An Integrated Dynamic Method for Allocating Roles and Planning Tasks for Mixed Human-Robot Teams

Fabio Fusaro[1,2*], Edoardo Lamon[1*], Elena De Momi[2], and Arash Ajoudani[1]

Abstract— **This paper proposes a novel dynamic method based on Behavior Trees (BTs) that integrates planning and allocation of tasks in mixed human robot teams, suitable for manufacturing environments. The Behavior Tree formulation allows encoding a single job as a compound of different tasks with temporal and logic constraints. In this way, instead of formulating an offline centralized optimization problem, the role allocation problem is solved with multiple simplified online optimization sub-problems, without complex and cross-schedule task dependencies. These sub-problems are defined as Mixed-Integer Linear Programs (MILPs), that, according to the worker-actions related costs and the workers' availability, allocate the yet-to-execute tasks among the available workers. To characterize the behavior of the developed method, we opted to perform different simulation experiments, in which the results of the action-worker allocation and the computational complexity are evaluated. The obtained results, due to the nature of the algorithm and to the possibility of simulating the agents' behavior, illustrate adequately also how the algorithm performs in real experiments.**

I. INTRODUCTION

The increasing demand for flexible and highly reconfigurable production lines of small-medium size enterprises needs industrial manipulators to be able to quickly adapt to diversified manufacturing requirements. In this context, torque-controlled collaborative robots (cobots), are not only able to deal with complex tasks [1] and to execute safe plans in human-populated and partially unstructured environments [2], but also to offload workers from repetitive and hard tasks [3].

Moreover, cobots are expected to be able to perform a wide variety of tasks, both autonomously and in collaboration with human co-workers, that can supervise and complement robot performance with superior expertise and task understanding, setting up proper mixed human-robot teams. This scenario reveals two fundamental problems: how to systematically assign a role to each member of the team to achieve a shared goal, and how to adapt online the robot plan to dynamical changes of role between the team agents.

In the literature, a common approach to model the problem of allocating roles in a team of agents is through combinatory approaches. A formal analysis, comprehensive of computational models, is presented first by *Gerkey and Matarić* [4] and then updated by *Korsah et al.* [5]. Examples of such methods, applied to human-robot mixed teams are represented by [6], [7]. In order to enable the algorithm with dynamic behaviors, that can adapt the offline plan with online contingencies, researchers started to add also dynamic re-scheduler [8]. Another extensively used method in human-robot teams is represented by AND/OR graphs, for the ability to decompose assembly tasks [9], [10]. An online scheduler based on time Petri nets is developed in [11], where the robot adapts its schedule based on human activities. In this direction, researchers started to study also which features should be considered in such problems, adapting the robot plan to human co-worker preferences, capabilities, and ergonomics [12]–[16].

The main limitation of these approaches consists in the fact that the task allocation and planning problems are solved in two different phases, often using two completely different and separated methods. The proposed method, instead, exploits the strength of a centralized reactive and modular task planning method, in charge of scheduling the tasks executions, with the advantages of a cost-based role allocator, that,

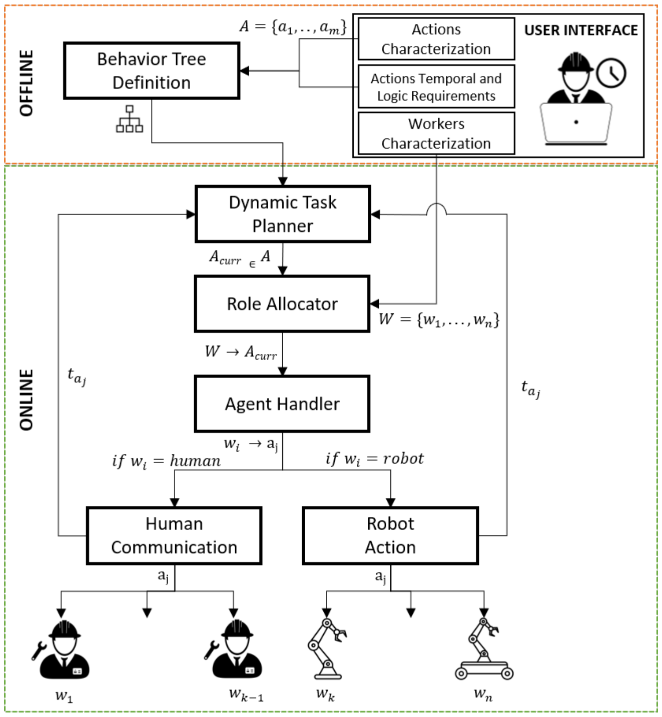

Fig. 1. Scheme of the allocation and planning method. First from the user interface, actions and workers information are defined offline. Next, the BT is constructed using the received characterizations. Then, in the online phase, the BT starts its execution and dynamically allocate the actions to the workers, exploiting the *Role Allocator*. Finally, through the *Agent Handler*, the BT either communicates the action to the human worker or makes the robot executing the task.

* Contributed equally to this work.

1- HRI[2] Lab, Istituto Italiano di Tecnologia, Via Morego 30, 16163, Genova, Italy. 2- Department of Electronics, Information and Bioengineering, Politecnico di Milano, Milano, Italy. fabio.fusaro@iit.it

This work was supported by the European Research Council's (ERC) starting grant Ergo-Lean (GA 850932).

before the proper execution phase, solves dynamically the problem of allocating a subset of actions to the agents. The role of the task planner is to dynamically schedule different tasks, ensuring that the task temporal and logic constraints, in terms of sequence or parallel of tasks, are satisfied. In this way, the planner reduces the centralized problem of allocating all the tasks to all the agents to different sub-optimal, but computationally less expensive, problems that allocate a subset of tasks to the agents, without any constraint generated by the plan.

In particular, the centralized task planner is defined by means of Behavior Trees (BTs), which triggers the action execution of all the agents of the team. Differently from the usual approach, where the behavior of each robotic agent is ruled by its own planner, here the BT models the task instead of the agent. The main advantage consists in the fact that, in this way, each task can be delegated to a different agent, according to the results of the *Role Allocation* node. In particular, thanks to the *Agent Handler* node, the tasks assigned to the robotic agents will be directly executed through the *Action* nodes, whereas the other ones will be delegated to the human workers by means of the *Human Communication* node. The architecture of the method is depicted in Figure 1. On the other hand, the role allocation problem allows to dynamically assign a suitable agent for each task. The suitability is evaluated through the action-worker related costs, which are used to describe how well an agent performs a task. These costs should model both general kino-dynamic agent features, tasks duration, availability, and also more specific ones, such as expertise and ergonomics. In particular, by taking into account tasks' duration and agent availability, it is also possible to dynamically schedule the tasks, to coordinate agents' effort. The method has been tested with simulation experiments in which both the proposed BT structure and the role allocation problem are evaluated. First, the computation complexity of the whole framework is computed increasing progressively the number of actions and workers. Then, three variations of the allocation algorithm, with different values of the agent-related availability cost, are compared.

To summarize, the contribution of this manuscript consists of a generic method to manage complex decomposable jobs with schedule constraints, able to optimally assign an agent and plan every single task of the job for a heterogeneous human-robot team. To enable the task planner to dynamically decide which agent is the most suitable to allocate the task, we created four new BT nodes to solve the sub-tasks role allocation and the consequent agents' handler and executor.

II. MODIFIED BTs WITH ROLE ALLOCATION

A. Preliminaries on Behavior Trees

A BT is a directed rooted tree, consisting of internal nodes for control flow and leaf nodes for action execution or condition evaluation. It is composed by *parent* and *child* nodes that are adjacent pairs. The main node, root, is the only one without parents and starts the execution of its children by propagating a signal, called *tick*, through the tree. Once the children are ticked, they immediately return a status to the parent: if the child is executing returns *RUNNING*, if the node completes the execution successfully returns *SUCCESS*, otherwise, if it fails, returns *FAILURE*. There are two main types of nodes: control and execution. The control ones are divided into four standard categories (Sequence, Fallback, Parallel, Decorator), while the execution ones in two (Action, Condition). The standard types of nodes, with their symbol and the return status depending on each case, are summarized in Table I. Standard BTs were designed to control an agent behavior, by reactively plan tasks to execute [17]. Thanks to their design, they can generate different behaviors that satisfy conditions evaluated online. Moreover, since they allow the execution of tasks both in parallel and in sequence, their adaptation to human-robot industrial tasks, such as cooperative assemblies, looks promising.

TABLE I
BT NODE TYPES AND RETURN STATUS.

Type of Node	**Symbol**	**Success**	**Failure**	**Running**
Sequence	$\rightarrow$	All children succeed	One child fails	One child returns Running
Fallback	?	One child succeeds	All children fail	One child returns Running
Decorator	$\diamond$	Custom	Custom	Custom
Parallel	$\rightrightarrows$	$\geq M$ children succeed	$> N - M$ children fail	else
Condition	⬭	True	False	Never
Action	▭	Upon completion	Impossible to complete	During execution

B. Nodes for Role Allocation

To integrate the role allocation problem in a BT, the standard usage of the method should have modified. First, it is important to specify that, in our scheme, the BT controls the job behavior, instead of the agent behavior, where a job is represented by the set of tasks composing it plus their temporal constraints. In this way, we exploit the BT to model all the possible execution of a job, and only when one or multiple tasks composing the job should be executed, the role allocation problem is solved and, finally, the different agents are informed of the results (see Figure 1). For this purpose, we defined four custom nodes and a particular subtree, named *Planning and Allocation Handler*, is designed.

Each developed node has to communicate with the others to share information and data. When the nodes are not directly connected or the return values are not sufficient to achieve desired behaviors, the BT exploits input/output ports. Each port is defined by a unique name and can be used as static to read elements that are input from the external in the creation phase of the BT or as dynamic to read and/or write data. The output ports write the elements into a shared *blackboard* associating to each variable a fixed name.

The general structure of the BT developed to plan and allocate tasks is shown in Figure 2. The tree shows the combination of tasks in series and parallel, that ends with a sequence. Before the execution layer, i.e. the action nodes, we designed a fixed subtree where the *Role Allocator* node, represented by the symbol $W \rightarrow A$, is in charge of solving

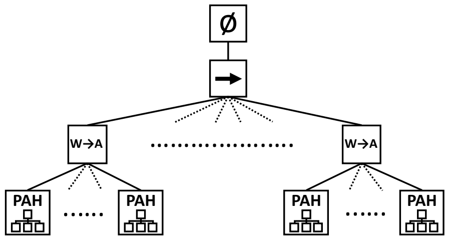

Fig. 2. Planning and Allocation Behavior Tree. The structure of the BT is composed by a sequence of *Role Allocator* nodes ($W \rightarrow A$). Each allocator node has the *Planning and Allocation Handler* subtree (PAH), shown in Figure 3, as children.

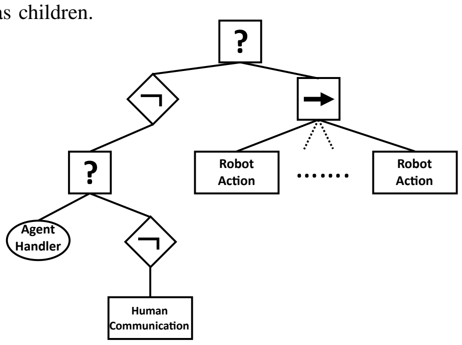

Fig. 3. Planning and Allocation Handler subtree. The developed structure, using two fallbacks (?) and two inverters (¬) allows to manage the return status of the *Agent Handler* node to either communicate the allocation to the human or make the robot executing the sequence of primitive actions.

the allocation of such actions to the agents. The subtree *Planning and Allocation Handler*, children of the Role Allocator displayed in Figure 3, is in charge of delegating the task to each agent. Each allocator node has a number of children equal to the number of tasks that can be executed in parallel.The custom nodes are explained in details in the following subsections.

1) Role Allocator Node: This node is the parent of the *Planning and Allocation Handler* subtree. It is defined as a control node and shares few similarities with the parallel node. The children of this node are the *Planning and Allocation Handler* subtrees, one for each task that can be executed in parallel. The node reads from the static input port the info of the children's actions that still have to be executed, and the agents' info. In this way, the node can compute all the agents-task related costs, generate the role allocation problem as explained in section III, and output the result. Then, each allocated task ticks the related child. The node is then executed again until all the tasks are completed. The pseudocode of the *Role Allocator* node is synthesized in Algorithm 1.

2) Agent Handler Node: This node is a custom condition node and it is in charge of selecting the agent, according to the allocation results. Specifically, the *Agent Handler* reads the results and returns different status in case the task has to be communicated to a human worker or it has to be accomplished by a robot. Thanks to the structure of the *Planning and Allocation Handler* subtree, if the node returns *FAILURE* the BT ticks the *Human Communication* node that is in charge to communicate the allocated action to the human. While, if the node returns *SUCCESS*, the BT

Algorithm 1 Tick() function of the "RoleAllocator" node.

```
1:  procedure RoleAllocator::tick()
2:      if $action\_to\_be\_allocated \neq 0$ then
3:          $[W,\ A] = Allocate()$
4:          $setOutput([W,\ A])$
5:      end if
6:      for $[w,\ a]$ in $[W,\ A]$ do
7:          $child\_idx \leftarrow a.ID$
8:      end for
9:      for $idx$ in $child\_idx$ do
10:         if $\neg\ idx$ in $executing\_child$ then
11:             $child\_status \leftarrow child(idx).Tick()$
12:         else
13:             $child\_status \leftarrow child(idx).Status()$
14:         end if
15:         if $child\_status == FAILURE$ then
16:             CLEAR($executed\_child$)
17:             return $FAILURE$
18:         else if $child\_status == SUCCESS$ then
19:             if $\neg\ idx$ in $executed\_child$ then
20:                 ADD($idx$ IN $executed\_child$)
21:             end if
22:             if $executed\_child.size() == child.size()$ then
23:                 return $SUCCESS$
24:             end if
25:         end if
26:     end for
27:     return $RUNNING$
28: end procedure
```

ticks the *Robot Action* nodes that make the robot executing the scheduled actions. The pseudocode of the *Agent Handler* node is summarized in Algorithm 2.

Algorithm 2 Tick() function of the "AgentHandler" node.

```
1:  procedure AgentHandler::tick()
2:      $getInput([W,\ A])$
3:      for $[w,\ a]$ in $[W,\ A]$ do
4:          if $w.type$ == HUMAN then
5:              return $FAILURE$
6:          else if $w.type$ == ROBOT then
7:              return $SUCCESS$
8:          end if
9:      end for
10: end procedure
```

3) Robot Action Nodes: The action nodes are in charge to trigger the activation of specific motions of the robots. The node communicates directly with the robot motion planner or with the controller, depending on the development of the node itself. In our case, we defined a finite set of actions, that represent motion primitives e.g. grasp, move, etc. The advantage of primitive actions as nodes is that we do not need to create an action for each robot or different specifications of the primitive itself. A motion primitive has an interface, where it is possible to define all the specific information needed to be executed, position in space, force to be exerted, etc., but it is implemented differently in each robot. Another advantage consists in the fact that this action info can be used not only in the execution phase but also by the role allocator node to compute the execution costs. When the agent starts the action execution, the node changes the worker availability and outputs it in the blackboard. Each agent has its own port to allow the execution of different actions in parallel avoiding more than one node changing the status of the same worker.

4) Human Communication Node: The *Human Communication* node is the corresponding node to the *Robot Action* for the human workers: it is in charge of communicating to each human agent the action is asked to execute by reading the allocation results. The communication means can differ in relation to the environment and/or the workers' equipment, e.g. displaying the information through a monitor or wearable devices such as smartwatches or mixed reality smartglasses [14].

III. ROLE ALLOCATION

A. Problem Statement

The multi-agent task allocation (MATA), considered as a more general class of the well-known multi-robot task allocation (MRTA), is the problem of determining which agent, either human or robotic, is in charge of executing every single task that is needed to achieve the team's goal.

Before formalizing the role allocation problem, we need to specify which are the main components of the problem. Following the symbolism introduced in [14], we consider a mixed human-robot team of workers, or agents, $W = \{w_1, ..., w_n\}$, $|W| = N$. The goal is to complete a general single job A that could be further decomposed into the sequence and parallel of tasks, or actions, $A = \{a_1, .., a_m\}$, $|A| = M$. The set of L actions that can be executed by each agent $w_i \in W$ are $A_i = \{a_{i1}, .., a_{il}\}$, $|A_i| = L$, where $A_i \subseteq A$. We want to obtain is the allocation of an agent w_i to each of the actions he is able to execute a_{ij}, denoted $w_i \rightarrow a_{ij}$. The set of worker-action allocations is denoted $W \rightarrow A$.

B. Mathematical Model

In this work, the MATA problem is formalized as a Mixed-Integer Linear Program. The most general scenario of MATA problems in human-robot collaboration, according to an adapted version of the taxonomy presented by *Gerkey and Matarić* [4], is characterized by multi-task agents, i.e agents that can execute multiple tasks simultaneously, multi-agent tasks, i.e, tasks which requires multiple agents, and, finally, time-extended assignments, i.e. the allocation considers also future allocations. To the best of the authors' knowledge, a mathematical model that captures the features of such a complex problem is still missing in literature [5]. However, in our framework, thanks to the decomposition of the task that BTs are able to achieve, the problem is extremely simplified. First, each agent, by definition, can perform only a task at once; second, each task requires a single agent. Moreover, collaborative tasks, that require more than a single agent at the same time, are already decomposed by the BT into parallel tasks. By doing so, we have removed all the *complex* and *cross-schedule* dependencies, i.e. the effective cost of an agent for a task and the allocation constraints do not depend on the schedules of other agents. In practice, the BT *Allocator* node deals only with the allocation of a subset of tasks, that are only the tasks that, according to the job schedule represented by the BT, should be allocated within the available agents, having as constraints, in the worst-case scenario, *intra-schedule* dependencies, i.e. the agent cost for an action depends on the other actions the agent is performing. Hence, the problem of allocating L actions, where $L \leq M$, to N workers, can be formalized in the following way.

Minimize:

$$\sum_{w_i \in W} \sum_{a_j \in A} (c_{ij} + \chi_i) x_{ij} \tag{1}$$

Subject to:

$$\begin{aligned}
x_{ij} &\in \{0,1\} && \forall w_i \in W, \forall a_j \in A \\
x_{ij} &= 0 && \text{if } a_j \notin A_i \\
\sum_{w_i \in W} x_{ij} &\leq 1 && \forall a_j \in A \\
\sum_{a_j \in A} x_{ij} &\leq 1 && \forall w_i \in W \\
\sum_{w_i \in W} \sum_{a_j \in A} x_{ij} &= \min(L, N) && \\
\sum_{w_i \in W} \sum_{a_j \in A} t_{ij} x_{ij} &\leq T_k && \forall k \in K
\end{aligned} \tag{2}$$

where c_{ij} represents the cost related to an agent w_i in executing the task a_j, χ_i is the availability cost function, and the x_{ij} represents the $M \times N$ binary optimisation variables of the problem, where $x_{ij} = 1$ means that the worker i is assigned to action j ($w_i \rightarrow a_j$).

1) Constraints Design: The problem constraints are exploited to ensure the feasibility of the problem:

- the first is representative of the binary nature of the variables;
- the second one ensures that the solver does not allocates a worker to an action that is not capable to execute;
- the third and fourth constraints ensures that to each agent only one task is allocated, and the same task is not allocated to two different agents.
- the fifth constraint ensures that there are exactly a number of allocations equal to the number of agents N, in case $N > L$, where L are the tasks to be allocated, and L otherwise;
- the budget constraint ensures limits to the number of tasks assigned to each agent, where t_{ij} is the budget that w_i would spend for a_i, and T_k is the budget limit for the K joint agent-task constraints.

2) Costs Design: In this work, the optimization costs are split into two main components: the agent-actions costs c_{ij} and the agent-related availability cost χ_i. The first should describe how capable is a worker in performing each task, considering the kino-dynamics features of the agents, the tasks duration, the human ergonomics, etc. The role and the design of the costs for such problems has already been described in our previous work [14] and, hence, it will not be repeated here.

The availability value has been added to the cost function, and not to the problem constraints, since some since we cannot ensure that at least an agent is always available. In case multiple agents are available, the allocator node tries to assign the task to the agent whose cost is smaller. We consider an agent available if it is present in the workcell and if it is not occupied by any other task. One simple definition of the availability activation is the following:

$$\chi_i = \begin{cases} 0, & \text{if } w_i \text{ is } \textit{available}; \\ \alpha_i, & \textit{otherwise.} \end{cases} \tag{3}$$

α_i is the availability cost of w_i. To ensure that the availability weights more than the other costs, we set $\alpha_i > \max\{c_{ij}\}$. With this method, we are able to ensure that the allocation node favors an available agent, minimizing in this way the single-agent waiting times. In this case, the agent's suitability for the task is not considered. On the other hand, in some situations, instead of allocating a task only among the available agents, which might all be strongly unsuitable for the task, it might be convenient to make the system wait for the most suitable agent. For this reason, we modified the binary nature of the availability to account for the remaining execution time:

$$\chi_i = \begin{cases} 0, & \text{if } w_i \text{ is } \textit{available}; \\ \alpha_i \frac{T_{a_{ij}} - t_{a_{ij}}}{T_{a_{ij}}}, & \textit{otherwise.} \end{cases} \tag{4}$$

where $T_{a_{ij}}$ is the nominal duration of a_j performed by w_i, $t_{a_{ij}}$ is the time spent by w_i for a_j from the beginning of a_j, where $t_{a_{ij}} \leq T_{a_{ij}}$. In this way, $\frac{T_{a_{ij}} - t_{a_{ij}}}{T_{a_{ij}}} = 1$ if the task has just started, and, as the agent finishes the task, it goes to 0. To ensure that the two costs are comparable, $\alpha_i = \max\{c_{ij}\}$.

IV. EXPERIMENTS

The performances of the proposed approach are evaluated through simulation experiments. In order to analyze the performances of the method, different experiments are conducted varying the job characteristics, such as the number of actions in sequence, in parallel, and the number of workers. The simulation experiments were run on a laptop with an Intel Core i7-8565U 1.8 GHz × 8-cores CPU and 8 GB RAM. The architecture has been developed in C++, on Ubuntu 18.04 and ROS Melodic, exploiting the BehaviorTree.CPP (`https://github.com/BehaviorTree/BehaviorTree.CPP`) library to define the BT nodes and the Osi (`https://github.com/coin-or/Osi`) library with GLPK (GNU Linear Programming Kit) solver to formalize the MATA problem.

First, the computational complexity of the whole proposed method is evaluated. The computation time has been counted from the initialization of the BT until the action execution, excluded the execution time of the action, as the mean of 10 repetitions of the same BT. To generate different situations, the number of tasks in sequence, in parallel, and the number of agents is progressively increased. The tasks in parallel are

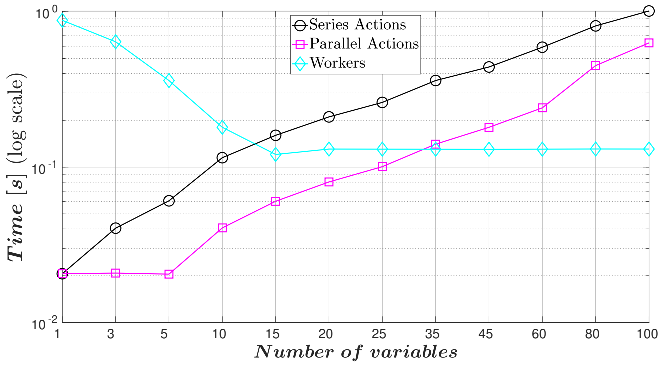

Fig. 4. Computational time (in log scale) of series (black) and parallel (pink) actions and workers (cyan), with different number of variables.

all children of the same allocation node, hence increasing parallel tasks means increasing the size of the MILP. On the other hand, the tasks in sequence are not in parallel with any task, and, hence, increasing the number of tasks in sequence means increasing the number of MILPs to solve. While the number of tasks increases, the number of workers is fixed to 4. Both actions in parallel and in series are increased from 1 to 100. The results in Figure 4 show that the trend of the computation time, increasing the number of actions both series and parallel, can be approximated with a linear function. It is interesting to notice that, even if the trends are similar, in this specific scenario, it is faster to solve a N-sized MILP than N MILPs. This, however, does not imply that solving the offline centralized problem is faster than the decomposed sub-problems since the centralized one should include all the task temporal constraints. A different simulation is conducted to estimate the evolution of the computation time increasing the number of workers, keeping the number of actions fixed. In this case, the BT is constructed to have 11 *Role Allocator* nodes in sequence, each one with: 12, 8, 10, 12, 8, 10, 12, 8, 10, 5, 5 number of parallel actions, respectively. Again, the number of workers is increased starting from 1 until 100, and, as can be noticed in Figure 4 in cyan, increasing the number of agents the computation time decreases. This is because the number of actions executed in parallel increases with the workers until the maximum degree of parallelism is reached. Specifically, this happens when the agents are 15 and, hence, are more than the maximum number of actions in parallel, i.e. 12. From that value on, the computation time is approx. constant.

Then, the allocation approach is validated with a single fixed job. The goal is to run the allocation method for the same job with different costs, to validate the different design of the availability cost χ_i explained in subsubsection III-B.2. The job is made of a total of 14 actions, combined in a sequence of 4 different sets of actions that can be executed in parallel. For this reason, 4 *Role Allocator* nodes are present in the BT. Each node has 3, 4, 5, and 2 actions as children, respectively. The number of workers is fixed to 4. The agent-actions related costs c_{ij}, for simplicity, are defined as the time ($t_{a_{ij}}$) required by the worker w_i to execute the action a_j. First, we compute the allocation with availability cost defined in Equation 4. The results in Table II show

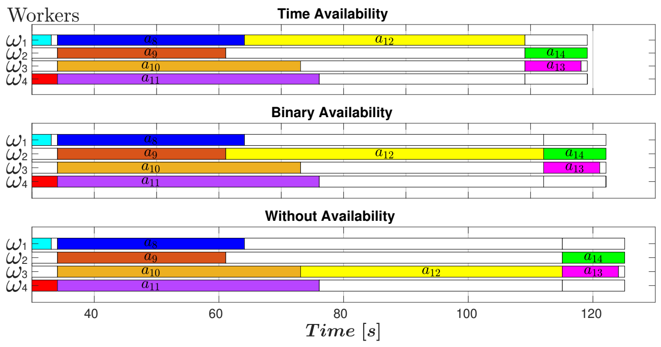

Fig. 5. Detail of the Gantt charts related to the allocation of actions a_8–a_{12}: in the top plot the availability cost χ_i is computed as in Equation 4, in the middle one as in Equation 3 and in the bottom one $\chi_i = 0$.

that, in general, the algorithm picks always the agent that minimizes the total execution time. These results are then compared with the same allocation method computed with the binary availability Equation 3 and without any availability cost ($\chi_i = 0$). The results differ for the allocation of action a_{12} (see Figure 5). Without considering the availability of the agents, the action a_{12} is assigned to the worker w_3 minimizing only the agent-actions related costs c_{ij}. This, however, is not optimal since w_3 is occupied by another task. In the case of the binary availability, the action a_{12} is assigned to the worker w_2, since he's the first available after finishing a_{10}. But, even if the w_2 starts before, the execution time required by him to achieve a_{12} is higher than the expected waiting time for the execution of a_8 by w_1 plus the following a_{12}, that is the solution proposed with availability cost in Equation 4. Consequently, this solution minimizes also the overall duration of that set of tasks, and also the waiting time for the other agents for the next task. It can be noticed that actions a_8, a_9, a_{10} and a_{11} starts at the same time because are defined as parallel actions but in series with the previous ones, hence, these can start only when all the previous ones are completed.

V. CONCLUSIONS

In this work, we proposed a novel integrated method to allocate and plan tasks for mixed human-robot teams. The method extends the standard formulation of BTs with custom reusable nodes that enable to dynamically generate and solve different MILPs. The results showed the crucial role of the cost definition in the allocation behavior. For these reasons, further metrics should be evaluated with the method, to reduce the agent workload by optimizing the human ergonomics. Furthermore, future studies will compare the method with other state-of-the-art approaches, focusing not only on the computational complexity but also on the intuitiveness and user-friendliness assessment of the interface to generate the job. Finally, the effectiveness of the allocation method should be evaluated with real experiments in multi-human and multi-robot teams.

TABLE II

ALLOCATION RESULTS OF THE ACTIONS OF THE SIMULATED JOB.

Action	$t_{w_1}(s)$	$t_{w_2}(s)$	$t_{w_3}(s)$	$t_{w_4}(s)$	Worker allocated
a_1	20	13	15	15	w_2
a_2	17	20	22	16	w_4
a_3	10	12	17	11	w_1
a_4	13	15	9	21	w_3
a_5	22	18	24	18	w_4
a_6	11	9	15	15	w_2
a_7	17	23	18	16	w_1
a_8	30	54	48	57	w_1
a_9	66	27	60	39	w_2
a_{10}	60	66	39	75	w_3
a_{11}	63	48	57	42	w_4
a_{12}	45	51	42	54	w_1
a_{13}	14	17	9	16	w_3
a_{14}	21	10	15	18	w_2

REFERENCES

[1] A. Ajoudani, A. M. Zanchettin, S. Ivaldi, A. Albu-Schäffer, K. Kosuge, and O. Khatib, "Progress and prospects of the human–robot collaboration," *Autonomous Robots*, vol. 42, pp. 957–975, 2018.

[2] S. Haddadin, A. Albu-Schaffer, A. De Luca, and G. Hirzinger, "Collision detection and reaction: A contribution to safe physical human-robot interaction," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 3356–3363.

[3] W. Kim, J. Lee, L. Peternel, N. Tsagarakis, and A. Ajoudani, "Anticipatory robot assistance for the prevention of human static joint overloading in human–robot collaboration," *IEEE Robotics and Automation Letters*, 2018.

[4] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *The International Journal of Robotics Research*, vol. 23, no. 9, pp. 939–954, 2004.

[5] G. A. Korsah, A. Stentz, and M. B. Dias, "A comprehensive taxonomy for multi-robot task allocation," *The International Journal of Robotics Research*, vol. 32, no. 12, pp. 1495–1512, 2013.

[6] C. Ferreira, G. Figueira, and P. Amorim, "Scheduling human-robot teams in collaborative working cells," *International Journal of Production Economics*, vol. 235, p. 108094, 2021.

[7] M. C. Gombolay, C. Huang, and J. Shah, "Coordination of human-robot teaming with human task preferences," in *2015 AAAI Fall Symposium Series*, 2015.

[8] A. Pupa, W. Van Dijk, and C. Secchi, "A human-centered dynamic scheduling architecture for collaborative application," *IEEE Robotics and Automation Letters*, pp. 1–1, 2021.

[9] L. Johannsmeier and S. Haddadin, "A hierarchical human-robot interaction-planning framework for task allocation in collaborative industrial assembly processes," *IEEE Robotics and Automation Letters*, vol. 2, no. 1, pp. 41–48, Jan 2017.

[10] K. Darvish, E. Simetti, F. Mastrogiovanni, and G. Casalino, "A hierarchical architecture for human–robot cooperation processes," *IEEE Transactions on Robotics*, vol. 37, no. 2, pp. 567–586, 2021.

[11] A. Casalino, A. M. Zanchettin, L. Piroddi, and P. Rocco, "Optimal scheduling of human-robot collaborative assembly operations with time petri nets," *IEEE Transactions on Automation Science and Engineering*, 2019.

[12] M. Gombolay, A. Bair, C. Huang, and J. Shah, "Computational design of mixed-initiative human–robot teaming that considers human factors: situational awareness, workload, and workflow preferences," *The International journal of robotics research*, vol. 36, no. 5-7, pp. 597–617, 2017.

[13] G. Michalos, J. Spiliotopoulos, S. Makris, and G. Chryssolouris, "A method for planning human robot shared tasks," *CIRP journal of manufacturing science and technology*, vol. 22, pp. 76–90, 2018.

[14] E. Lamon, A. De Franco, L. Peternel, and A. Ajoudani, "A capability-aware role allocation approach to industrial assembly tasks," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3378–3385, 2019.

[15] I. El Makrini, K. Merckaert, J. De Winter, D. Lefeber, and B. Vanderborght, "Task allocation for improved ergonomics in human-robot collaborative assembly," *Interaction Studies*, vol. 20, no. 1, pp. 102–133, 2019.

[16] F. Fusaro, E. Lamon, E. De Momi, and A. Ajoudani, "A human-aware method to plan complex cooperative and autonomous tasks using behavior trees," in *2020 IEEE-RAS International Conference on Humanoid Robots (Humanoids 2020)*, 2021, p. forthcoming.

[17] M. Colledanchise and P. Ögren, *Behavior trees in robotics and AI: An introduction*. CRC Press, 2018.