Knowledge-Based Systems 260 (2023) 110181

Contents lists available at ScienceDirect

Knowledge-Based Systems

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

Automatic Decentralized Behavior Tree Synthesis and Execution for Coordination of Intelligent Vehicles

Tadewos G. Tadewos, Laya Shamgah, Ali Karimoddini *

Department of Electrical and Computer Engineering, North Carolina Agricultural and Technical State University, Greensboro, NC 27411, USA

ARTICLE INFO

Article history:
Received 27 March 2022
Received in revised form 13 November 2022
Accepted 4 December 2022
Available online 9 December 2022

Keywords:
Intelligent vehicles
Behavior Tree
Automatic synthesis
Unmanned aerial vehicles

ABSTRACT

This paper proposes a systematic approach for automatic tasking and coordination of a heterogeneous team of cooperative autonomous vehicles forming an intelligent vehicle. Each vehicle is equipped with different resources, operating in a shared dynamic environment, and capable of executing a set of specific tasks. To coordinate such a heterogeneous team of vehicles, we develop a hierarchical modular coordination algorithm for generating local Behavior Trees (BTs) for tasking individual vehicles so that they, as a team, can collectively achieve a global mission. A hierarchical auctioning algorithm is embedded in the proposed framework to effectively assign tasks among the vehicles, so that they can complement each other and complete a mission that may not be achieved individually. Furthermore, by introducing operational and duration cost weight terms, the proposed approach provides the possibility to adjust the total operation cost and duration of the tasks. The details of the developed algorithms are illustrated through different case studies.

1. Introduction

With advances in communication, computation, and control technologies, it is now becoming possible to deploy intelligent systems in the form of a heterogeneous team of autonomous vehicles with different capabilities (sensors and actuators) to collectively accomplish complex missions and tasks, which are distributed in time and space and may not be possible to be achieved individually [1–4]. A cooperative control strategy not only can handle such complex scenarios, but also could significantly reduce the cost, enhance the resilience of the overall system, and improve the team functionality through sharing resources and distributing tasks and loads [5–8]. Nonetheless, multi-agents cooperation introduces challenges and complexities including but not limited to task decomposition, task assignment, communication, task execution, and task monitoring [9]. A common method for tasking multi-agent systems is to employ scheduling mechanisms [10]. For example, [11] provides a reliable scheduling algorithm for a team of agents with the capability to conduct dynamic rescheduling. However, cooperative tasking in general is beyond simple scheduling and often involves dependencies and join execution of actions. Heuristic methods [12,13] and bio-inspired approaches [14–16] have been employed in the literature for more complex tasking scenarios. However, such methods often lack proof of performance. Alternatively, one can use consensus-based algorithms to minimize the cost or maximize the number of tasks assigned to an agent [17]. The challenge is that generally tasks are sequences of actions that have to be completed in a particular sequence and often in collaboration with other agents, which is beyond the scheduling problem as it requires a coordinator(s) to synthesize and execute a scheduling and sequencing plan in a collaborative setting.

Another common approach to address tasking and coordination of multi-agent systems is to rely on group behaviors that emerge from group interactions and individual decision-makings based on local information. Threshold-based methods follow a simple rule for decision making: if "decision criteria > threshold", then the agent picks the task. A decision to select a particular task depends on an individual's perception of a task and individual's response threshold for the task. The game theoretical approaches formulate the tasking and coordination problem for multi-agent systems to form a disjoint coalition as a hedonic formation game between the agents and the tasks that are interacting [18,19]. The advantage of these methods is that they rely less on the information about modeling the environment, tasks, and individuals. However, this makes it difficult to predict the exact behavior of individuals, focus on single static global tasks, and formally design, analyze and implement a cooperative strategy [18]. In market-based methods [20,21], the task is decomposed into some subtasks to be assigned through an auctioning process. For this purpose, each individual agent offers its bid and then the auctioneer collects the bids and decides which agent is more eligible to

* Correspondence to: Department of Electrical and Computer Engineering, North Carolina A&T State University, 1601 East Market Street, Greensboro, NC, 27411, USA.
E-mail address: akarimod@ncat.edu (A. Karimoddini).

https://doi.org/10.1016/j.knosys.2022.110181
0950-7051/

win the announced task. In [22], an optimal multi-robot tasking framework is introduced by modeling each robot as a weighted transition system and composing the model of the robots with the mission requirement expressed as a linear temporal logic (LTL). However, this framework is centralized and as the number of participating robots increases, scalability becomes a bottleneck. In [23], a decentralized multi-agent control strategy following a bottom-up approach is presented, where each agent synthesizes a local coordinator to ultimately meet a global mission. However, since the agents have no prior knowledge about each local task, conflicts among agents could arise which has to be resolved by a centralized mission controller. In [24–26], an automated task decomposition approach is introduced using the natural projection to local sensing/actuation maps when the tasks are given in the form of automata. In [27,28], an automated supervisory control framework models the system and its corresponding specifications as a discrete event system (DES) where the developed supervisor centrally coordinates the robots indirectly by enabling or disabling the events. In turn, each robot chooses to execute or not to execute the enabled events that are communicated back to the supervisor to enforce a teaming behavior. These automata-based approaches commonly face state explosion problem and cannot be flexibly used/re-used with existing structures in a modular way.

In [29], an autonomous task allocation mechanism is proposed where agents initially split into groups to cover an area, and then, the task allocation is conducted autonomously by negotiation among agents. In [30], a task is distributed among agents by considering the current state of the agents and adding a behavior layer that takes into account the previous decisions of the agents to improve collaboration among the agents. In [31], multiple heuristic algorithms are used to come up with the best task allocation plan for agents operating in a battle-field. Each agent selects an algorithm based on the flight characteristics and the environment they operate to maximize efficiency. These methods are often computationally expensive, challenging their scalability and the application to cases where the environment changes or tasks are introduced on-the-fly.

In [32] a decentralized coordination architecture to allocate and manage resources to meet the mission requirement has been proposed, however the framework requires a global view of available resources to assign tasks. In [33], the problem of efficient task allocation is addressed in a distributed manner by solving non-linear MILP formulation. The major issue in MILP-based techniques is that the computation is done offline and as the environment changes, an expensive computation has to be performed again which is costly if the environment changes frequently. Like MILP, a major challenge in almost all aforementioned techniques is that with these techniques either the computation is done offline or the agents have limited intelligence to coup with dynamic environments and emergent scenarios.

An alternative planning solution is to employ Behavior Trees (BTs) [34–37]. BTs are graphical mathematical models for the execution of tasks with inherent hierarchical, modular, and reactive properties. With BTs it is fairly more convenient to manage, modify, and add tasks or subtasks due to the modular and scalable structure of BTs. Further to incorporate safety, in [38,39] a safe BT synthesis method for a single agent has been proposed. In [40] BTs are used to address a logistic/load-delivery problem that utilizes a predefined BT. In [41], given a global BT controller and assuming that the tasks are decomposable, a heuristic approach is employed to create local BTs for each agent. However, the question of how to determine the global BT is left unanswered. To extend this framework to multi-agent systems, one way is to use the method in [42] to develop a global BT and then decompose the global BT to local BTs using the method in [41]. However, this approach is not computationally efficient and may end up with the state explosion problem for larger systems.

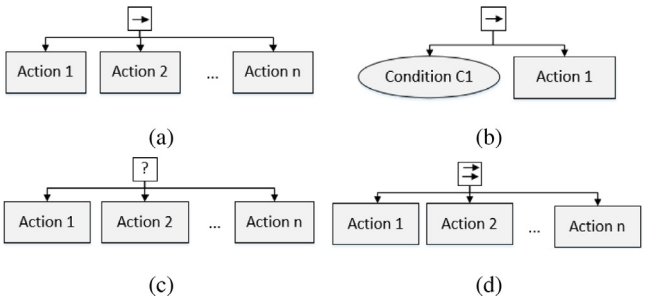

Fig. 1. Building blocks of Behavior Trees: (a) A sequence node, (b) Execution of Action 1 based on Condition C1, (c) A selector node, (d) A parallel node.

To address the challenges on the coordination of multi-agent systems, in this paper, we distributively and reactively synthesize the local BTs on-the-fly for a set of streamed tasks, so that each agent is responsible for synthesizing its own BT. Moreover, our proposed technique incorporates a tasking mechanism by assigning the tasks via a market-based auctioning algorithm to minimize the cost. In the proposed framework, collaboration among agents is needed if and only if a single robot cannot do the task alone, thus resource utilization is improved leaving other robots for new tasks. In summary, the contributions of the paper include:

- developing a hierarchical modular coordination algorithm for automatically generating local BTs on-the-fly for tasking individual vehicles so that they, as a team, can collectively and reactively achieve a global mission,
- incorporating a market-based auctioning algorithm to for assigning the tasks and actions while minimizing the overall operation and duration cost.
- deriving the sufficient conditions and formally proving the correctness of the proposed method in the sense that the assigned mission can be always completed.
- analyzing the computation cost and investigating the scalability of the method by applying the method to scenarios with large numbers of tasks and agents.
- applying the proposed method to several case studies, demonstrating the effectiveness of the proposed approach.

The rest of the paper is organized as follows. The background and necessary preliminaries on behavior trees are provided in Section 2, followed by problem formulation for the decentralized coordination of multi-agent systems. Section 3 describes our proposed approach for synthesizing decentralized BTs in detail. Section 5 applies the proposed method to several case studies to illustrate the implementation of the developed algorithms. Finally, Section 6 concludes the paper. The terms being used in this paper along with their explanation are provided in Table 8 at the end of the paper.

2. Problem description

2.1. Behavior Tree structure

Formally BTs are defined as a directed acyclic graph with three types of nodes: Root node, Leaf node (condition and action), and Composite node (selector, sequence, parallel). Root node is the starting vertex in the BT graph. Leaf nodes are terminal nodes in a BT's branch in the form of an *action* or a *condition*. Composite nodes are used to compose leaf nodes in the form of *selection*, *serial* (*sequence*), or *parallel*. Fig. 1 shows different composite

nodes. In the hierarchical architecture of a BT graph, the nodes on the higher layers are called *parent* nodes, whereas linked nodes in their subtrees are called *children*. A node in a BT's hierarchical structure could return to its parent with *success*, *running*, or *failure*, if it completes its job or satisfies a condition, if it is in the process of completing its job or checking the condition, and if the job cannot be completed or the condition is violated, respectively. Generally, the execution of a task is initiated by the root node which sends a tick (enabling signal) with a certain frequency to its children. Then, the enabled child activates another child or returns its execution status as *running*, *failure*, or *success* to its immediate parent. In this way, the actions are executed from bottom left of the BT, returning success/failure/running to their parents. By systematically composing leaf nodes (actions and condition nodes), using one of the composite nodes (*sequence, selector or parallel* nodes) a complex BT with a modular architecture can be designed.

2.2. Preliminaries and notations

We use BTs to formulate the coordination and tasking for multi-agent systems over the following components:

(1) The set $R = \{R_1, \ldots, R_M\}$, which includes a team of robots, where $M \in \mathbb{N}$ is the number of agents. We also define a set $P_R = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ and a set $V = \{v_1, \ldots, v_M\}$ which represent the position and velocity of the agents, respectively. In this paper, the terms agents, robots, and vehicles are used interchangeably. For each robot, any component in its operating space other than the robot itself is considered to be the environment.

(2) The set A is the global action bank, which contains a set of actions A_k, $k = 1, \ldots, L$, where $L \in \mathbb{N}$ is the total number of actions. We define a set of action capability indicators $\hat{a}_{ik}$, $i = 1, \ldots, M$, $k = 1, \ldots, L$, for which $\hat{a}_{ik} = 1$ if the robot R_i can accomplish Action A_k, otherwise $\hat{a}_{ik} = 0$. Here, the robots are assumed to perform a single action at a time.

(3) The set T which includes a set of complex tasks (a task can be decomposed into multiple sets of actions that could satisfy the same task goal in different ways [43]) T_j, $j = 1, \ldots, N$, where $N \in \mathbb{N}$ is the number of tasks along with a task position $P_T = \{P_{T_1}, \ldots, P_{T_N}\}$. The accomplishment of each task (*objective* of T_j), can be captured by meeting a condition C_j. For example, if the task T_1 is to "reach a goal region", then C_1 is "being at the goal region". We also define a set of task assignment indicators x_{ij}, $i = 1, \ldots, M$, $j = 1, \ldots, N$, for which $x_{ij} = 1$ if the task T_j is assigned to R_i to handle it individually or in collaboration with other robots, otherwise $x_{ij} = 0$. Similarly, we define a set of action assignment indicators $\hat{x}_{ijk}$, $i = 1, \ldots, M$, $j = 1, \ldots, N$, $k = 1, \ldots, L$, for which $\hat{x}_{ijk} = 1$ if action A_k is assigned to R_i for completion of T_j. To reach the "goal" of a task T_j, depending on the agent that is responsible to handle the task, a series of actions from the action bank A should be completed, where the last action should meet C_j. In our proposed framework, only a robot that can accomplish an action which meets C_j can be a candidate for being selected to handle T_j. Such a robot can complete an action to meet C_j and may delegate the prerequisite actions to other agents if necessary. Further, to coordinate the delegation process we define the indicators $\hat{a}_{ijk}$, $i = 1, \ldots, M$, $j = 1, \ldots, N$, $k = 1, \ldots, L$, for which $\hat{a}_{ijk} = 1$ if robot R_i is needed to accomplish A_k to complete the task T_j, otherwise $\hat{a}_{ijk} = 0$.

(4) The set F includes a set of value functions. A value function $F_{ij} : R \times T \to \mathbb{R}^+$ describes the cost of handling the task T_j by R_i based on the performance, energy, and proximity. Robot R_i can accomplish the actions towards T_j individually or delegate the actions to other robots if necessary.

(5) The set $\hat{C}$ includes preconditions c_{ik} for an action A_k of agent R_i, where $i = 1, \ldots, M$, $k = 1, \ldots, L$. Further, since an action could have multiple preconditions, c_{ik} includes a list of preconditions c_{ikp}, $i = 1, \ldots, M$, $k = 1, \ldots, L$, and $p = 1, \ldots, P_k$, where P_k is the total number of preconditions for action A_k, and c_{ikp} specifies the *pth* precondition for completing action A_k by robot R_i. We also capture the accomplishment of each action A_k by meeting the condition $\hat{C}_k$.

(6) Consider a discrete clock *clk* with a granularity of 1 s, i.e., $clk = clk + 1$ (this can be of different step sizes if needed). The clock *clk* represents the elapsed time starting from the first task announcement. Then, we define $\triangle\hat{t}_{ik}$, $i = 1, \ldots, M$, $k = 1, \ldots, L$, which represents the duration that the agent R_i needs to complete the action A_k. Consider $o = 1, 2, \ldots$, where $o \in \mathbb{N}$ is the sample time index. We then define an action timeline indicator t_{io}, $i = 1, \ldots, M$, where $t_{io} = 1$ during the time that R_i is assigned to perform one of the actions A_*, which takes R_i for $\triangle\hat{t}_{i*}$ time units.

(7) We define an operation $R_i \models con$ which checks if the agent R_i satisfies the condition *con* at its current state, where the condition *con* can be a condition for a task, i.e., C_j, or a precondition for an action, c_{ikp}.

2.3. Assumptions and problem formulation

To do automatic tasking for multi-agent systems, analogues to [42], we make the following assumptions:

Assumption 1. Each agent can verify if any of its local actions have succeeded, failed, or if it is running. This enables the agent to monitor the status of each action.

Assumption 2. Each agent can verify if any of its local action's condition is true or false. This enables the agent to perform an action only when a condition is false, i.e, the condition acts as a guard.

Assumption 3. For each goal and for each initial configuration of the agents, there exists a sequence of actions that can be executed by the agents leading to the achievement of the goal.

Assumption 4. The effect of the dynamic environment can void the accomplishment of the actions at most a finite number of times. This assumption is made to avoid sticking in a live-lock of repeating an action and being voided by the environment over and over, preventing the agent to achieve its goal.

Assumption 5. Given any two actions A_i and A_j, if the execution of A_i requires the execution of A_j, A_j must not require the execution of A_i. This assumption prevents deadlocks due to cyclic dependency.

Assumption 6. All actions are ultimately reversible. That is, each action can be undone through a finite sequence of actions. The ability to reverse an action is required to resolve possible conflicts.

Assumption 7. For each action, there exists at least one agent to achieve it, which can be accomplished by a low level controller embedded in that agent.

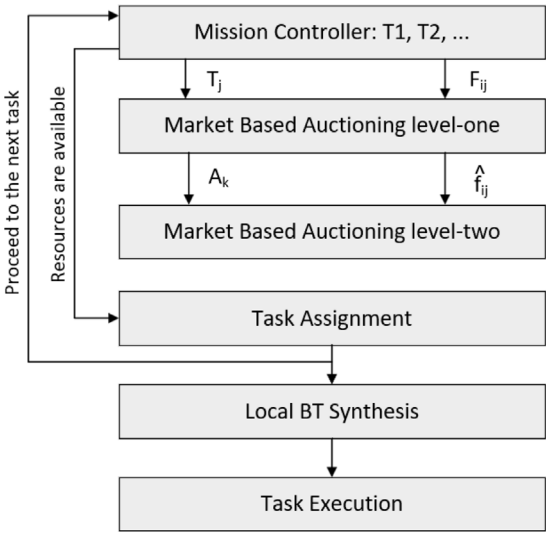

Fig. 2. The proposed automatic BT coordinator synthesis and execution framework.

Assumption 8. We assume that the global communication is available, i.e., messages broadcasted by an agent can be received by all other agents.

Now, given R, T, F, A, and $\hat{C}$, and making Assumptions 1–7, the tasking and coordination problem for multi-agent systems can be stated as:

Problem 1. Consider a *mission* described by T as a set of streamed tasks T_j, $j = 1, \ldots, N$, to be completed by a set of robots R_i, $i = 1, \ldots, M$, by executing the actions A_k, $k = 1, \ldots, L$, assuming that each action is executable by some of the agents (at least one). Also, consider that there is no order and dependency among the tasks, other than the order in which tasks are issued. Synthesize and execute decentralized BT_i to coordinate the individual robots R_i to collectively achieve a set of tasks T_j.

3. Automatic behavior tree synthesis and execution

To address Problem 1, we propose a systematic method for generating the local (individual) BTs by combining a market-based auctioning algorithm with reactive BT synthesis, so that the generated local BTs can collectively satisfy the mission specification. Fig. 2 shows the general description of the proposed technique. Given a mission in the form of a set of tasks T_j, $j = 1, \ldots, N$, we conduct a two-level auctioning to assign the tasks and their corresponding actions to the agents as it will be discussed in Section 3.1. Then, local BTs will be generated for the robots which will be used for task execution as detailed in Section 3.2. To manage the multi-level delegation, we allow two levels of auctioning so that in the second level of auctioning, the delegated agent cannot delegate an action and should return to the first level auctioneer if needed. However, the proposed method can be extended to more levels of auctioning by following a similar approach discussed in this paper with proper adjustments on the developed algorithms. Clearly, adding more levels of auctioning requires more communication and process burden.

3.1. Task assignment for coordination of multi agent systems

Task assignment to a set of robots with different resources and capabilities might raise a conflict when every participating agent strives to maximize its utility by performing as many tasks as possible. To fairly assign tasks and avoid conflicts, we propose a two-level market-based auctioning algorithm. Generally, an auctioning algorithm has four stages: *announcement stage*, where the auctioneer announces the auctioning item; *submission stage*, where the individual agents submit (bid) the price (cost) for the auctioning item and communicate it with the auctioneer; the *selection stage*, where the auctioneer evaluates the cost and selects the lowest offer, and the *contract stage*, where the negotiation is concluded by forming a contract between the auctioneer and the responsible agent. Following this process, at the first level of the proposed auctioning-based task assignment, the *Mission Controller* (*MC*) announces a task T_j. We assume that MC announces and auctions the tasks one at a time. Then, those agents, R_i, that can accomplish an action to meet C_j bid for this task. For this purpose, these candidate robots should identify the actions that are needed (to be done by themselves or through delegation to other robots) to complete T_j. For each robot, R_i, using a weighted sum method, the cost of each action has to be calculated (both local and delegated actions) to find the cost, F_{ij}, for completing the task T_j as:

$$F_{ij} = \alpha f_{ij} + \beta t_{ij} \tag{1}$$

where f_{ij} is the operation cost of task T_j when handled by R_i, t_{ij} is the duration that R_i needs to complete task T_j, and α and β are weights for operational cost and duration respectively, which should be selected proportional to their relative importance based on the preferences of the designer/operator. Clearly, f_{ij} and t_{ij} would be different for different actions and for different agents. For example, in action *MoveTo*(*destination*), which involves moving toward an object, the distance from the object and the velocity of the agent impacts f_{ij} and t_{ij}. However, if the action does not involve any movement, e.g., the action is to take an image, we simply assume a fixed cost for the action. Then, *MC* selects the agent R_i that can handle T_j with the lowest cost and concludes the auction by forming the contract with R_i and letting $x_{ij} = 1$. This indeed is equivalent to the following minimization:

$$\begin{aligned} & F_j = \min_{x_{ij}} \sum_{i}^{M} (\alpha f_{ij} + \beta t_{ij}) x_{ij}, \ \forall j \\ & \textit{subject to} \sum_{i}^{M} x_{ij} = 1 \\ & \alpha, \beta > 0 \\ & x_{ij} \in \{0, 1\}, \ \forall i \end{aligned} \tag{2}$$

where x_{ij} is an indicator that task T_j is assigned to R_i.

When an agent R_i forms a contract to complete a task T_j, the robot should perform a sequence of actions. So, the cost and duration of completing the task T_j by robot R_i, f_{ij} and t_{ij}, should be computed as the summation of costs and duration, $\hat{f}_{ik}$ and $\delta\hat{t}_{ik}$, for all actions A_k required for the task T_j. The point is that some of the actions may need to be delegated to other agents. Thus, the cost function f_{ij} in the first level auction may not be possible to be calculated solely based on the costs of R_i's actions, and hence, the agent R_i should find the cost of delegated actions through a second level auction. In this way, assuming that the actions that the robot R_i is capable of doing will not be delegated, the total cost and duration for completing the task T_j through the robot R_i will be:

$$f_{ij} = \begin{cases} \sum_{k=1}^{L} \hat{a}_{ijk}(\hat{a}_{ik}\hat{f}_{ik} + (1 - \hat{a}_{ik})\hat{f}_D(ijk)), \forall i, j, \ \text{if } R_i \models C_j \\ \infty, \ \text{Otherwise} \end{cases} \tag{3}$$

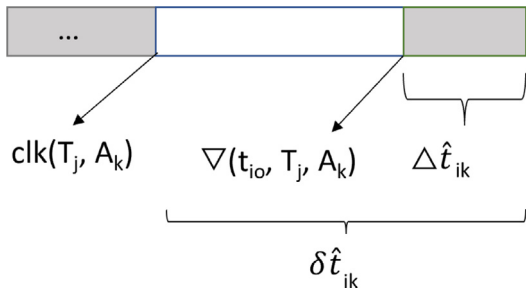

Fig. 3. Timeline for executing A_k by R_i toward Task T_j.

and

$$t_{ij} = \begin{cases} \sum_{k=1}^{L} \hat{a}_{ijk}(\hat{a}_{ik}\delta\hat{t}_{ik} + (1-\hat{a}_{ik})\delta\hat{t}_D(ijk)), \forall i,j, \text{ if } R_i \models C_j \\ \infty, \text{ Otherwise} \end{cases} \tag{4}$$

where $\delta\hat{t}_{ik}$ is the total time (including both operation time and the delay between the instant that the action is required and the action start time) needed to complete action A_k, and $\hat{f}_D(ijk)$ and $\delta\hat{t}_D(ijk)$ are the operational and duration cost of the actions A_k delegated by R_i for completing the task T_j. To calculate the delay in both local and delegated actions, we introduce the function $\nabla(t_{io}, T_j, A_k)$ that returns the time instant that action A_k is available to be started by R_i for task T_j at or after $clk(T_j, A_k)$, where $clk(T_j, A_k)$ represents the time that the action A_k is called for the task T_j, and t_{io} is the timeline indicator. Then, the total time, $\delta\hat{t}_{ik}$, for the execution of an action A_k for R_i, $\delta\hat{t}_{ik}$, can be computed as:

$$\delta\hat{t}_{ik} = \nabla(t_{io}, T_j, A_k) - clk(T_j, A_k) + \triangle\hat{t}_{ik} \tag{5}$$

in which $\delta\hat{t}_{ik}$ is considered as the delay part, $\nabla(t_{io}, T_j, A_k) - clk(T_j, A_k)$, plus the actual action execution time, $\triangle\hat{t}_{ik}$ (see Fig. 3 for the details).

To avoid double assignment, once an action or a task is assigned to an agent, the function $\hat{\nabla}(t_{io}, T_j, A_k)$ returns $[\nabla(t_{io}, T_j, A_k), \nabla(t_{io}, T_j, A_k) + \triangle\hat{t}_{ik}]$ as the time interval in which the action A_k will be executed by R_i and updates the availability indicator t_{io} from 0 to 1 for $\triangle\hat{t}_{ik}$ over this time interval.

The selection of a robot for action A_k through the delegation process can be conducted via the following minimization:

$$\begin{aligned} &\hat{F}_D(ijk) = \min_{\hat{x}_{djk}} \sum_d (\alpha\hat{f}_{dk} + \beta\delta\hat{t}_{dk})\hat{x}_{djk}, \forall k \\ &\textit{subject to } \sum_d \hat{x}_{djk} = 1 \ \forall k, \\ &\quad d = 1 \cdots M, \ d \neq i, \\ &\quad \alpha, \beta > 0 \\ &\quad \hat{x}_{djk} \in \{0, 1\}, \ \forall d \end{aligned} \tag{6}$$

where $\hat{f}_{dk}$ and $\delta\hat{t}_{dk}$ are the operational and duration cost of the delegated action A_k when done by R_d, $\hat{F}_D(ijk)$ is the total cost of action A_k, and $\hat{x}_{djk}$ indicates if action A_k of task T_j is assigned to the agent R_d or not. After finalizing the second-level auctioning in (6), the minimum cost and duration $\hat{f}_D(ijk) = \hat{f}^*_{dk}$ and $\delta\hat{t}_D(ijk) = \delta\hat{t}^*_{dk}$ are returned to (3) and (4) to be used in the first-level auctioning in (2), where $\hat{f}^*_{dk}$ and $\delta\hat{t}^*_{dk}$ are the cost and duration of action A_k by robot R_d which minimize (6).

3.2. Decentralized behavior tree synthesis and execution algorithm

In contrast to the execution of a BT which is bottom-up, to synthesize a BT we should follow a top-down approach where individual actions with predefined conditions are sequenced in a way that the execution of the synthesized BT satisfies the mission goal. Since we are synthesizing the BTs on-the-fly, we use a mixed approach. We start with executing the existing parts of BTs from the bottom, whereas we synthesize the missing parts from the top (initially from the task condition C_j). Utilizing the hierarchical and modular nature of BTs, the algorithm automatically generates a BT for coordination of each agent that allows collaborative and decentralized execution of tasks. The algorithm synthesizes and executes a local coordinator BT for each agent and if the assigned agent has the necessary capability to do the entire mission, it will perform the task without any collaboration. In case the agent cannot perform an action, the agent can delegate it to another agent. In other words, the collaboration is need-based, i.e., agents collaborate only if collaboration is required to complete a task.

The overall procedure to generate BTs for individual agents is explained in Algorithms 1–9. These algorithms work as follow: a task T_j is generated by the *Mission Controller* (*MC*). Then, capable agents send the estimated cost of the task, T_j, to *MC* based on which *MC* selects the best agent using a market-based auctioning algorithm. The selected agent, R_i, automatically synthesizes and executes a BT coordinator to meet the mission goal. Since R_i may or may not have the necessary capabilities (actions) to complete the task T_j on its own, it could potentially act as an auctioneer to receive assistance for some of the actions from other agents. Hence, the framework has a two-levels of auctioning mechanism where the first level is responsible for the task assignment and the second level is responsible for the action assignment.

Algorithm 1: *Main*()

```
1 // Initialization
2 Set all x_{i*}= 0 // no task is assigned to R_i
3 Set all x̂_{i**}= 0 // no action is assigned to R_i
4 Set t_io = 0, for o = 1, 2, ··· // the agent is available
5 DelegatedAgents_{i*} ← ∅
6 // Run the BT_i which initially is composed of
    sub-trees needed for auctioning
7 BT_i ← Parallel(T_{Auction_i}, T_{Bid_i}, T_{Contract_i})
```

Algorithm 1 initializes the parameters of the local BT coordinators for the correct operation of the proposed algorithms. Initially, since the agent R_i at the beginning is not assigned a task or an action, the availability indicator variables for tasks (Line 2) and actions (Line 3) as well as its availability indicator (Line 4) are set to zero. Further, a set of global variables $DelegatedAgents_{i*}$ are defined to track the delegated actions and agents (Line 5). Then, in Line 7, BT_i is initialized by the parallel composition of the following sub-trees: $\mathcal{T}_{Auction_i}$, $\mathcal{T}_{Bid_i}$, and $\mathcal{T}_{Contract_i}$, which are responsible for auctioning, bidding, and contracting stages, respectively.

Algorithm 2: *Add BT for* R_i

```
1 function AddBT (C_j) ;
  Input : C_j : Condition for the assigned task T_j
  Output: T_ij : Synthesized BT
2 T_ij ← C_j
    // Start the BT for task T_j from the condition C_j, which is
    used to check if the task is completed or not
3 BT_i ← Parallel(BT_i, T_ij)
    // BT_i represents all BTs of R_i running in parallel
    to execute multiple tasks
4 BTSynthesisandExecution(T_ij, C_j)
```

Now, assume that the task T_j is assigned to the robot R_i as it can meet the condition C_j. For any newly assigned task T_j to R_i, Algorithm 2 initializes a new BT $\mathcal{T}_{ij}$ and sets it to the goal condition C_j (Line 2) (this condition will be used to determine if the task is completed or not). Since R_i can be assigned to do

multiple tasks, $\mathcal{T}_{ij}$ is composed in parallel with the existing BTs, $BT_i = Parallel(BT_i, \mathcal{T}_{ij})$, to handle the current assigned task and previously assigned tasks (Line 3). The BT $\mathcal{T}_{ij}$, which is initialized in Algorithm 2, will then be synthesized by calling Algorithm 3 (Line 4).

Algorithm 3: *BT Synthesis and Execution for R_i*

```
1  function BTSynthesisandExecution (T_ij, C_j) ;
   Input  : C_j : Condition for the assigned task to agent i
            T_ij = Initialized Bt to handles T_j
   Output: T_ij : Syntheisized BT
2  do
3      do
4          r, Ĉ_k ← Execute(T_ij)
5          if Ĉ_k == C_j then
6              break
7              // stop the synthesis and execution process
8          end
9          if (ConflictWithOtherAgents() ≠ ∅) then
10             A_k' = ConflictWithOtherTasks()
11             ReassignAction(j, A_k')
                 //Priority is based on the order of the tasks j and
                 A_k' is the action in conflict
12         end
13     while r == Executable;
14     c_if ← GetConditionToExpand(T_ij)
         //Identify the reason why T_ij is not executable
15     T_ij, T_subtree_ij ← ExpandBT(T_ij, c_if)
         //Resolve the cause by Algorithm 4
16     while Conflict(T_ij, T_subtree_ij) do
17         T_ij ← IncreasePriority(T_ij, T_subtree_ij)
18     end
19 while ¬(R_i ⊨ C_j);
```

Algorithm 3 starts by checking if the agent R_i satisfies the "goal" C_j. If not, the algorithm iteratively updates the BT $\mathcal{T}_{ij}$ until a sequence of actions is obtained (Lines 11–15) (this stage of the algorithm is responsible for synthesizing sub-trees to meet failed condition). Then, the synthesized BT as a whole is executed to achieve the goal of a task (Lines 3–10). Throughout this process, there might occur two kinds of conflicts: conflicts between the actions of a single agent (two actions of an agent are conflicting) and conflicts between tasks (agents that are executing different tasks might need the same resource, e.g., occupying the same spot). Conflicts between the actions can be locally handled during the synthesis process (Lines 13–15). However, conflicts among the tasks cannot be handled during the synthesis process, as the tasks are introduced to *Mission Controller* in real-time. Therefore, conflicts between the agents should be resolved during the execution (Lines 8–10).

So, to synthesize a BT for robot R_i to complete the task T_j, the algorithm starts with $\mathcal{T}_{ij} = C_j$. Then, in a *do while* loop (Lines 3–10), the synthesized BT is executed. For this purpose, $\mathcal{T}_{ij}$ is tested to determine whether it is executable (Line 4). If by the execution of $\mathcal{T}_{ij}$, the assigned task's condition is satisfied, i.e., the condition $\hat{C}_k$ of the last accomplished action is the same as the goal's condition C_j, the execution loop will be terminated (Lines 5–7). When executing the BT, if $ConflictWithOtherTasks() \neq \emptyset$ (Line 8), then this means that there exists a conflict between tasks, i.e., an action, A'_k, from a task T_j, conflicts with another action from a different task, $T_{j'}$. To handle this issue, the function *ReassignAction* updates the agents' BTs that are in conflict (Lines 8–10), on a priority-based criteria by delaying the lower priority BTs, while keeping the highest priority BT as it is. Here, for simplicity, we assume that the tasks are given to the system in order, and T_j has a higher priority than $T_{j'}$ if $j \leq j'$. As a result, the agent(s) with the updated (delayed) BT performs the conflicting actions on another time slot, which results in BTs free from agent-related conflicts.

If $\mathcal{T}_{ij} = C_j$ is not executable, then Algorithm 3 synthesizes/expands the BT (Lines 11–15). A BT is not executable if one of the children condition nodes returns failure. Line 11 identifies the *cause* of failure, c_{if}. The identified *cause* will become a condition in a subtree to resolve the problem by finding alternative actions or other agents (Line 12), as it will be described in Algorithm 4. After updating the BT, due to the addition of a new subtree, $\mathcal{T}_{subtree_{ij}}$, a conflict between actions could arise. For example, consider a situation that $\mathcal{T}_{subtree_{ij}}$ has a condition, c_{ikp}, that has to be *False* (e.g., the robot arm has to be available to open a door, captured by $c_{ikp} = False$, i.e., the arm is free) but another subtree in $\mathcal{T}_{ij}$ sets c_{ikp} to *True* (e.g., if a robot picks an object, captured by $c_{ikp} = True$, i.e., the arm is busy). In this case, $\mathcal{T}_{subtree_{ij}}$ cannot be executed due to the conflict between the actions of "opening the door" and "picking an object". To resolve the conflict, the function $IncresePriority(\mathcal{T}_{ij}, \mathcal{T}_{subtree_{ij}})$ increases the priority of $\mathcal{T}_{subtree_{ij}}$ by moving the subtree toward the left (Lines 13–15), e.g., opening a door should be done when the robot arm is free either by putting down the object or at a time that the arm is free.

As mentioned in Algorithm 3, if a task is not executable, the function $GetConditionToExpand(\mathcal{T}_{ij})$ returns c_{if} as the condition to be met to make $\mathcal{T}_{ij}$ executable. Algorithm 4 then synthesizes the subtree that satisfies the condition c_{if}. In line 2 of Algorithm 4, a local action A_k is identified that could satisfy the condition c_{if}. In case, there are multiple local actions that can satisfy the condition c_{if}, the Algorithm chooses the one with a lower cost. The preconditions of the action A_k, captured by c_{ikp}, are composed by a sequence node to form $\mathcal{T}_{seq_{ij}}$ (Lines 5–8). Then, to allow A_k to be executed at $clk = \nabla(t_{io}, T_j, A_k)$, the timing condition $T_j\_A_k\_time$ is added to the sequence node as a precondition of the action A_k (Line 9), followed by the action A_k itself (Line 10). Here, we assume that the elapsed time (measured in time unit) for each action is an integer multiple of the BT clock, i.e, 1 *time unit* $= k(Bt\_tick\_time)$, $k \in \mathcal{N}$. This facilitates the synchronization of BT_tick_time with actual elapsed time. Next, $\mathcal{T}_{seq_{ij}}$ is composed with $\mathcal{T}_{sel_{ij}}$, initially set as c_{if} (Line 3), by a selector node, to enforce the execution of $\mathcal{T}_{seq_{ij}}$ only in situations where c_{if} is not satisfied (Line 11). To enforce the allocation of A_k to R_i, the timeline indicator t_{io} and action assignment variable $\hat{x}_{ijk}$ are updated (Lines 12–13). However, if no local action exists ($GetLocalActionwithPrecondition(c_{if}) == \emptyset$), then Algorithm 5 is called to find an agent that can accomplish c_{if} (Lines 15–17) through the level-two auctioning mechanism. Finally, c_{if} in T_{ij} is replaced by the sub-tree that can accomplish c_{if} (Line 18), where both the synthesized tree and sub-tree are returned to Algorithm 3 (Line 19).

Algorithm 5 performs the level-two auctioning to find a suitable agent that can execute an action to satisfy c_{if}. The challenge in level-two auctioning is that even though an agent may win a bid for a level-two announced action, the contract/assignment would not be finalized until the level-one auctioning for the corresponding task is concluded. To handle this situation, one way is to repeat the level-two auctioning once the level-one auctioning is concluded to delegate the actions. However, to increase efficiency, we define the variable $DelegatedAgent_{ij}$ to keep track of winning bidders of level-two auctioning announced by R_i for Task T_j. In this way, the bidding/submission for an action to meet c_{if} will only happen if it is not done before. For this purpose, the function $DelegatedAgentSelected(c_{if})$, by examining the variable $DelegatedAgent_{ij}$, returns *False* if an agent has not yet been identified (Line 2). In this case, the condition c_{if} is announced (Line 4) and the participating agents submit a cost (Line 5) for the action A_f as calculated in Algorithm 8. In Line 6, the function *Selection_L2*() conducts the level-two auction in (6) to select the

Algorithm 4: *Expand Behavior Tree Module for R_i*

1 function ExpandBT ($\mathcal{T}_{ij}, c_{if}$);
Input : c_{if} = condition (cause) for T_j not being executable
Output: $\mathcal{T}_{ij}$ = Expanded BT
2 $A_k \leftarrow GetLocalActionwithPrecondition(c_{if})$
// Identify local actions that satisfy c_{if}
3 $\mathcal{T}_{sel_{ij}} \leftarrow c_{if}$
4 **if** $GetLocalActionwithPrecondition(c_{if}) \neq \emptyset$ **then**
5 $c_{ik} = GetPreconditionforAction(A_k)$
6 **for** c_{ikp} *in* c_{ik} **do**
7 $\mathcal{T}_{seq_{ij}} \rightarrow Sequence(\mathcal{T}_{seq_{ij}}, c_{ikp})$
// sequence BT with the condition of action
8 **end**
9 $\mathcal{T}_{seq_{ij}} \leftarrow Sequence(\mathcal{T}_{seq_{ij}}, T_j\_A_k\_time)$
// $T_j\_A_k\_time$ is true if $clk == \nabla(t_{io}, T_j, A_k)$)
10 $\mathcal{T}_{seq_{ij}} \leftarrow Sequence(\mathcal{T}_{seq_{ij}}, A_k)$
// Generate a sequence subtree containing action A_k and its preconditions
11 $\mathcal{T}_{sel_{ij}} \leftarrow Selector(\mathcal{T}_{sel_{ij}}, \mathcal{T}_{seq_{ij}})$
12 $\hat{\nabla}(t_{io}, T_j, A_k)$ // t_{io} is set to 1 for $\triangle t_{ik}$ time units
13 set $\hat{x}_{ijk}$ = 1 // Action A_k of T_j is assigned to R_i
14 **end**
15 **else**
16 $AuctionModule\_L2(c_{if})$
//If there is no action to meet a condition, initialize the *Level IIAuction Module* for delegation
17 **end**
18 $\mathcal{T}_{ij} \leftarrow Substitute(\mathcal{T}_{ij}, c_{if}, \mathcal{T}_{sel_{ij}})$
// add the subtree $\mathcal{T}_{sel_{ij}}$ to $\mathcal{T}_{ij}$ replacing c_{if}
19 return $\mathcal{T}_{ij}$, $\mathcal{T}_{sel_{ij}}$

Algorithm 5: *Level II Auctioning Module for R_i*

1 function AuctioningModule_L2 (c_{if}) ;
Input : c_{if} : *condition for the delegated action*
Output: $Selected_f$: *selected agent information*
$\hat{F}_D(ijf)$: *cost of the delegated action*
2 **if** ($DelegatedAgentSelected(c_{if}) == False$) **then**
3 $Selected_f \leftarrow \emptyset$
// The agent selected to resolve the condition c_{if}
4 $Announcing(c_{if})$
//broadcasting action in the communication range
5 s = $ReceiveSubmission()$
// agents with action A_f replies
6 $Selected_f$ = $Selection\_L2(s)$
// choose agent d that minimizes cost $\hat{F}_D(ijf)$
7 $UpdateDelegatedAgent(Selected_f, c_{if})$
8 **end**
9 **if** $x_{ij} == 1$ **then**
10 $Contract\_Agent(DelegatedAgent(c_{if}), c_{if})$
11 **end**
12 **else**
13 return $Selected_f, \hat{F}_D(ijf)$
14 **end**

agent with minimum bid. Then, to keep track of selected agent-action pairs, the function $UpdateDelegatedAgent(Selected_f, c_{if})$ modifies the global variable $DelegatedAgents_{ij}$ (Line 7). Further, the variable $Selected_f$ and the cost of the action, $\hat{F}_D(ijf)$, are returned (Lines 12–14) to the caller function, that are used for the estimation of cost in Algorithm 7.

When the level-one auction for the task T_j is concluded and R_i is assigned to handle T_j, again Algorithm 5 is called to make a contract for the delegated action to meet c_{if}. In this case, where Task T_j is assigned to R_i ($x_{ij} == 1$), since the bidder for c_{if} is already known ($DelegatedAgentSelected(c_{if}) \neq \emptyset$), the auction will be concluded with a contract (Lines 9–11) where the function $Contract\_Agent(.)$ informs the selected agent about the action delegation.

Algorithm 6: *Level I Auctioning Module for Mission*

1 function AuctioningModule_L1 (T_j) ;
Input : T_j : *task to be auctioned*
2 $selected_i \leftarrow \emptyset$ // The agent selected to handle the task T_j
3 $Announcing(C_j)$ // Broadcasting the condition C_j for task T_j
4 s = $ReceiveSubmission()$
// Agents with the specified action replies
5 $selected_i = Selection\_L1(s)$
// choose the agent that minimizes cost and duration
6 $Contract\_MC(selected_i, T_j)$

In Algorithm 3, we had assumed that the task T_j is assigned to the robot R_i. We now can discuss the task assignment, which is done in Algorithm 6. First, Algorithm 6 announces the task T_j in Line 3. In Line 4, the algorithm receives the estimated operational and duration cost returned by the participating agents, f_{ij} and t_{ij}, from Algorithm 8. Then, by calling the function *Selection_L*1, Algorithm 6 conducts the level-one auctioning in (2) to find a suitable agent to assign a task (Line 5). Then, in Line 6, the auction is concluded by activating the contract module, Algorithm 9, of the selected agent and appointing the agent $selected_i$ as the handler and coordinator of the task T_j (Here the function *Contract_MC* basically notifies the selected agent about the task assignment).

Algorithm 7: *Bidder Module For R_i*

1 function Biddermodule() ;
Input : c : *task or action condition*
2 **while** *True* **do**
3 c = $listen()$ // Listen to broadcasted tasks or actions
4 **if** $c \neq Null$ *and* $(\exists\ A_k \in LocalAction(R_i)\ s.t.\ A_k \models c)$ **then**
5 **if** $c.isatask()$ **then**
6 $F_{ij} = EstimateTaskCost(c)$ // call Algorithm 8
7 $submit(F_{ij})$
// Submit cost for level one auctioning, i.e., for task auctioning
8 **end**
9 **if** $c.isanaction()$ **then**
10 $\delta\hat{t}_{ik} = \nabla(t_{io}, T_j, A_k) - clk(T_j, A_k) + \triangle\hat{t}_{ik}$
11 $\hat{F}_{ik} = \alpha\hat{f}_{ik} + \beta\delta\hat{t}_{ik}$ // cost of action A_k
12 $Bidder_{ik} = (i, A_k, Pre(A_k), \triangle\hat{t}_{ik}, \nabla(t_{io}, T_j, A_k)$
13 $submit(Bidder_{ik}, \hat{F}_{ik})$)
// Submit cost for level-two auctioning
14 **end**
15 **end**
16 **end**

In response to level-one/level-two task/action announcement, Algorithm 7 performs biding operation. The algorithm continuously listens to announced tasks or actions from the *Mission Controller* (*MC*) or other agents (Line 3). If the robot has an action that satisfies the condition for an announced task, then it calls Algorithm 8 to estimate the cost of the task and submits the estimated cost (Lines 5–8). Otherwise, if the robot has an action that can satisfy an announced action, then the cost of the action is calculated and submitted (Lines 9–14). Here, the variable $Bidder_{ik}$ contains parameters that are useful for cost estimation and auctioning such as the preconditions of action A_k, i.e., $Pre(A_k)$ and the time that action A_k is available, i.e., $\nabla(t_{io}, T_j, A_k)$ (see Line 12).

Algorithm 8: *Estimate Task Cost For R_i*

```
1  function EstimateTaskCost (C_j);
   Input  : C_j : Task goal
   Output: F_ij : Estimated task cost
2  con = C_j, F_ij = 0
3  B ← GetLocalActionToMeetCondition(con)
4  do
5      for A_k in B do
6          C_{A_k} ← GetPreconditionForAction(A_k)
7      end
8      B ← ∅
9      for C_{A_kp} in C_{A_k} do
10         A_k ← GetLocalActionToMeetCondition(C_{A_kp})
11         if A_k == NULL then
12             Bidder_ik, F̂_D(ijk) ← AuctionModule_L2(con)
               // Level-two auctioning
13             F_ij = F_ij + F̂_D(ijk)
14         end
15         else
16             F_ij = F_ij + (α f̂_ik + β δ t̂_ik)
17         end
18         B ← B ∪ (A_k or Bidder_ik.A_k)
19     end
20 while (⋁(¬(R_i ⊨ C_{A_k})));
21 return F_ij
```

Called by Algorithm 7, Algorithm 8 estimates the total cost of a task. This algorithm starts by setting the condition, *con*, to the input C_j, the cost F_{ij} to 0 (Line 2), and the actions to meet C_j to variable B (Line 3). Algorithm 8 calculates C_{A_k} as the set of preconditions that have to be satisfied before executing actions in B (Lines 5–7). Then, in a *do while* loop, as long as R_i does not satisfy all $A_k \in B$ (checked in Line 20), actions to meet the preconditions $C_{A_{kp}}$ are identified either from R_i's local action bank by using the function *GetLocalActionToMeetCondition*(.) (Line 10) or via an auctioning mechanism (Line 11–14). At the same time, the set B initialized to $\emptyset$ (Line 8) is used to collect sets of actions for the next iteration (Line 18). Computing and updating the cost values for local actions in Line 13 and delegated actions in Line 16 will continue in a loop (Lines 4–20), until the satisfaction operation $\bigvee(\neg(R_i \models C_{A_k}))$ is False (Line 20), which implies that all preconditions can be met by the set of agents to execute the task T_j. After the loop terminates, the algorithm returns the total estimated cost of the task (Line 21).

Algorithm 9: *Contract Module For R_i*

```
1  function Contract () ;
2  while True do
3      c = listen()
4      if c.selected == i then
5          if c.isaction() then
6              V̂(t_io, T_j, A_k)  // t_io is set for △t_ik time units
7              set x̂_ijk = 1 // A_k of T_j is assigned to R_i
8              con = Ĉ_k  // the condition for meeting A_k
9          end
10         if c.isatask() then
11             set x_ij = 1 // The task T_j is assigned to R_i
12             con = C_j  // the condition for meeting T_j
13         end
14         AddBT(con)
               // start BT synthesis for con = C_j or con = Ĉ_k
15     end
16 end
```

Algorithm 9 continuously listens to broadcasted messages (Line 3) and forms a contract among agents, by updating task/action assignment variables, in response to level-one and level-two auctioning (Lines 2–16). If the assigned agent index matches the robots index (Line 4) and the received message is an action assignment (Line 5), then the agent timeline, t_{io}, and action assignment indicator variables, $\hat{x}_{ijk}$, are updated (Lines 5–9). Otherwise, if the received message is a task assignment, the task assignment indicator variable is set to 1, $x_{ij} = 1$. Setting x_{ij} implies that the agent R_i is a handler for the task T_j and hence acts as an auctioneer for all delegated actions required to meet C_j. Then, the function *AddBT*(*con*) in Algorithm 3 is called to generate the appropriate BT, either for a task T_j or action A_k (Line 14).

The overall flow of the algorithms from task announcement to the synthesis and execution of the BT to satisfy the mission goals is shown in Fig. 4. As shown in Fig. 4, the Mission Controller starts announcing a task (Algorithm 6) with condition C_j, where agents with the appropriate local action to handled the task perform a bidding operation (Algorithm 7). The bidding operation mainly consists of estimating the cost of tasks and their actions (Algorithm 8). Based on the estimated cost the Mission Controller selects the agent with the minimum cost and executes a contract phase (Algorithm 9). The contact process in Algorithm 9 calls for Algorithm 2 that initializes a BT with the goal condition C_j. Then, in the selected agent, Algorithm 3 synthesizes and executes the BT, in which actions are appropriately sequenced to meet C_j. These actions can either be executed immediately or expanded further in situations where the preconditions of A_k are not met. Thus, Algorithm 4 expands the preconditions of an action A_k, c_{ik}, and identifies appropriate actions, $A_*, * \neq k$, to satisfy c_{ikp} from the R_i's local action bank or via a second level auctioning (Algorithm 5).

3.3. Correctness of the proposed approach

In this section, we first show that the proposed algorithm is conflict-free. By conflict-free, we mean that each action, in the generated BT, does not violate the preconditions of an immediate or subsequent successor action. Then we show that the process for generation of BT takes a finite-time and the generated BT is live-lock free. Finally, we prove the developed algorithms solve Problem 1.

Lemma 1. *Under Assumptions 3 and 6, conflict-free BTs always exist which accomplish the goal of a task T_j.*

Proof. Based on Assumption 3, there always exists a sequence of actions to satisfy a task T_j. These sequences of actions in the form of sub-trees (each sub-tree contains the action with its preconditions) are added one after the other to generate the BT. With this approach, including a new sub-tree may conflict with one of the existing sub-trees preconditions. Considering the worst-case scenario, where adding an action always generate a conflict, we show that the conflict can be handled by Algorithms 3–9. Lines 13–15 of Algorithm 3, repeatedly increase the priority of the conflicting sub-trees to resolve conflict. Further, imagine that the robot R_i was executing the action A_t toward the accomplishment of the task $T_{j'}$. Now, the task T_j which has a higher priority as it had been announced earlier ($j < j'$) requires the robot R_i to accomplish the action A_k which conflicts with A_t. In this case, A_t has to be undone and delayed until the completion of A_k, as described in Lines 8–10 of Algorithm 3. Assumption 6 ensures that all actions can be undone to resolve a conflict. This conflict-free mechanism is postulated below:

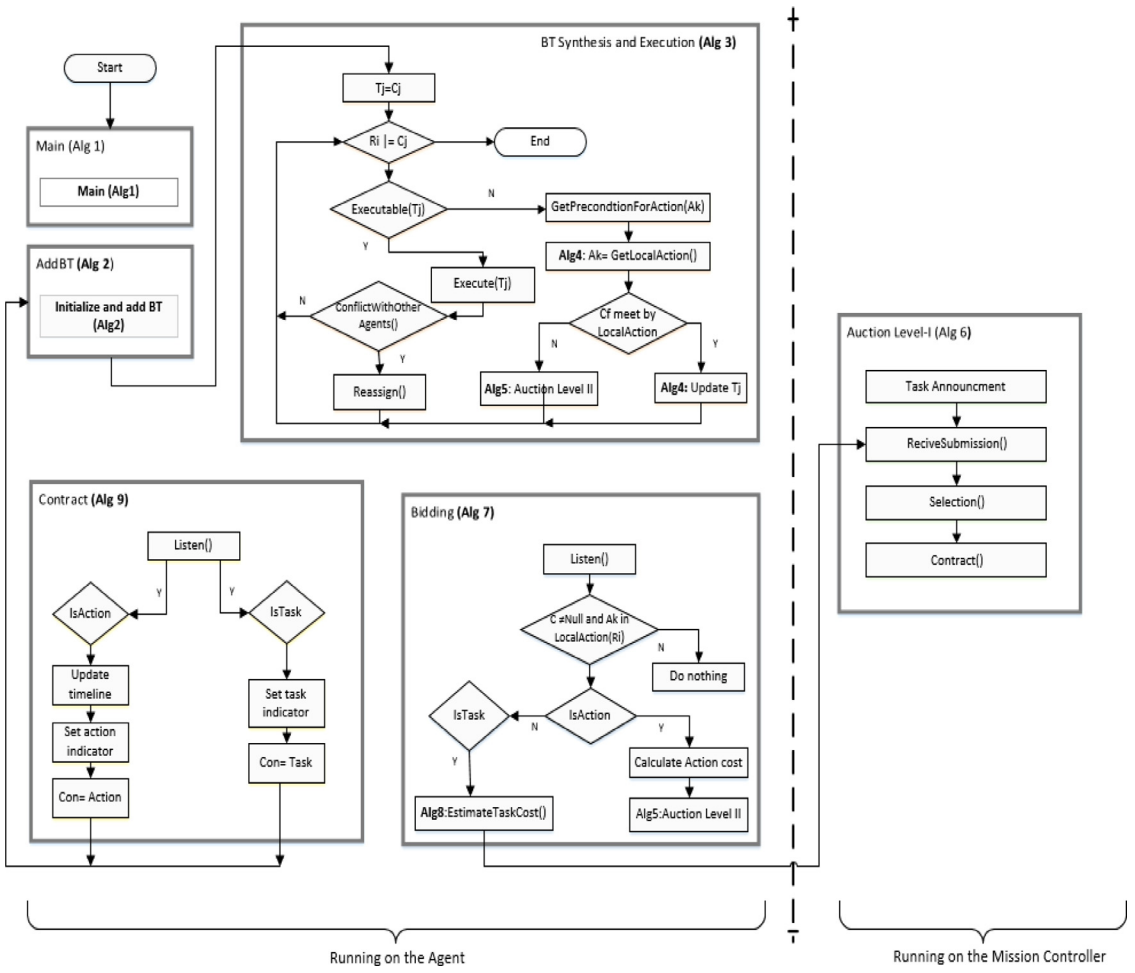

Fig. 4. A flowchart for Algorithms 1–9. The boxes on the left side of the blue broken line represent algorithms running on the agent R_i, while the orange solid box on the right of the blue line represents algorithm running on the Mission Controller.

Premise 1 $\langle$ Adding $A_k\rangle \rightarrow \langle$ Conflict with $\hat{c}_{ijt}, t \neq k\rangle$
Premise 2 $\langle$ Conflict with $\hat{c}_{ijt}$ for action $A_t, t \neq k\rangle \rightarrow$
$\langle$ Undo A_t and Increase priority of A_k
or delay $A_t\rangle$
Premise 3 Repeat $\langle$Increase priority of $A_k\rangle \rightarrow$
$\langle$Resolve conflict with all actions $A_t, t \neq k\rangle$
$\langle$Adding $A_k\rangle \rightarrow$
$\langle$ Resolve conflict with all $A_t, t \neq k$ $\rangle$

Now, since based on Assumption 3 any task T_j can be accomplished by a series of actions in the action bank, and for all actions, as stated above, it is possible to resolve the conflict with all other actions in progress, then a conflict-free BT exists for coordinating the task T_j as postulated below:

Premise 1 $\langle$Assumption 3$\rangle \rightarrow$
$\langle$ $\exists$ sequence of A_k for T_j $\rangle$
Premise 2 $\langle$ Adding A_k $\rangle \rightarrow$
$\langle$Resolve conflict with all $A_t, t \neq k\rangle$
$\langle$Premises 1, 2 $\rangle \rightarrow$
$\langle$Conflict-free BTs exit for T_j $\rangle$

■

Lemma 2. *Under Assumptions 3 and 5, a deadlock-free BTs always exist which accomplish the goal of a task T_j.*

Proof. A deadlock occurs if an action A_i wait for the completion of another action, let us say A_j. On the other hand, assume that the completion of A_j needs A_i to be completed first. This phenomenon creates a dependency among the actions which block the entire process resulting in a non-progressing BT, always waiting for the completion of each other. Assumption 5 prevents the situation that two actions are dependent on each other. Therefore, based on Assumptions 3 and 5 a sequence of actions to accomplish a task T_j exists, which is deadlock-free. This is postulated as:

Premise 1 $\langle$Assumption 3$\rangle \rightarrow$
$\langle$ $\exists$ sequence of A_k for T_j $\rangle$
Premise 2 $\langle$Assumption 5$\rangle \rightarrow$
$\langle$All actions, A_k, are deadlock-free$\rangle$
$\langle$Premise 1, 2 $\rangle \rightarrow$
$\langle$Deadlock-free BTs exit for T_j $\rangle$

■

Lemma 3. *Under Assumptions 3 and 4, a live-lock free BTs always exist which accomplish the goal of a task T_j.*

Proof. A live-lock occurs in a situation where the completed action, A_k, is continuously violated by the environment (being undone). Since the environment in which the agent operates is dynamic, an agent may be forced to infinitely repeat the same action as a result of the environment interference. However, according to Assumption 4, the completion of an action can be violated only a finite number of times by the environment. Thus, combining Assumptions 3 and 4, it is possible to conclude that the generated BT is live-lock free. This is postulated as:

Premise 1 ⟨Assumption 3⟩ →
⟨ ∃ sequence of A_k for T_j ⟩
Premise 2 ⟨Assumption 4⟩ →
⟨A_k can be violated a finite number of time⟩
Premise 3 ⟨A_k can be violated a finite number of time⟩ →
⟨Action $A_k$is live-lock free⟩
⟨Premises 1, 2, 3 ⟩ →
⟨Live-lock free BTs exist for T_j ⟩

■

Lemma 4. *Under Assumptions 3 and 7, as well as Lemmas 2 and 4, there exist BTs that can accomplish a task T_j in a finite-time.*

Proof. Based on Assumption 3, for each task, T_j, there exists a sequence of actions to meet its goal, C_j. Also, based on Assumption 7, for each action A_k, there is at least one agent. However, some of the actions in T_j may not be accomplished at the time the action is required either because the agent which is assigned to handle the task does not have a local action, A_k, or other agents capable of accomplishing action A_k are busy. We showed that all actions are deadlock-free and live-lock free, which guarantees that agents which are capable of performing action A_k will be free after a finite amount of time. Therefore, as each action can be completed in a finite-time, the BTs generated in a backward way starting from the goal of a task T_j and executed in a forward way can always be completed within a finite amount of time. This is postulated as:

Premise 1 ⟨Assumption 3⟩ →
⟨ ∃ sequence of A_k for T_j ⟩
Premise 2 ⟨Lemma 2⟩ →
⟨ Deadlock-free BTs for T_j ⟩
Premise 3 ⟨Lemma 3⟩ →
⟨Live-lock free BTs for T_j ⟩
Premise 4 ⟨Assumption 7⟩ →
⟨$\exists R_i$ for action A_k⟩
⟨Premises 1, 2, 3, 4 ⟩ →
⟨Finite time BTs for T_j ⟩

■

Theorem 1. *The proposed decentralized BT synthesis and execution approach, described by Algorithms 1–9, addresses Problem 1.*

Proof. Given Lemmas 1–4, we show that there exist BTs for coordinating a group of robots to accomplish a mission which consists of multiple tasks as described in problem 2.

According to Lemmas 1, 2, 3, and 4, the generated BTs for each task is conflict-free, deadlock-free, and live-lock free, which can be executed in finite-time. Also, we showed that conflict within or among tasks can be resolved either by increasing the priority of actions, A_k, or delaying agents executing a lower priority task. Thus, it is always possible to guarantee that Algorithms 1–9, addresses the coordination of multi-robots to meet the goals of several tasks, in any given order (mission). This is postulated as:

Premise 1 ⟨Lemma 1⟩ →
⟨Conflict-free BTs exits for all T_j ⟩
Premise 2 ⟨Lemma 2⟩ →
⟨ Deadlock-free BTs for all T_j⟩
Premise 3 ⟨Lemma 3⟩ →
⟨Live-lock free BTs for all T_j⟩
Premise 4 ⟨Lemma 4⟩ →
⟨Finite-time BTs for all T_j⟩
⟨Premises 1, 2, 3, 4 ⟩ →
⟨Problem 1 is addressed for all T_j⟩

■

Table 1
Action bank for the UAV in case study 5.1.

Global Action Bank No	Action	Precondition	Effect
1	MoveTo(Np, $path$)	*path is collisionfree*	*uav at Np*
2	Detect(m)	–	*m is detected*
3	Deliver(o, p)	*uav at Np m is detected*	*o at p*

4. Complexity of the proposed algorithm

The complexity of the proposed framework can be investigated by analyzing the complexity of synthesis and execution processes. The contributions of other parts of the framework such as the initialization steps (Algorithm 1) and the BT invoking routine (Algorithm 2) to the complexity analysis are not significant, and hence, are ignored.

For the synthesis part, Algorithm 3 (Lines 11–15) starts by identifying unsatisfied preconditions of an action, with a computational requirement of $O(1)$ (Here, we assume a lookup table and a library of sub-trees containing the preconditions and actions is already available). Then, using Algorithm 5, assigning an action to the right agent among M agents at the worst case requires $O(M)$ computation operations when the agent plays the auctioneer role (compared to $O(1)$ computation operation when the agent has the role of bidder). This is followed by a conflict resolution mechanism which requires $O(n)$ computation operations to check if the assigned actions conflict with maximum $n-1$ previous actions. As a result, to synthesize a single action the auctioneer requires $O(M+n)$ operations. Consequently, for synthesizing a task with n actions, the synthesis procedure requires $O(Mn+n^2)$ computation operations.

Regarding to the execution of a BT, as can be seen in Algorithm 3 (Lines 4–10), for a task with n actions, the execution stage has a complexity of $O(n)$, as the loop should be repeated for n actions.

Therefore, the complexity of the whole process is dominated by the complexity of the synthesis part, which is $O(Mn+n^2)$.

5. Case studies

5.1. Single agent single task: Search and delivery UAV mission

Consider a UAV whose mission objective is to deliver an object o to a specific place marked by m near position p. The UAV has to search for the marking m in the close vicinity of p, N_p, before delivering the object o. Then, the problem is to generate a BT for the UAV with the action capabilities listed in Table 1. Since it is a single agent, no bidding mechanism is needed and it is sufficient to use Algorithms 3 and 4 to achieve this search and delivery task.

Algorithm 3 starts from the goal, "o at p", i.e., the object o should be at position p, as shown in Fig. 5a. Since initially the goal is not satisfied yet and the generated *BT* is not executable, the function *GetConditionsToExpand* is called to identify the preconditions (Line 11 of Algorithm 3). From Table 1, the *Deliver* action can meet the condition "*oatp*", and hence the *ExpandBt* function (Line 12 of Algorithm 3) uses this action to update the BT by composing the conditions of *Deliver* action via a sequence node and the goal by a selector node (Lines 4–14 of Algorithm 4) as shown in Fig. 5b. Again since the preconditions, *uav at* N_p and *m is detected*, are not true, they have to be expanded, following the same procedure, by their corresponding actions *MoveTo* and *Detect* as shown in Fig. 5c.

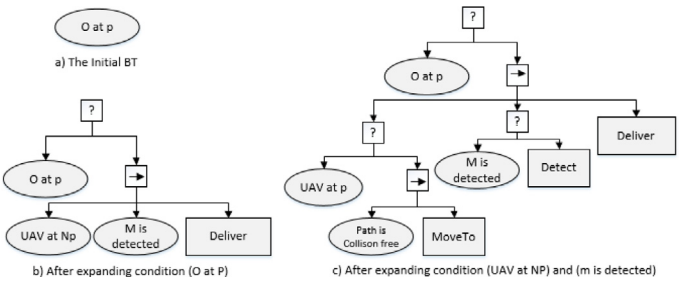

Fig. 5. Synthesizing a BT for a UAV to search and deliver an object to a particular position: (a) Algorithm 3 starts with generating a BT with the goal condition "*o at p*", (b) Algorithm 4 finds out that the action "*Deliver*" can meet the goal condition, and hence, the action "*Deliver*" and its preconditions ("*being at Np*" and "*detecting m*") are added as a subtree with a selector node, (c) Algorithm 4 expands the false preconditions ("*being at Np*" and "*detecting m*") to find the actions that can meet these preconditions.

Table 2
Operational and duration cost of an action. The actions *MoveTo* and *Reconnaissance* are functions of distance divided by velocity.

Cost of action			
Action	Decription	Operation	Duration
A_1	*MoveTo*	α_{i1}*d(s)	α_{i1}*d(s)/v
A_2	*Recon*	α_{i2}*d(s)	α_{i2}*d(s)/v+M
A_3	*Pick*	4	2
A_4	*Attack*	10	2
A_5	*Deliver*	4	2
A_8	*Repair*	5	8

Table 3
Agents capabilities.

Resource		
No	Agent	Agent capability
1	UAV_1	$\{A_1, A_4\}$
2	UAV_2	$\{A_5, A_1, A_8\}$
3	UAV_3	$\{A_5, A_1, A_3\}$
4	UAV_4	$\{A_1, A_2\}$
5	UAV_5	$\{A_1, A_2\}$

Table 4
Five tasks expanded using Algorithm 3.

Mission		
Task	goal	Sequence of actions
T_1	C_1	*Recon*(s_1), *MoveTo*(s'_1), *attack*(s'_1)
T_2	C_2	*Recon*(s_2), *MoveTo*(s'_2), *Deliver*
T_3	C_3	*MoveTo*(s_3), *Pick*, *MoveTo*(s'_3), *Deliver*
T_4	C_4	*MoveTo*(s_4), *Repair*
T_5	C_5	*MoveTo*(s_5), *attack*(s'_1)

5.2. Multiple agent multiple task

Consider a heterogeneous team of agents $R = \{R_1, R_2, R_3, R_4, R_5\}$ with the maximum velocity of $V = \{20, 20, 20, 20, 20\}$ [m/s], initially located at $P_R = \{(60, 60), (54, 24), (90, 53), (25, 40), (75, 85)\}$ [m]. The capabilities of these UAVs are described by the action bank in Table 3 where each action has an operational and duration cost associated with it as shown in Table 2. The cost for actions *MoveTo* and *Reconnaissance* is in the form of $\alpha_{ik}d(s)/v$, where $d(s)$ is the distance to be traveled by the agent i, v is its velocity, and α_{ik} is a proportional term that depends on the type of agent. In this section, we have $\alpha_{1k} = \alpha_{2k} = \alpha_{3k} = 1.2$ and $\alpha_{4k} = \alpha_{5k} = 1$. For *Recon*, the UAV should travel to a particular location and conduct reconnaissance to find a target. Therefore, since the exact location of the target is not known beforehand, after reaching the search location, an additional duration cost is considered for searching a target, which is assumed to have a maximum value, M (in this section we let M=10). For simplicity, the actions *Pick*, *Attack*, *Repair* and *Deliver* are assumed to have fixed costs.

Our aim is to synthesize and execute BTs in a decentralized way to accomplish the tasks T_1, T_2, T_3, T_4, and T_5, which are announced at time instances 1, 3, 4, 6 and 13, and located at $P_T = (15, 60), (80, 20), (60, 90), (20, 20), (50, 80)[m]$, as listed in Table 4. The sequence of actions for each task is given in Column 4 of Table 4 (expansion of a task to a sequence of actions is explained in Section 5.1) so that the last action meets the task condition C_j. Then, we consider two scenarios in an environment with an area of $10,000m^2$ and will follow Algorithms 1–9 to generate BTs.

B.1. Scenario 1: In the first scenario, we aim to complete the tasks in Table 4, where the duration is of higher importance by setting $\alpha = 0.6$ and $\beta = 0.4$, assigning more weight to the operational costs of the tasks and a lower weight to the duration of the tasks in the cost function given in (2). Further, the location of a task in Table 4 is used as an argument for the actions, e.g., the location of T_4 ($s_4 = [25, 40]$) is given as an argument for the action *MoveTo*. Then, we follow Algorithms 1–9 to generate BTs for each task, T_j, for which the details of the generated BTs are provided in Table 5. For example, consider the expanded task T_1 with the sequence *Reccon*, *MoveTo*, *Attack* (Row 1 of Table 5), where UAV_1 is the winner of T_1 (since only UAV_1 can do *Attack* as the last action of T_1). To complete task T_1, UAV_1 can perform actions *MoveTo* and *Attack*, but it does not have the capability to perform action *Reccon*. Hence, UAV_1 initiates a level-two auction (Lines 2–8 of Algorithm 5) to assign action *Reccon* (Row 2 of Table 5), for which using (6), UAV_4 wins the auction with minimum cost (the duration cost for UAV_4 to accomplish *Reccon* is $\delta\hat{t}_{42} = 12$ (using Table 2 the duration cost is $\alpha_{42}*\sqrt{(P_{T_1} - P_{R_4})}/v_4 + M = \sqrt{(25-15)^2 + (40-60)^2}/20 + 10 = 12$) and its operation cost is $\hat{f}_{42} = \alpha_{42} * \sqrt{(P_{T_1} - P_{R_4})} = (\sqrt{(25-15)^2 + (40-60)^2} = 22.4$), resulting in a total cost of $\hat{F}_{42} = \alpha\hat{f}_{42} + \beta\delta\hat{t}_{52} = 0.4 * 22.4 + 0.6 * 12 = 18.2$ (Line 6 of Algorithm 5). Therefore, using (4), the duration cost of T_1, being handled by UAV_1, is $t_{11} = \delta\hat{t}_D(112) + \delta\hat{t}_{12} + \delta\hat{t}_{13} = 18$, where $\delta\hat{t}_D(112) = \delta\hat{t}_{42} = 12$. Similarly, using (3), the operation cost of T_1, being handled by UAV_1, is $f_{11} = \hat{f}_D(112) + \hat{f}_{12} + \hat{f}_{13} = 88.3$, where $\hat{f}_D(112) = \hat{f}_{42} = 22.4$. Using (1) and (2), task T_1 can be concluded with the total cost of $F_1 = F_{11} = \alpha f_{11} + \beta t_{11} = 0.4 * 88.3 + 0.6 * 18 = 60.2$. Further, as detailed in Row 3 of Table 5, upon finalizing the contract, UAV_1 will be assigned to handle T_1 and accomplish actions *MoveTo* and *attack* over $\hat{\nabla}(t_{1o}, T_1, A_1) = [14, 17)$ and $\hat{\nabla}(t_{1o}, T_1, A_4) = [17, 19)$, respectively, while delegating UAV_4 to accomplish *Reccon* over $\hat{\nabla}(t_{5o}, T_1, A_2) = [2, 14)$, ensuring that task T_1 will be completed in the time range from [2, 19). Assignment of all tasks follows the same procedure.

Since task T_1 is the first task, the delay in the calculation of durational cost is 0, i.e., using (5), $\nabla(t_{*o}, T_1, A_*) - clk(T_1, A_*) = 0$.

Table 5
Task assignment for Scenario B.1 ($\alpha = 0.6, \beta = 0.4$) : "$MC,\ L1$" represents the level-one auctioning by the Mission Controller, and "$UAV_*, L2$" represents the level-two auctioning by UAV_*.

Auctioning steps for assigning the tasks $T_1, \cdots, T_5$					
Step	Auctioneer	Task/Action	Bidder/Candidate	Submission	Selection/Contract
1	$MC, L1$	T_1	UAV_1		
2	$UAV_1, L2$	A_2	UAV_4, UAV_5	$\{UAV_4 : \delta\hat{t}_{42} = 12, \hat{f}_{23} = 22.4, \hat{F}_{42} = \alpha\hat{f}_{42} + \beta\hat{t}_{42} = 18.6\}$ $\{UAV_5 : \delta\hat{t}_{52} = 14, \hat{f}_{52} = 65, \hat{F}_{52} = \alpha\hat{f}_{52} + \beta\hat{t}_{52} = 44.6\}$	$UAV_4 : \delta\hat{t}_D(112) = 14, \hat{f}_D(112) = 22.4, \hat{F}_D(112) = 18.6$
3	$MC, L1$	$T1$	UAV_1	$t_{11} = \delta\hat{t}_D(112) + \delta\hat{t}_{12} + \delta\hat{t}_{13} = 18, f_{11} = \hat{f}_D(112),$ $+\hat{f}_{12} + \hat{f}_{13} = 88.3, F_{11} = \alpha f_{11} + \beta t_{11} = 60.2$	$UAV_1 : F_1 = F_{11} = 60.2, \bar{\nabla}(t_{1o}, T_1, A_2) = [2, 14),$ $\bar{\nabla}(t_{1o}, T_1, A_1) = [14, 17), \bar{\nabla}(t_{1o}, T_1, A_4) = [17, 19)$
4	$MC, L1$	T_2	UAV_2, UAV_3		
5	$UAV_2, L2$	A_2	UAV_4, UAV_5	$\{UAV_4 : \delta\hat{t}_{42} = 23, \hat{f}_{42} = 58.5, \hat{F}_{42} = \alpha\hat{f}_{42} + \beta\hat{t}_{42} = 44.3\}$ $\{UAV_5 : \delta\hat{t}_{52} = 15, \hat{f}_{52} = 70.2, \hat{F}_{52} = \alpha\hat{f}_{52} + \beta\hat{t}_{52} = 48.1\}$	$UAV_4 : \delta\hat{t}_D(222) = 23, \hat{f}_D(222) = 58.5, \hat{F}_D(222) = 44.3$
6	$UAV_3, L2$	A_2	UAV_4, UAV_5	$\{UAV_4 : \delta\hat{t}_{42} = 23, \hat{f}_{42} = 58.5, \hat{F}_{42} = \alpha\hat{f}_{42} + \beta\hat{t}_{42} = 44.3\}$ $\{UAV_5 : \delta\hat{t}_{52} = 15, \hat{f}_{52} = 70.2, \hat{F}_{52} = \alpha\hat{f}_{52} + \beta\hat{t}_{52} = 48.9\}$	$UAV_4 : \delta\hat{t}_D(222) = 23, \hat{f}_D(222) = 58.5, \hat{F}_D(222) = 44.3$
7	$MC, L1$	T_2	UAV_2, UAV_3	$UAV_2 : t_{22} = \delta\hat{t}_D(222) + \delta\hat{t}_{21} + \delta\hat{t}_{25} = 29,$ $f_{22} = \hat{f}_D(222) + \hat{f}_{21} + \hat{f}_{25} = 90.3, F_{22} = \alpha f_{22} + \beta t_{22} = 84.2;$ $UAV_3 : t_{32} = \delta\hat{t}_D(322) + \delta\hat{t}_{31} + \delta\hat{t}_{35} = 17,$ $f_{32} = \hat{f}_D(332) + \hat{f}_{31} + \hat{f}_{35} = 59.7, F_{32} = \alpha f_{32} + \beta f_{32} = 65.8$	$UAV_3 : F_3 = F_{32} = 65.8, \bar{\nabla}(t_{4o}, T_2, A_2) = [14, 27),$ $\bar{\nabla}(t_{3o}, T_2, A_1) = [27, 29), \bar{\nabla}(t_{3o}, T_2, A_5) = [29, 31)$
8	$MC, L1$	T_3	UAV_3	$UAV_3 : t_{33} = \delta\hat{t}_{31} + \delta\hat{t}_{33} + \delta\hat{t}_{31} + \delta\hat{t}_{39} = 14, f_{33} = \hat{f}_{31}$ $+\hat{f}_{33} + \hat{f}_{31} + \hat{f}_{39} = 128.5, F_{33} = \alpha\hat{f}_{33} + \beta\hat{t}_{33} = 83.5$	$UAV_3 : F_3 = F_{34} = 83.5, \bar{\nabla}(t_{3o}, T_4, A_3) = [5, 13)$ $\bar{\nabla}(t_{3o}, T_3, A_3) = [13, 17), \bar{\nabla}(t_{3o}, T_3, A_5) = [17, 19)$
9	$MC, L1$	T_4	UAV_2	$UAV_2 : t_{24} = \delta\hat{t}_{21} + \delta\hat{t}_{28} = 4, f_{24} = \hat{f}_{21} + \hat{f}_{28} = 59.1,$ $F_{24} = \alpha\hat{f}_{24} + \beta\hat{t}_{24} = 39.5$	$UAV_2 : F_4 = F_{24} = 39.5, \bar{\nabla}(t_{2o}, T_4, A_1) = [7, 9)$ $\bar{\nabla}(t_{2o}, T_4, A_8) = [9, 11)$
10	$MC, L1$	T_5	UAV_1	$UAV_1 : t_{15} = \delta\hat{t}_{11} + \delta\hat{t}_{14} = 13, f_{15} = \hat{f}_{11} + \hat{f}_{11} = 93.6,$ $F_{15} = \alpha\hat{f}_{15} + \beta\hat{t}_{15} = 61.4$	$UAV_2 : F_5 = F_{15} = 61.4, \bar{\nabla}(t_{1o}, T_5, A_1) = [19, 23)$ $\bar{\nabla}(t_{1o}, T_5, A_4) = [23, 25)$

Table 6
Task assignment in Scenario B.2 ($\alpha = 0.4, \beta = 0.6$) : "$MC,\ L1$" represents the level-one auctioning by the Mission Controller and "$UAV_*, L2$" represents the level-two auctioning by UAV_*.

Auctioning steps for assigning the tasks $T_1, \cdots, T_5$					
Step	Auctioneer	Task/Action	Bidder/Candidate	Submission	Selection/Contract
1	$MC, L1$	T_1	UAV_1		
2	$UAV_1, L2$	A_3	UAV_1, UAV_2	$\{UAV_2 : \delta\hat{t}_{23} = 2, \hat{f}_{23} = 0.5, \hat{F}_{23} = \alpha\hat{f}_{23} + \beta\hat{t}_{23} = 1.4\}$ $\{UAV_3 : \delta\hat{t}_{33} = 1, \hat{f}_{33} = 0.7, \hat{F}_{33} = \alpha\hat{f}_{33} + \beta\hat{t}_{33} = 0.88\}$	$UAV_3 : \delta\hat{t}_D(113) = 1, \hat{f}_D(113) = 0.7, \hat{F}_D(113) = 0.88$
3	$MC, L1$	T_1	UAV_1	$t_{11} = \delta\hat{t}_{11} + \delta\hat{t}_D(113) + \delta\hat{t}_{13} = 6, f_{11} = \hat{f}_{11} + \hat{f}_D(113) + \hat{f}_{13} = 1$ $F_{11} = \alpha f_{11} + \beta t_{11} = 4$	$UAV_1 : F_1 = F_{11} = 4, \bar{\nabla}(t_{1o}, T_1, A_1) = [1, 4),$ $\bar{\nabla}(t_{3o}, T_1, A_3) = [4, 5), \bar{\nabla}(t_{1o}, T_1, A_1) = [5, 7)$
4	$MC, L1$	T_2	UAV_2, UAV_3		
5	$UAV_2, L2$	A_4	UAV_3	$\{UAV_3 : \delta\hat{t}_{34} = 1, \hat{f}_{34} = 0.7, \hat{F}_{34} = \alpha\hat{f}_{34} + \beta\hat{t}_{34} = 0.88\}$	$\{UAV_3 : \delta\hat{t}_D(224) = 1, \hat{f}_D(224) = 0.7, \hat{F}_D(224) = 0.88\}$
6	$UAV_2, L2$	A_6	UAV_3	$\{UAV_3 : \delta\hat{t}_{36} = 7, \hat{f}_{36} = 1, \hat{F}_{36} = \alpha\hat{f}_{36} + \beta\hat{t}_{36} = 4.6\}$	$\{UAV_3 : \delta\hat{t}_D(226) = 7, \hat{f}_D(226) = 1, \hat{F}_D(226) = 4.6\}$
7	$MC, L1$	T_2	UAV_2, UAV_3	$UAV_2 : t_{22} = \delta\hat{t}_D(226) + \delta\hat{t}_D(224) + \delta\hat{t}_{23} = 10,$ $f_{22} = \hat{f}_D(226) + \hat{f}_D(224) + \hat{f}_{23} = 2.2, F_{22} = \alpha f_{22} + \beta t_{22} = 6.88;$ $UAV_3 : t_{32} = \delta\hat{t}_{36} + \delta\hat{t}_{34} + \delta\hat{t}_{33} = 6,$ $f_{32} = \hat{f}_{36} + \hat{f}_{34} + \hat{f}_{33} = 2.7, F_{32} = \alpha f_{32} + \beta f_{32} = 4.56$	$UAV_3 : F_2 = F_{32} = 4.56, \bar{\nabla}(t_{3o}, T_2, A_6) = [5, 9),$ $\bar{\nabla}(t_{3o}, T_2, A_4) = [9, 10), \bar{\nabla}(t_{2o}, T_2, A_3) = [10, 11)$
8	$MC, L1$	T_3	UAV_1		
9	$UAV_1, L2$	A_5	UAV_2	$\{UAV_2 : \delta\hat{t}_{35} = 3, \hat{f}_{23} = 0.4, \hat{F}_{35} = \alpha\hat{f}_{35} + \beta\hat{t}_{35} = 1.96\}$	$\{UAV_2 : \delta\hat{t}_D(135) = 3, \hat{f}_D(135) = 0.4, \hat{F}_D(135) = 1.96\}$
10	$MC, L1$	T_3	UAV_1	$UAV_1 : t_{13} = \delta\hat{t}_D(135) + \delta\hat{t}_{13} = 7$ $f_{13} = \hat{f}_D(135) + \hat{f}_{13} = 0.5, F_{13} = \alpha f_{13} + \beta t_{13} = 4.4$	$UAV_1 : \bar{F}_3 = F_{13} = 4.4, \bar{\nabla}(t_{2o}, T_3, A_5) = [3, 5),$ $\bar{\nabla}(t_{1o}, T_3, A_1) = [7, 10)$
11	$MC, L1$	T_4	UAV_3	$UAV_3 : \delta\hat{t}_{34} = 7, \hat{f}_{34} = 0.7, F_{34} = \hat{F}_{34} = \alpha\hat{f}_{34} + \beta\hat{t}_{34} = 4.48$	$UAV_3 : F_{34} = 4.48, \bar{\nabla}(t_{3o}, T_4, A_4) = [12, 13)$
12	$MC, L1$	T_5	UAV_1, UAV_2	$UAV_1 : t_{15} = \delta\hat{t}_{12} + \delta\hat{t}_{18} = 9, f_{15} = \hat{f}_{12} + \hat{f}_{18} = 1$ $F_{15} = \alpha f_{15} + \beta f_{15} = 5.8;$ $UAV_2 : t_{25} = \delta\hat{t}_{23} + \delta\hat{t}_{27} = 5, f_{25} = \hat{f}_{23} + \hat{f}_{27} = 0.9,$ $F_{25} = \alpha f_{25} + \beta t_{25} =$	$UAV_2 : F_{25} = 3.36, \bar{\nabla}(t_{2o}, T_5, A_3) = [6, 8),$ $\bar{\nabla}(t_{1o}, T_5, A_7) = [8, 10)$

Table 7
In Scenario B.2, a mission consists of 100 tasks assigned to different number of agents.

Number of tasks	Number of agents	Computation	Duration time
100	1	3.6	5932
100	10	8.10	927
100	20	13.25	509
100	30	33.15	282
100	40	58.5	237

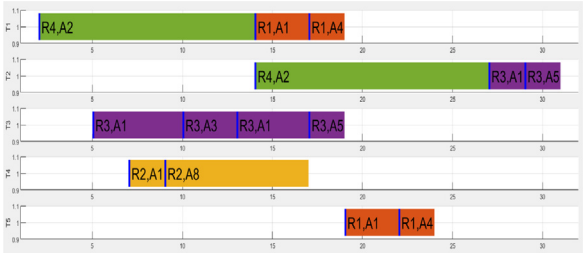

Fig. 6. Task and action assignment timeline for Scenario B.1.

This may not be the case for other tasks. For example, for task T_2 in Table 5, since there are ten time units delay between the announcement of T_2, $clk(T_2, A_2) = 4$, and the availability of UAV_4 for the execution of task T_2, $\nabla(t_{4o}, T_2, A_2) = 14$, the duration cost of A_2 is $\delta\hat{t}_{42} = \nabla(t_{4o}, T_2, A_2) - clk(T_2, A_2) + \Delta\hat{t}_{42} = 23$. Therefore, the cost of A_2 for task T_2 is $f_{42} = \alpha * \hat{f}_{42} + \beta * \delta\hat{t}_{42} = 0.6 * 58.5 + 0.4 * 23 = 44.3$). As another example, task T_5 should start with the action A_1 of UAV_1. Even though UAV_1 is available at $t = 11$ for 3 time unit to start action A_1, this is not enough to complete A_1 with $\Delta\hat{t}_{11} = 4$ due to prior contract with T_1 (the timeline for the tasks T_1, T_2, and T_3 can be seen in Fig. 6). Therefore, since the available time for UAV_1 is not sufficient to complete the action A_1 (this is checked by Line 12 of Algorithm 4 for local actions and Line 6 of Algorithm 9 for delegated actions, action A_1 is deferred to a later time as shown in Fig. 6 and Row 10 of Table 5. The corresponding execution of the task by the agents is shown in Fig. 7.

For each task, once the auctioning process is completed, the winning agent synthesizes the BT to meet the goal of the task. As an example, the synthesized BT for UAV_1 is shown in Fig. 8. The tree BT_1 is composed of five subtrees running in parallel. The left subtrees are responsible for the market based auctioning (auctioning, bidding, and contract), while the two right subtrees sequence actions appropriately to satisfy the mission goals: the top right subtree and the bottom right subtree contribute to tasks T_1 and T_5. The level-two auctioning is conducted at the agent level (the left subtrees in Fig. 8). The *Level-II Auction* subtree on the top-left announces the actions to be delegated, receives the submissions, selects the winner agent, and makes a contract by informing the selected agent about the assigned action. On the other hand, the *MissionController* (*MC*) handles level-one auctioning. For this purpose, the *Level-I Auction* subtree in Fig. 9, announces the tasks, receives the submissions, selects the winner/handler agent, and makes a contract by informing the selected agent about the assigned tasks. The *Bidder* subtree in

Table 8
Terms and their explanations.

No	Terms	Explanation
1	T	Set of tasks
2	T_j	jth task
3	N	Total number of tasks
4	A	Global action bank
5	A_k	kth action
6	L	Total number of actions
7	R	Set of robots i
8	R_i	ith Robot
9	M	Total number of agents/robots
10	x_{ij}	Task assignment indicator; $x_{ij} = 1$ if T_j is assigned to R_j
11	$\hat{x}_{ijk}$	Action assignment indicator; $\hat{x}_{ijk} = 1$ if A_K is assigned to R_1 to complete of task T_j
12	C_j	Condition to meet task T_j
13	$\hat{a}_{ik}$	Action capability indicator for R_i to handle action A_k; $\hat{a}_{ik} = 1$ if R_i can handle A_k
14	$\hat{C}$	Set of preconditions for the actions
15	$\hat{C}_k$	Condition to meet action A_k
16	p_k	Number of preconditions for action A_K
17	$\hat{a}_{ijk}$	$\hat{a}_{ijk}$ =1 if action A_k of R_i is needed for task T_j
18	c_{ik}	Set of preconditions for action A_k when being handled by R_i
19	c_{ikp}	pth precondition for action A_k when being handled by R_i
20	F	Set of value functions
21	F_{ij}	Total cost of T_j when R_i is the handler
22	F_j	The minimum total cost to complete T_j
23	f_{ij}	Operational cost of T_j when R_i is the handler
24	t_{ij}	Durational cost of T_j when R_i is the handler
25	$\hat{f}_{ik}$	Operational cost of A_k to be done by R_i
26	$\hat{t}_{ik}$	Durational cost of A_k when done by R_i
27	$\hat{F}_{ik}$	Total cost of action A_k to be done by R_i
28	$\hat{F}_D(ijk)$	The operational and duration cost of A_k delegated by R_i to complete T_j
29	$\hat{f}_D(ijk)$	The operational cost of A_k delegated by R_i to complete T_j
30	$\delta\hat{t}_D(ijk)$	The durational cost of A_k delegated by R_i to complete T_j
31	α	Operational cost weight term
32	β	Durational cost weight term
33	clk	Discrete clock
34	$clk(T_j)$	The time instant that T_j is announced
35	$clk(T_j, A_k)$	The time instant that action A_k is called for T_j
36	o	The sample time index from start to the end of mission $(1, 2, \cdots)$
37	t_{io}	Action timeline indicator for R_i; t_{io} = 1 during the time R_i is assigned an action
38	$\nabla(t_{io}, T_j, A_k)$	Returns the time instant that A_k is available to be started by R_i for task T_j
39	$\hat{\nabla}(t_{io}, T_j, A_k)$	A function that makes $t_{io} = 1$ over $[\nabla(t_{io}, T_j, A_k), \nabla(t_{io}, T_j, A_k) + \triangle\hat{t}_{ik}]$ when R_i should execute A_k
40	$\triangle\hat{t}_{ik}$	The time duration that R_i needs to complete action A_k
41	$\delta\hat{t}_{ik}$	The duration between request and completion of A_k by R_i
42	$\models$	Satisfaction operator
43	BT_i	The whole BT structure of agent R_i
44	$\mathcal{T}_{ij}$	Sub-tree of BT_i responsible for meeting task T_j
45	$\mathcal{T}_{Auction_i}$	Sub-tree of BT_i responsible for auctioning
46	$\mathcal{T}_{Bid_i}$	Sub-tree of BT_i responsible for bidding
47	$\mathcal{T}_{contract_i}$	Sub-tree of BT_i responsible for contract
48	$DelegatedAgents_{ij}$	A global variable that store the delegated agent-action pairs for task T_j

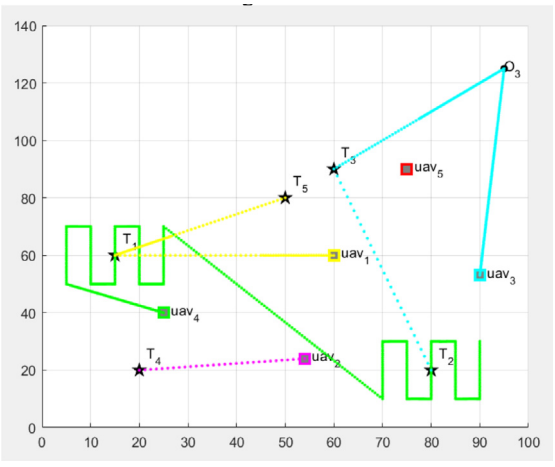

Fig. 7. Tasking and coordination of agents in Scenario B.1 where the execution of tasks by each agent is associated with a unique color. For example, the color green represents UAV_4 moving to the target location performing action *Reconnaissance* for task T_1 and T_2.

Fig. 8), submits the costs for both actions and tasks if they can be handled by UAV_1. Once UAV_1 is selected for a task by Level I or for an action by Level II auctioning, then the *contract* subtree reserves UAV_1 for the assigned task/action.

B.2. Scenario 2: In the second scenario, our aim is to complete the tasks for more agents by letting $\beta = 0.6$ and $\alpha = 0.4$ to assign more weight to the duration of the tasks in the cost function compared to their operational costs as in (2). We pursued the same procedure similar to Scenario 1 by following Algorithms 1–9, and generating decentralized BTs to coordinate the robots to complete the tasks in order. The details of the generated BTs are provided in Table 6 and is shown in Figs. 10 and 11.

B.3. Scenario 3: In the third scenario, our aim is to access the scalability and computational cost of the proposed framework. Table 7 shows the execution of 100 randomly generated tasks from Table 4, which are assigned to a different number of agents. For example, when the tasks are done using 10 agents the computation time needed for the task allocation is 8.10 *s* and the duration of execution of the tasks is 927 *s*, while using 40 agents the computation and execution time are 58.5 *s* and 237 *s*, respectively. This result shows that as the number of agents

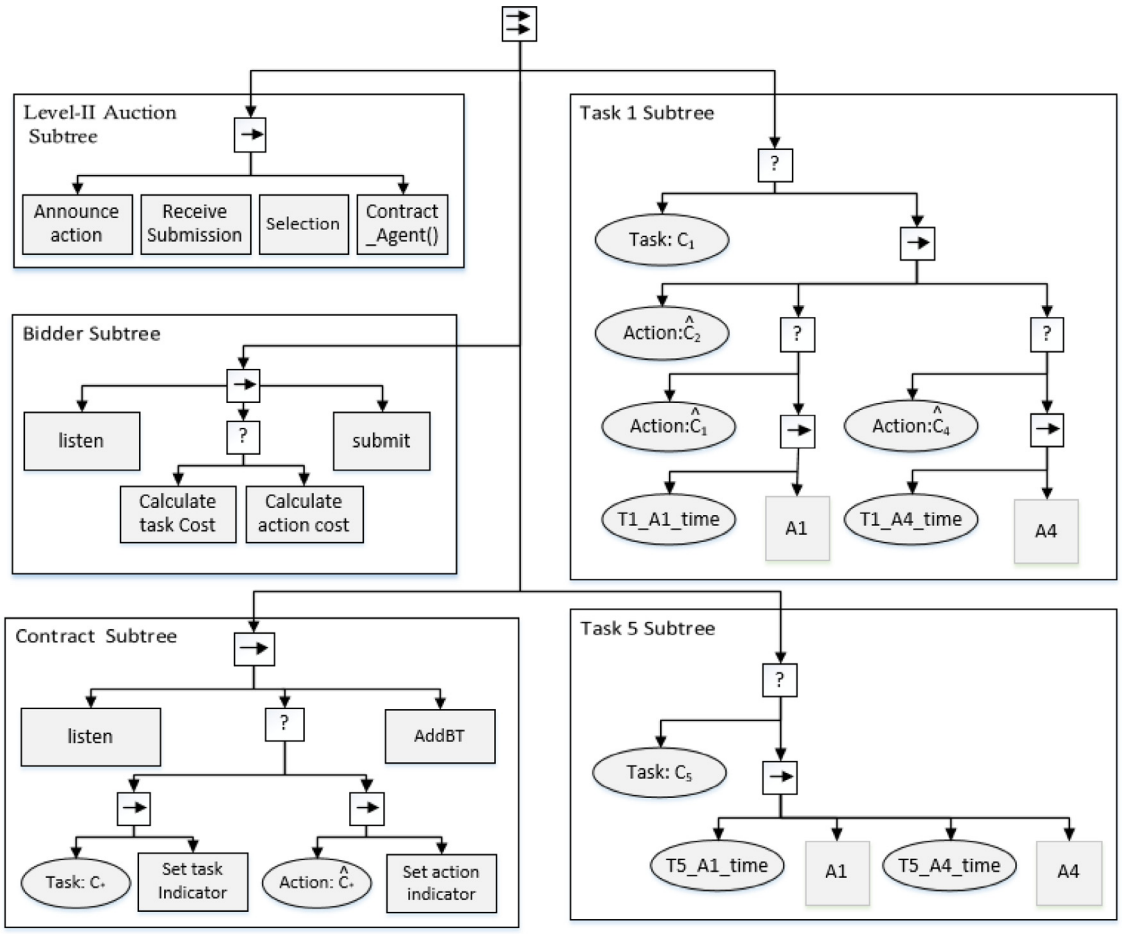

Fig. 8. The synthesized BT for UAV_1 in Scenario B.1 in Section 5.2. The sub-trees on the left are responsible for the *Market* based auctioning method on the agent side, while the sub-trees on the right are responsible for the execution of assigned tasks.

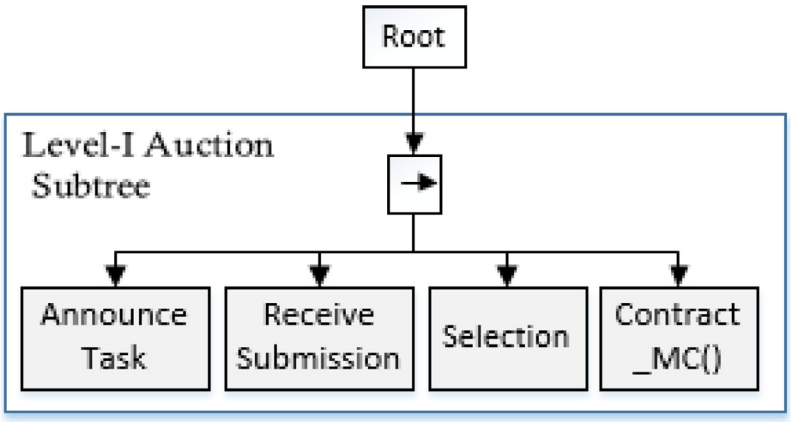

Fig. 9. The BT for the Mission Controller for Scenario B.1 in Section 5.2. This tree is responsible for the *Market* based auctioning level I.

increase, the computation cost increases while the execution time decreases as more resources (agents) are available for parallel execution of the tasks.

6. Conclusion

This paper developed a new automatic tasking approach for decentralized coordination of a heterogeneous team of autonomous agents. The agents have different capabilities in terms of executing different tasks. In the proposed framework, the collaboration can take place when an agent cannot perform a part of a mission individually but can accomplish the mission in collaboration with other agents which have complementary capabilities. For this purpose, we developed a hierarchical modular coordination approach and the required algorithms for synthesizing and executing local Behavior Trees (BTs) for tasking individual vehicles so that they can collectively achieve a set of tasks. Further, a two-level auctioning algorithm was incorporated into the developed framework to assign tasks among the vehicles with lower costs. The developed framework allows a trade-off between

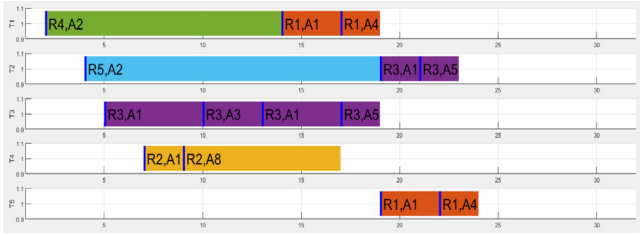

Fig. 10. Task and action assignment timeline for Scenario B.2.

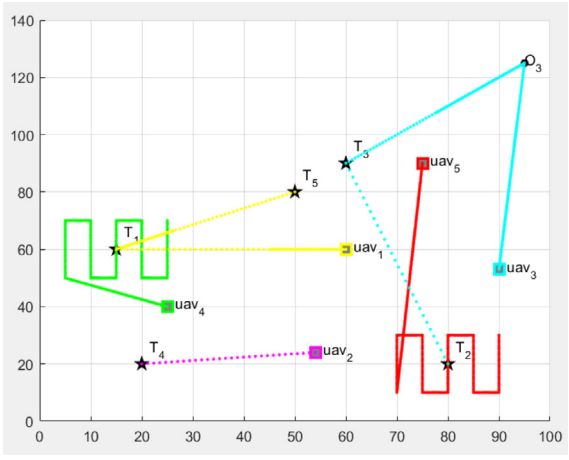

Fig. 11. Tasking and coordination of agents in Scenario B.2, where each agent execution is associated with a unique color. For example, the color green represents UAV_4 moving from the agents initial position to the target location performing action *Reconnaissance* for tasks T_1.

the total cost and the duration of the accomplishment of the tasks through an embedded cost function. Illustrative examples were provided to describe the implementation of the proposed approach. Future work includes implementing the developed framework for decentralized collaboration of multiple UAVs using Robot Operating System (ROS) as well as developing the ROS packages for the general purpose applications of this framework. Another potential future research direction is to consider the impact of uncertainty of measurements on the task allocation mechanism.

CRediT authorship contribution statement

Tadewos G. Tadewos: Roles/Writing – original draft, Software, Validation, Visualization. **Laya Shamgah:** Writing – review & editing, Software, Investigation. **Ali Karimoddini:** Methodology, Project administration, Resources, Supervision, Writing – reviewing and editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

The authors would like to acknowledge the support from NSF under the award number 1832110 and Air Force Research Laboratory and OSD for sponsoring this research under agreement number FA8750-15-2-0116.

References

[1] B. Yin, Y. Wu, T. Hu, J. Dong, Z. Jiang, An efficient collaboration and incentive mechanism for internet of vehicles (IoV) with secured information exchange based on blockchains, IEEE Internet Things J. 7 (3) (2020) 1582–1593.
[2] N. Chakraborty, Y. Chao, J. Li, S. Mishra, C. Luo, Y. He, J. Chen, Y. Pan, RTT-based rogue UAV detection in IoV networks, IEEE Internet Things J. (2021) 1.
[3] J.W. Baxter, G.S. Horn, D.P. Leivers, Fly-by-agent: Controlling a pool of UAVs via a multi-agent system, Knowl.-Based Syst. 21 (2008) 232–237.
[4] B. Ai, J. Jiang, S. Yu, Y. Jiang, Multi-agent path finding with heterogeneous edges and roundtrips, Knowl.-Based Syst. 234 (2021) 107554.
[5] K. Okumura, M. Machida, X. Défago, Y. Tamura, Priority inheritance with backtracking for iterative multi-agent path finding, Artificial Intelligence 310 (2022) 103752.
[6] Y. Chen, C. Lu, W. Chu, A cooperative driving strategy based on velocity prediction for connected vehicles with robust path-following control, IEEE Internet Things J. 7 (5) (2020) 3822–3832.
[7] J. Zhang, J. Sha, G. Han, J. Liu, Y. Qian, A cooperative-control-based underwater target escorting mechanism with multiple autonomous underwater vehicles for underwater internet of things, IEEE Internet Things J. 8 (6) (2021) 4403–4416.
[8] L. Zhou, S. Leng, Q. Liu, Q. Wang, Intelligent UAV swarm cooperation for multiple targets tracking, IEEE Internet Things J. (2021) 1.
[9] Daneshfar, Fatemeh, Bevrani, Hassan, Multi-agent systems in control engineering: A survey, J. Control Sci. Eng. (2009).
[10] A. Whitbrook, Q. Meng, P.W. Chung, Reliable, distributed scheduling and rescheduling for time-critical, multiagent systems, IEEE Trans. Autom. Sci. Eng. 15 (2) (2017) 732–747.
[11] J. Turner, Q. Meng, G. Schaefer, A. Whitbrook, A. Soltoggio, Distributed task rescheduling with time constraints for the optimization of total task allocations in a multirobot system, IEEE Trans. Cybern. 48 (9) (2017) 2583–2597.
[12] K. Vinh, S. Gebreyohannes, A. Karimoddini, An area-decomposition based approach for cooperative tasking and coordination of UAVs in a search and coverage mission, in: 2019 IEEE Aerospace Conference, 2019, pp. 1–8.
[13] P.E. Hart, N.J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Trans. Syst. Sci. Cybern. 4 (2) (1968) 100–107.
[14] W.-M. Shen, P. Will, A. Galstyan, C.-M. Chuong, Hormone-inspired self-organization and distributed control of robotic swarms, Auton. Robots 17 (1) (2004) 93–105.
[15] X. Yu, X. Ye, S. Zhang, Floating pollutant image target extraction algorithm based on immune extremum region, Digit. Signal Process. 123 (2022) 103442.
[16] Y. Quiñonez, J.d. Lope, D. Maravall, Bio-inspired decentralized self-coordination algorithms for multi-heterogeneous specialized tasks distribution in multi-robot systems, in: International Work-Conference on the Interplay Between Natural and Artificial Computation, Springer, 2011, pp. 30–39.
[17] W. Zhao, Q. Meng, P.W. Chung, A heuristic distributed task allocation method for multivehicle multitask problems and its application to search and rescue scenario, IEEE Trans. Cybern. 46 (4) (2015) 902–915.
[18] W. Saad, Z. Han, T. Basar, M. Debbah, A. Hjorungnes, Hedonic coalition formation for distributed task allocation among wireless agents, IEEE Trans. Mob. Comput. 10 (9) (2011) 1327–1344.
[19] D. Vickrey, D. Koller, Multi-agent algorithms for solving graphical games, in: AAAI/IAAI, 2002.
[20] M.B. Dias, R. Zlot, N. Kalra, A. Stentz, Market-based multirobot coordination: A survey and analysis, Proc. IEEE 94 (7) (2006) 1257–1270.
[21] M.B. Dias, R. Zlot, N. Kalra, A. Stentz, Market-based multirobot coordination: A survey and analysis, Proc. IEEE 94 (7) (2006) 1257–1270.
[22] A. Ulusoy, S.L. Smith, X.C. Ding, C. Belta, D. Rus, Optimality and robustness in multi-robot path planning with temporal logic constraints, Int. J. Robot. Res. 32 (8) (2013) 889–911.
[23] I. Filippidis, D.V. Dimarogonas, K.J. Kyriakopoulos, Decentralized multi-agent control from local LTL specifications, in: 2012 IEEE 51st IEEE Conference on Decision and Control, CDC, 2012, pp. 6235–6240.
[24] M. Karimadini, H. Lin, A. Karimoddini, Cooperative tasking for deterministic specification automata, Asian J. Control 18 (6) (2016) 2078–2087.

[25] M. Karimadini, A. Karimoddini, H. Lin, Modular cooperative tasking for multi-agent systems, in: 2018 IEEE 14th International Conference on Control and Automation, ICCA, 2018, pp. 618–623.
[26] M. Karimadini, H. Lin, Guaranteed global performance through local coordinations, Automatica 47 (5) (2011) 890–898.
[27] Y. Liu, M. Ficocelli, G. Nejat, A supervisory control method for multi-robot task allocation in Urban search and rescue, in: 2015 IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR, 2015, pp. 1–6.
[28] P. Ramadge, W. Wonham, The Control of Discrete Event Systems, Vol. 77, (1) 1989, pp. 81–98.
[29] A. Sugiyama, V. Sea, T. Sugawara, Effective task allocation by enhancing divisional cooperation in multi-agent continuous patrolling tasks, in: 2016 IEEE 28th International Conference on Tools with Artificial Intelligence, ICTAI, IEEE, 2016, pp. 33–40.
[30] P. García, P. Caamaño, R.J. Duro, F. Bellas, Scalable task assignment for heterogeneous multi-robot teams, Int. J. Adv. Robot. Syst. 10 (2) (2013) 105.
[31] H. Huang, T. Zhuo, Multi-model cooperative task assignment and path planning of multiple UCAV formation, Multimedia Tools Appl. 78 (1) (2019) 415–436.
[32] G. Wu, W. Pedrycz, H. Li, M. Ma, J. Liu, Coordinated planning of heterogeneous earth observation resources, IEEE Trans. Syst. Man Cybern. Syst. 46 (1) (2016) 109–125.
[33] X. Han, S. Mandal, K.R. Pattipati, D.L. Kleinman, M. Mishra, An optimization-based distributed planning algorithm: A blackboard-based collaborative framework, IEEE Trans. Syst. Man Cybern. Syst. 44 (6) (2014) 673–686.
[34] T.G. Tadewos, L. Shamgah, A. Karimoddini, On-the-fly decentralized tasking of autonomous vehicles, in: Proc. of 58th IEEE Conference on Decision and Control, CDC, 2019.
[35] M. Colledanchise, R.M. Murray, P. Ögren, Synthesis of correct-by-construction behavior trees, in: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, 2017, pp. 6039–6046.
[36] M. Colledanchise, P. Ögren, How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees, IEEE Trans. Robot. 33 (2) (2017) 372–389.
[37] P. Ogren, Increasing modularity of UAV control systems using computer game behavior trees, in: AIAA Guidance, Navigation and Control Conference, Minneapolis, MN, 2012.
[38] T.G. Tadewos, L. Shamgah, A. Karimoddini, Automatic safe behaviour tree synthesis for autonomous agents, in: 2019 IEEE 58th Conference on Decision and Control, CDC, 2019, pp. 2776–2781.
[39] T.G. Tadewos, A.A. Redwan Newaz, A. Karimoddini, Specification-guided behavior tree synthesis and execution for coordination of autonomous systems, Expert Syst. Appl. 201 (2022) 117022.
[40] S. Sitanskiy, L. Sebastia, E. Onaindia, Behaviour recognition of planning agents using behaviour trees, Procedia Comput. Sci. 176 (2020) 878–887.
[41] M. Colledanchise, A. Marzinotto, D.V. Dimarogonas, P. Oegren, The advantages of using behavior trees in mult-robot systems, in: Proceedings of ISR 2016: 47st International Symposium on Robotics, VDE, 2016, pp. 1–8.
[42] M. Colledanchise, D. Almeida, P. Ogren, Towards blended reactive planning and acting using behavior trees, 2016, arXiv preprint arXiv:1611.00230.
[43] G.A. Korsah, A. Stentz, M.B. Dias, A comprehensive taxonomy for multi-robot task allocation, Int. J. Robot. Res. 32 (12) (2013) 1495–1512.