Online Scheduling and Reactive Behaviors for Effective Human-Robot-Collaboration

Marina Ionova, Petr Vanc, and Jan Kristof Behrens

***Abstract*—Assembly processes involving humans and robots are challenging scenarios because the individual activities and access to shared workspace have to be coordinated. Fixed robot programs leave no room to diverge from a fixed protocol. Working on such a process can be stressful for the user and lead to ineffective behavior or failure. We propose a novel approach of utilizing uncertainty-aware constraint-based scheduling in a reactive execution control framework facilitating behavior trees. This allows the robot to adapt to uncertain events such as delayed activity completions and activity selection (by the human). The user can rest assured that the robot will wait for him to finish his activity and will adapt to best complement the human-selected activities to complete the common task. In addition to the improved working conditions, our algorithm also leads to increased efficiency, even in scenarios with high uncertainty. We evaluate our algorithm using a probabilistic simulation study and initial real robot experiments using a Franka Emika Panda robot and human tracking based on HTC Vive VR gloves.**

I. Introduction

Human-robot collaboration is a very challenging task for many reasons. One is the uncertainty introduced by the human collaborator and amplified by complex task dependencies. These include uncertainties regarding the task durations, the probability of task completion, and perception uncertainties. However, humans and robots working on a shared task promise to reap the best of two worlds: the robot's diligence and the human's dexterity and intelligence. This paper proposes a reactive architecture that utilizes Constraint Programming-based scheduling and Behavior Trees (BT) to tame the inherent uncertainty. We focus on the uncertainty in the timing of task execution and task rejection by humans. We assume that agents always finish their tasks, which might take longer than expected. We evaluate our approach in an extensive probabilistic simulation study. We implemented the method also using a real Franka Emika Panda robot, a vision system with marker-based task state tracking, and HTC Vive-based human activity tracking.

We compare our method to the method proposed by Petzoldt et al. [5], which uses a greedy approach and assigns tasks to the agents based on the current task availability without using scheduling. In contrast, our method considers a whole schedule for the rest of the task. Another recent work, the PLATINUM system, utilizes time-line based scheduling [7]. While PLATINUM implements its flaw-resolver-based temporal planning with uncertainty, we utilize similar data structures in our model. Still, we reduce the problem to Constraint Satisfaction Problems and use an off-the-shelf solver that produces solutions fast enough to run online. We incorporate a measure for uncertainty into the costs to incentivize the solver to favor more stable schedules. Most often, the proposed scheduling model proves even the optimality of the returned solution.

The remainder of the paper is structured as follows. section II introduces preliminaries about Constraint Programming and Behavior trees that are required to understand the rest of the paper. In section III, we discuss the class of use cases, the scheduling model, and the online scheduling architecture. section IV provides a short overview about the implementation. section V describes the conducted experiments and section VI discusses the achieved results. Finally, we conclude the paper in section VII.

II. Preliminaries and Background

In this Section, we introduce concepts about constraint programming-based scheduling and reactive behavior generation using behavior trees that are required to understand the content of this paper.

A. Constraint Programming for Scheduling

In Constraint Programming (CP), we model a (planning) problem declaratively in terms of Constraint Optimization Problems (CSP). A Constraint Satisfaction Problem is generally specified by a triple $\mathcal{P} = (X, D, C)$, where X is a n-tuple of variables $X = \{x_1, x_2, \ldots, x_n\}$, D is a n-tuple of domains $D = \{D_1, D_2, \ldots, D_n\}$, and C is a t-tuple of constraints $C = \{C_1, C_2, \ldots, C_t\}$. The domain D_i maps the variable x_i to possible values of x_i: $D(x_i) = D_i$, i.e., $x_i \in D_i$. A constraint C_j is a tuple $\{R_{S_j}, S_j\}$, where S_j is the subset of variables in X, which are involved in the constraint C_j. R_{S_j} is a relation between the variables S_j, which effectively defines a subset of the Cartesian product of the domains of the variables in S_j [2]. A solution of a CSP is a complete assignment $A = \{a_1, \ldots, a_n\}$, which assigns to each variable $x_i \in X$ a value a_i, which is within the domain $D(x_i)$ of this variable:

$$x_i \mapsto a_i, \forall i \in \{1, \ldots, n\}, \text{where } a_i \in D(x_i).$$

The task of a constraint solver is to find such a set A, such that A satisfies all constraints C. For minimization, the solver shall find the $A \in \mathcal{A}$ that minimizes or maximizes the value of a designated variable. $\mathcal{A}$ is the space of all solutions.

B. Behavior Trees

Behavior trees (BT) are a behavior modeling tool that is inherently composable and reactive. BTs were originally developed for the game industry to control Non-Player-Characters,

but recently received much attention in robotics deliberation [1, 3]. They could be seen as the link between the world of AI planning that operates on a closed world and robotic online deliberation that needs to deal with uncertainty and external perturbations of the environment [4].

BTs are trees composed of behavior nodes. Leaf nodes are actual behaviors or condition checks. Intermediate nodes are processors or modifiers. The central operation in a BT is the *tick*. The root node is regularly *ticked*. Each node recursively ticks its child nodes and returns a status of either *running*, *success*, or *failure*. The processors decide how to handle the states of their children. Typical processor nodes are *sequence*, *parallel*, *selector*, but many variants exist.

BTs can be, due to their clear structure, manipulated programmatically. This also allows us to refine them on the go. We could add a node for an abstract action into the tree and expand it when the node is ticked (for example, by planning). In this paper, we encode a base robotic behavior as BT and dynamically expand and prune the BT based on the current situation and the scheduling results.

III. Online Scheduling Architecture

In this section, we present the details about the use case formalization, the CP formulation for scheduling, and the integration into an online deliberation for HRC.

A. Use Case Specification

The setup consists of a table-top with a Franka Emika Panda manipulator. A human worker can approach the setup from the right side. A typical use case is shown in Figure 1. We use 3D printed assembly tasks as proposed by Riedelbauch and Hümmer [6]. The layout features an assembly area (the black construction plate), a part storage for the robot (left), a shared part storage (next to the robot base), and a part storage for the worker (right). The worker and the robot cannot access the assembly area and the shared part storage simultaneously. The human and the robot must collaborate to complete the assembly task (see the finished task state in Figure 2). Specifically, they have to coordinate their access to the shared areas and carry out the subtasks in an order consistent with the task's dependency graph shown in Figure 2 on the right.

Note that the subtask allocation, ordering, and timing can be used to optimize the collaboration. These decisions significantly influence human-robot coordination, the robustness of the schedule, and finally, the makespan.

B. Scheduling Model

The formulation of the scheduling model builds upon the notion of subtasks t with time points that form intervals (picking and placing one block is represented as one subtask). The constraint variables t_s and t_e designate the start and the end of the task t, respectively. The domains for all time point variables is $\{0 \dots H\}$, where H is the considered time horizon. Each subtask must be allocated to exactly one suitable actor (i.e., the robot or the human worker—whoever can reach the pickup and placing locations) that is designated by the variable $t_a \in A$, where A is the set of all actors. T is the set of all subtasks.

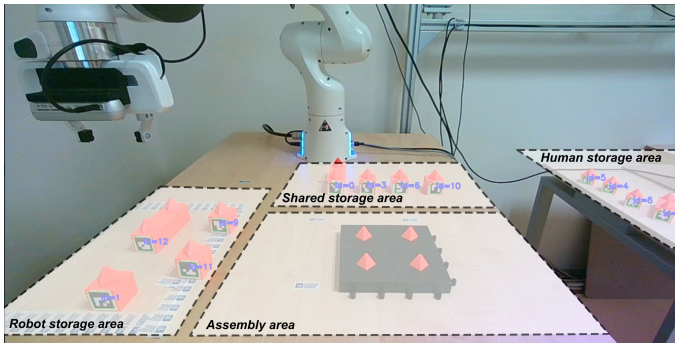

Fig. 1. The real robot setup and an example task based on the HRC task generator by Riedelbauch and Hümmer [6].

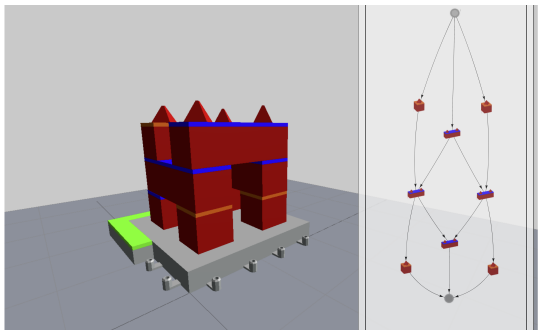

Fig. 2. The final state of the shared assembly task. On the right, the precedence graph is shown.

Each actor can only work on a single task at a time. Hence, subtasks allocated to an actor may not overlap.

$$\forall t^i, t^j \in T | i \neq j : t^i_a = t^j_a \implies t^i_s > t^j_e \veebar t^j_s > t^i_e.$$

We divide subtasks into three phases called *preparation*, *execution*, and *completion*. In the *preparation* phase, the actor picks up objects and moves toward the workspace. The *execution* phase is dedicated to work in the (possibly) shared workspace. All execution phases with the same shared target area must be non-overlapping. The *finishing* phase describes the time from leaving the shared workspace until new work can be started (for example, tidying up tools). We denote the time points separating the phases t_1 and t_2, respectively. Similarly, subtasks that require access to the same shared area may not overlap when allocated to different agents.

$$\forall t^i, t^j \in T | i \neq j : t^i_r \setminus t^j_r \neq \varnothing \implies t^i_1 > t^j_2 \veebar t^j_1 > t^i_2,$$

where t_r designates the resources required. We add a similar constraint for subtask preparation phases involving the shared part storage.

The subtask dependency graph $\mathcal{G} = (T, E)$, where T is the set of all subtasks and E is the set of all edges representing precedence constraints. A valid dependency graph $\mathcal{G}$ must be

directed and acyclic. If an edge $e \in E$ connects a vertex x^i with vertex t^j, then the completion of the execution interval of t^i is a condition for the execution interval of t^j, i.e., $t^i_2 < t^j_1$.

The optimization criterion to minimize is the makespan

$$c = \min_{A \in \mathcal{A}} \max_{t \in T} t_e;$$

i.e., minimize the latest end of all tasks. $\mathcal{A}$ is the set of all valid variable-value assignments in the CSP and T is the set of all task intervals and t_e is the end time point of the task t.

Task timing uncertainty can arise from human factors while sharing the workspace with a robot or other factors such as worn-out tools. The above model takes into account only this type of uncertainty. Moreover, we propose incorporating the uncertainty model for human task acceptance into the optimization. Human task rejection can disrupt the sequence of tasks, which leads to reduced schedule quality. This also impacts the reliability of predicted completion times.

Incorporating soft constraints into the scheduling process can be useful when data about a person's preferences (e.g., how often a certain task type was accepted or rejected) are available. We incentivize the solver to prefer assigning subtasks to the human that likely are accepted by introducing soft constraints, i.e., an additional component to the cost. Assigning a task to an agent is penalized by a value proportional to the probability that the assigned task is rejected.

The new objective function takes the form of minimizing the maximum completion time for all tasks plus the sum of all rejection probabilities:

$$\min_{A \in \mathcal{A}} (\max_{t \in T} t_e + \sum_{t \in T} p(t; t_a)(t_e - t_s)); \quad (1)$$

The function $p(\,)$ is the probability of the task t being rejected by actor t_a. The penalty is scaled based on the task duration $t_e - t_s$, as the rejection of a long task will have a bigger impact on the schedule. α scales the importance of the rejection risk penalty compared to the makespan.

C. Integrated Planning and Acting System

The online execution requires the management of the robot and the human agents and continuous supervision and incorporation of new observations about the progress into the scheduling. The robot is controlled by the BT shown in Figure 3, which ensures that the robot evades the human worker with high priority, works on the given subtasks with middle priority, or returns to its home pose. The human worker is informed about the task state using the GUI shown in Figure 4. Specifically, a dynamic Gantt chart (top-left) shows the current state and schedule of the task, and the dependency graph (bottom-left) shows the task-specific dependencies. On the right side, the current task details are displayed for the robot and the user. Human feedback about task completions or task rejections is collected via buttons. The scheduling model is always run when no schedule was computed or new conflicting evidence makes the last schedule obsolete (e.g., actual time points or task rejections). Observed facts are constrained to their actual values because the solver should not reason about the past.

Fig. 3. Behavior tree for managing the robotic agent. Successively, all tasks are written to the blackboard when they are due according to the schedule. First-added tasks have priority. In this way, the robot will repair the situation if accidentally progress is undone.

IV. Implementation

Our implementation is based on ROS 2. Our BTs utilize the py trees library. Object detection and 6D localization use Intel Realsense cameras and ducial markers on the object. The human pose is tracked using HTC Vive trackers. The robot is controlled using position-based servoing. The scheduling is implemented using Google OR-Tools.

V. Experimental Setup

We designed a probabilistic simulation to evaluate the effectiveness of the scheduling method under uncertainties in task rejection by humans and task duration. The task execution time is sampled from a bimodal probability distribution, where one mode represents normal execution, and the other represents failed attempts, which leads to increased task duration. We used the simulation to validate our implementation (e.g., that assignments were allocated and sequenced correctly). Furthermore, the rescheduling was tested to be consistent with all observed events. The simulation was designed to emulate the signals the agents would receive about the start of the tasks. Uncontrollable events, such as the completion or the rejection of a task are under the control of the simulation. Specifically, the scheduling algorithm was assigning tasks to robots (and simulated humans). Then simulating the duration for each task. After the simulated task completion, the feedback to the algorithm about the completion of the task, including information about the time for each phase is sent.

In addition to the temporal uncertainty we also simulate the human choice. When there is a decision to be made about accepting a task or a preference, the simulation randomly samples from a probabilistic distribution for that task. For