(self,tablelamp)”],
”WalkToPutRemoteInNightstand”: [”Walk(self,remotecontrol)”, ”RightGrab(self,remotecontrol)”,
”Walk(self,nightstand)”, ”RightPutIn(self,remotecontrol,nightstand)”]
},
{
”WalkToPutMugOnNightstand”: [”Walk(self,mug)”, ”RightGrab(self,mug)”, ”Walk(self,nightstand)”,
”RightPut(self,mug,nightstand) ”],
”WalkToSwitchOntablelamp”: [”Walk(self,tablelamp)”, ”SwitchOn(self,tablelamp)”],
”WalkToOpenBook”: [”Walk(self,book)”, ”Open(self,book)”],
”WalkToPutRemoteInNightstand”: [”Walk(self,remotecontrol)”, ”RightGrab(self,remotecontrol)”,

```

    "Walk(self,nightstand)", "RightPutIn(self,remotecontrol,nightstand)"],
    "WalkToOpenNightstand": [ "Walk(self,nightstand)", "Open(self,nightstand)"] ]
}

```

[System]

1. For each task, generate all possible composite actions for each robot based on its goals, initial state, and available action space. Repetition of composite actions is permissible.
2. **[multi_robot_subtree_ls]** is a list where each entry is a dictionary [subtree_dict] containing all task-related composite actions for a robot. Keys in [subtree_dict] are composite action names, and values are sequences of atomic actions, ordered such that each action's effect serves as the precondition for the next. Using the current [Task Information] and [Example], construct **[multi_robot_subtree_ls]** for each robot.
3. The length of **[multi_subtree_list]** equals the number of robots and corresponds to the number of action lists in **[action_space]**. With **{num_agent}** robots, **[multi_subtree_list]** contains **{num_agent}** dictionaries, each with 1-5 key-value pairs.

Automatic Reflective Feedback To validate the outputs generated by the LLM, we developed an automated checker that provides feedback in cases of grammatical errors, action sequences that cannot be pre-planned into a subtree, or an insufficient number of action sequences generated for each robot.

A specific example is provided in , where two robots and a key are in Room 0, with the key able to unlock the door to Room 1. The goal is to move a ball from Room 0 to Room 1. The LLM correctly inferred two action sequences: `UseKeyToOpenDoor` and `MoveBallBetweenRooms`. However, the output contained grammatical errors, including incorrect objects, action predicates, and missing action parameters. Additionally, the `action_space` list indicated two robots, but the LLM only provided an action sequence dictionary for one robot. Our automated checker detected these issues, generating natural language feedback for the LLM to refine its response. Upon revision, the LLM corrected the grammatical errors and produced two dictionaries representing the `action_space` for both robots. Subsequently, the pre-planning algorithm attempted to generate subtrees. However, it failed when expanding the `UseKeyToOpenDoor` subtree, as `Toggle(self,door-0)` could not be extended after `PickUp(self,ball-0)`. The failure occurred because the add effects of `Toggle(self,door-0)` (`IsOpen(door-0)`) do not satisfy the preconditions of `PickUp(self,ball-0)` (`IsEmpty(self)`, `IsIn(ball-0,room-0)`, `IsIn(self,room-0)`). The automated checker identified this mismatch and applied the reflective feedback template, which was then provided to the LLM for further refinement. Ultimately, the LLM successfully inferred a valid sequence.

An Example of LLM Process with Automatic Reflective Feedback

[Task Information]

```

"goal": ["IsInRoom(ball-0,room-1)"],
"init_state": ["IsInRoom(ball-0,room-0)",  

    "IsInRoom(key-0,room-0)", "IsClose(door-0)", "IsInRoom(robot-0,room-0)",  

    "IsInRoom(robot-1,room-0)", "IsInRoom(robot-1,room-0)", "IsHandEmpty(robot-0)", "IsHandEmpty(robot-1)"]
"objects": ["key-0", "ball-0", "door-0", "room-0", "room-1"],
"action_space": [  

    ["GoToInRoom", "PickUp", "PutInRoom", "Toggle"],  

    ["GoToInRoom", "GoBtwRoom", "PickUp", "PutInRoom"]]
}

```

[LLM]

```

"multi_robot_subtree_ls": [
    "UseKeyToOpenDoor": ["GoToInRoom(self,key-0,room-0)", "PickUp(key-0)", "GoToInRoom(self,door-0,room-0)",  

        "Open(self,door-0)",  

        "MoveBallBetweenRooms": ["GoToInRoom(self,ball-0,room-0)", "PickUp(self,ball-1)",  

            "GoBtwRoom(self,room-0,room-1)", "PutInRoom(self,ball-0,room-1)"]
    ]
}

```

Reflective Feedback:

1. The object **["ball-1"]** does not exist and must be an element of the set **[objects]**.
2. The action predicates **["PickUp(key-0)", "Open"]** either do not exist or are incorrectly formatted. They must be part of the set **[action_space]**.

3. The number of robots involved in this task is **{num_agent}**, which implies that the [multi_subtree_list] should contain **{num_agent}** dictionaries. Each of these dictionaries should have 1-4 key-value pairs. Please revise accordingly.

[LLM]

```
"multi_robot_subtree_ls": [{  
    "UseKeyToOpenDoor": ["GoToInRoom(self,key-0,room-0)", "PickUp(self,key-0)",  
    "GoToInRoom(self,door-0,room-0)", "Toggle(self,door-0)", "PickUp(self,ball-0)"]  
},  
{  
    "MoveBallBetweenRooms": ["GoToInRoom(self,ball-0,room-0)", "PickUp(self,ball-0)",  
    "GoBtwRoom(self,room-0,room-1)", "PutInRoom(self,ball-0,room-1)"]  
}]
```

Reflective Feedback:

- 1."UseKeyToOpenDoor" cannot be pre-planned.
- 2."Toggle(self,door-0)" and "PickUp(self,ball-0)" cannot be pre-planned together.
- 3.[Toggle(self,door-0).add] = ("IsOpen(door-0)")
- 4.[PickUp(self,ball-0).pre] = ("IsHandEmpty(self)", "IsIn(ball-0,room-0)", "IsIn(self,room-0)")
- 5.[Toggle(self,door-0).add] condition does not belong to the [PickUp(self,ball-0).pre] conditions.

[LLM]

```
"multi_robot_subtree_ls": [{  
    "UseKeyToOpenDoor": ["GoToInRoom(self,key-0,room-0)", "PickUp(self,key-0)",  
    "GoToInRoom(self,door-0,room-0)", "Toggle(self,door-0)"]  
},  
{  
    "MoveBallBetweenRooms": ["GoToInRoom(self,ball-0,room-0)", "PickUp(self,ball-0)",  
    "GoBtwRoom(self,room-0,room-1)", "PutInRoom(self,ball-0,room-1)"]  
}]
```