Adaptive Human-Robot Collaborative Missions using Hybrid Task Planning

Gricel Vázquez, Alexandros Evangelidis, Sepeedeh Shahbeigi, Simos Gerasimou
University of York, UK
{gricel.vazquez,alexandros.evangelidis,sepeedeh.shahbeigi,simos.gerasimou}@york.ac.uk

***Abstract*—Producing robust task plans in human-robot collaborative missions is a critical activity in order to increase the likelihood of these missions completing successfully. Despite the broad research body in the area, which considers different classes of constraints and uncertainties, its applicability is confined to relatively simple problems that can be comfortably addressed by the underpinning mathematically-based or heuristic-driven solver engines. In this paper, we introduce a hybrid approach that effectively solves the task planning problem by decomposing it into two intertwined parts, starting with the identification of a feasible plan and followed by its uncertainty augmentation and verification yielding a set of Pareto optimal plans. To enhance its robustness, adaptation tactics are devised for the evolving system requirements and agents' capabilities. We demonstrate our approach through an industrial case study involving workers and robots undertaking activities within a vineyard, showcasing the benefits of our hybrid approach both in the generation of feasible solutions and scalability compared to native planners.**

***Index Terms*—hybrid task planning, multi-robot multi-human systems, genetic algorithms**

I. Introduction

Cyber-Physical-Human Systems (CPHS) are advanced, interconnected systems comprising human agents, robotic components, and cyber infrastructure [1]. These systems are increasingly used in diverse domains such as agriculture [2], manufacturing [3], and healthcare [4], where they must collaborate effectively to complete their tasks while adhering to functional and non-functional, probabilistic and non-probabilistic requirements. A crucial CPHS challenge is developing robust task plans that account for uncertainty and ensure the successful completion of missions under dynamically changing conditions, while ensuring plan correctness [5], [6], [7], [8].

Uncertainty in CPHS arises from various factors, including unpredictable human behaviour, environmental variability, and mechanical failures [9]. Addressing such uncertainty often requires the verification of task plans by considering probabilistic properties, such as the likelihood of mission success. Probabilistic Model Checking (PMC) is a widely recognized approach for verifying these properties [10], [11]. However, while PMC excels at providing formal guarantees, its computational requirements grow exponentially with the size and complexity of the problem, a limitation commonly referred to as the state explosion problem [12]. This makes PMC impractical for large-scale CPHS task planning scenarios involving numerous agents, tasks, and potential disruptions [13], [14].

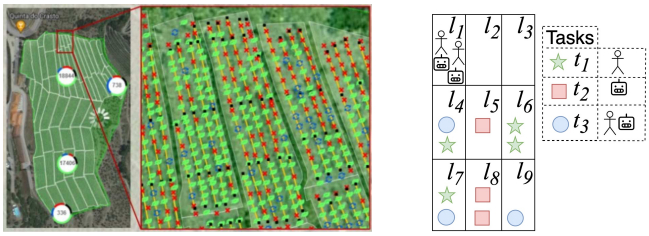
Existing approaches to task planning in CPHS can be broadly classified into deterministic and probabilistic methods [15]. Deterministic planners, such as those based on classical planning models like PDDL [16], are computationally efficient but fail to address uncertainties arising from dynamic environments or probabilistic task outcomes [17]. Probabilistic planning frameworks, such as those using Markov Decision Processes (MDPs), provide a formal way to model uncertainty but suffer from scalability issues due to the exponential growth of state space, as seen in large-scale multi-agent systems [13]. Probabilistic model checking tools like PRISM [18] and Storm [19] have been extensively used for verification, yet their application is often limited to relatively small problem instances due to computational constraints.

Recent advanced hybrid approaches combine the strengths of numerical and probabilistic methods. For instance, approaches like [20], [21] integrate constraint solving with probabilistic verification, while probabilistic extensions of the main definition language (PPDDL) [16] facilitate modelling and verification of uncertain planning domains. Despite these efforts, there remains a need for solutions that can generate adaptable, verified plans capable of addressing real-world requirements, such as task retries, energy consumption, and human factors, while accommodating runtime changes.

In this paper, we propose a hybrid task planning approach that decomposes the planning problem into two stages to tackle the scalability and efficiency challenges. First, we employ off-the-shelf numerical planning techniques to generate a feasible initial plan. Next, we augment this plan with uncertainty information yielding a parametric probabilistic model. Meta-heuristic search allows us to synthesise Pareto-optimal task plans that meet probabilistic requirements. We also introduce a task planning adaptation algorithm for the generation and unfolding of new correct and verified plans, if required.

We evaluate our approach in an industrial case study involving task execution at a vineyard, where human workers and robots collaborate to perform vineyard activities. The results demonstrate that our hybrid framework effectively balances the scalability of numerical planning with the rigour of probabilistic verification. By supporting incremental adaptation to runtime changes, such as task failures or evolving requirements, our approach ensures both robustness and flexibility in the CPHS plan, while also reducing computational demands.

Our main contributions are as follows: (1) we present a hybrid approach for the generation of correct verified plans leveraging numerical planning and PMC to mitigate the state explosion problem; (2) we integrate meta-heuristic search to



(a) Vineyard layout. (b) World abstraction

Fig. 1: Vineyard layout and abstract representation. Locations l_1-l_9 represent a section of a vineyard's row, where tasks $t1$-$t3$ have to be completed. Task $t1$ can be done by human workers, $t2$ by robots and $t3$ by either.

synthesise Pareto-optimal plans concerning maximising the probability of mission success while minimising the mission cost; (3) we propose an algorithm for task plan adaptation in response to failures in the completion of tasks and changes in requirements; and (4) we evaluate our hybrid planning approach and the adaptation algorithm on an industrial vineyard case study provided by our project partners.

Paper structure. Section II describes our industrial case study. Section III provides the required background. Section IV describes the problem formulation, while Sections V and VI present our approach and its evaluation, respectively. Finally, Sections VII and VIII cover related work and conclusions.

II. Vineyard Inspection CPHS Mission

We motivate our approach through an industrial case study on task automation in a vineyard in Portugal, provided by our research partner Quinta Do Castro and the University of Trás-os-Montes (see Figure 1a). The vineyard is divided into subsections, each with multiple rows where different varieties of vines grow. The old vineyards are equipped with GPS technology, providing the precise location of each plant.

The vineyard tasks are classified as follows:

- Harvesting ($t1$): Grape harvesting is an extremely arduous and repetitive activity. As a highly-dexterous task, this is delegated only to humans. Severe fatigue has to be mitigated whenever possible as, after many days in steep terrain conditions, workers show signs of physical and mental fatigue [22].
- Vineyard monitoring ($t2$): Continuous monitoring of vineyards includes monitoring the vines' health, the vines' state after a heatwave or heavy rainfall, and routine monitoring. As remote areas can be extremely difficult for humans to reach and magnify human fatigue, this task is delegated to robots.
- Grapevine identification ($t3$): Grapevine varietal identification occurs during the observation period between the hot months of June and August. This task, comprising vine leaf pictures taken and sent to a central unit, is shared between human workers and robots. This task might fail due to communication errors, hardware failures, or the collection of unsatisfactory low-quality images.

Costs and success probabilities associated with each task are shown in Table I. Costs are representative of human fatigue and the robot's battery consumption, both scaled from 1 to 5 units. We assume that tasks can be retried up to 5 times by human workers and up to 10 times by robots. Tasks are located at different locations (l_1-l_9) as shown in Figure 1b. Human workers ($w1, w2$) and robots ($r1, r2$) start at l_1 and can only move between adjacent locations. They must coordinate to complete all tasks while avoiding each other at all times after deployment. Each move incurs a cost of 1 unit and is allowed only if the destination location is unoccupied.

TABLE I: Task cost, success probability and maximum number of retries per task and agent (human worker or robot).

	Cost			Success prob.			Max. retries
Task	$t1$	$t2$	$t3$	$t1$	$t2$	$t3$	$t1, t2, t3$
Human	3	-	5	1.00	-	0.99	5
Robot	-	1	1	-	0.99	0.97	10

Addressing this problem entails producing a plan that enables successful mission completion while minimising human fatigue and energy costs. During operation, tasks might fail or requirements might change and the system is required to recover whenever possible to mitigate disruptions. Hence, maximising the probability of success while minimising cost and time is required. Finally, the task planner must ensure a success probability of 0.95.

III. Background

A. Numeric planning

A numerical planning model consists of a domain, specifying the world and its behaviours, and a planning problem, describing the specific task to be solved within that world [23], [24]. The tuple $\mathbb{D} = (\mathbb{T}, \mathbb{P}, \mathbb{F}, \mathbb{A})$ defines a domain, where $\mathbb{T}$ is a set of types, $\mathbb{P}$ a set of predicates, $\mathbb{F}$ a set of numerical-valued functions, and $\mathbb{A}$ a set of actions. Types categorise objects in the domain; predicates represent properties or relationships between objects; and actions describe possible state transitions, each comprising preconditions and effects. Functions represent quantities (e.g., distance, fuel level, travelling cost) and might depend on a finite number of typed objects. A numerical fluent is a function that models quantities that change over time.

A planning problem is defined by the tuple $\mathbf{P} = (\mathbf{O}, s_0, g, o)$ where $\mathbf{O}$ is the set of typed objects, s_0 is a set of grounded predicates that define the state of the world at the initial state, g is a set of predicates that must be satisfied in the goal state, and o is the planning optimisation objective [24]. Objects are instances of domain types.

PDDL. The planning model is aligned with the standard Planning Domain Description Language (PDDL) [16]. Different variants exist to accommodate classical, numeric or temporal planning requirements. The PDDL requirements for our task planning problem without uncertainty quantification are *:requirements :strips :typing :negative-preconditions :numeric-fluents*. Hence, we use PDDL2.1 [24], which allows conditional statements (e.g. $x > 0.5$) and numerical fluents (e.g., $x+ = 1$) in preconditions and effects, respectively. We refer interested readers to [24] for a full description.

Heuristic numerical planning. The Expressive Numeric Heuristic Search Planner (ENHSP) [25] is a planner that can solve problems defined in PDDL. ENHSP uses an expressive

representation for the planning problem and the heuristic search process and integrates numerical fluents to handle complex numeric constraints and effects efficiently. By combining domain-specific heuristics and search techniques, ENHSP can find optimal or near-optimal solutions more effectively than traditional planners in domains with numerical variables.

B. Probabilistic Model Checking

Probabilistic model checking (PMC) [12] is a formal verification technique for the quantitative analysis of probabilistic systems. PMC tools, such as PRISM [18] and Storm [19] automatically verify properties of such systems.

Discrete-time Markov chain (DTMC). A DTMC is a tuple $\mathcal{D} = (S, \bar{s}, \delta, AP, L)$ where S is a finite set of states and $\bar{s} \in S$ is an initial state; $\delta : S \rightarrow Dist(S)$ is a probabilistic transition function, mapping states to probability distributions over S; AP is a set of atomic propositions; and $L : S \rightarrow 2^{AP}$ is a state labelling function.

Markov Decision Process (MDP). An MDP extends a DTMC with actions, formalized as $\mathcal{Q} = (S, \bar{s}, Act, \delta, AP, L)$, where Act is a finite action set and $\delta : S \times Act \rightarrow Dist(S)$ is a partial transition function. For $s \in S$, let $Act(s) = \{a \in Act \mid \delta(s, a) \text{ is defined}\}$, with $Act(s) \neq \emptyset$.

Property Specification. To specify and analyse properties of probabilistic systems, we use quantitative extensions of temporal logic. In particular, we use Probabilistic Computation Tree Logic (PCTL) augmented with reward-based operators and leverage the PRISM property specification language [13], [18]. Given the set AP of atomic propositions, we identify four types of constructs: (i) *state formulae:* $\phi ::= true \mid a \mid \neg\phi \mid \phi \wedge \phi$, where $a \in AP$; (ii) *path formulae:* $\psi ::= \mathsf{X}\,\phi \mid \phi\,\mathsf{U}^{\leq k}\,\phi \mid \phi\,\mathsf{U}\,\phi$; (iii) *reward formulae:* $\rho ::= \mathcal{I}^{=k} \mid \mathcal{C}^{\leq k} \mid \mathcal{F}\,\phi$; and (iv) *queries:* $\Phi ::= \mathcal{P}_{\bowtie p}[\psi] \mid \mathcal{R}^{r}_{\bowtie q}[\rho]$, where $\bowtie \in \{<, \leq, >, \geq\}$, $p \in [0, 1]$, r is a reward structure, $q \in \mathbb{R}_{\geq 0}$, and $k \in \mathbb{N}$.

For example, the *quantitative* query $\mathcal{P}_{=?}[\mathcal{F}\text{"success"}]$ computes the probability of eventually reaching a state labelled "success", while $\mathcal{R}_{=?}[\mathcal{F}\text{"success"}]$ computes the corresponding expected reward. *Qualitative* queries check threshold conditions; for example, $\mathcal{P}_{\geq 0.9}[\mathcal{F}\text{"success"}]$ checks if the probability to reach a "success" state is at least 0.9. For MDPs, *min/max queries*, like $\mathcal{P}_{min/max=?}$, and their reward-based counterparts $\mathcal{R}_{min/max=?}$ enable computing the minimum/-maximum probability (or reward) over all MDP policies.

IV. Problem Formulation

We consider a CPHS comprising a set of agents (robots and humans) $\mathcal{A} = \{a_1, a_2, a_3, \ldots\}$ that reside in a world with locations $\mathcal{L} = \{l_1, l_2, l_3, \ldots\}$. A predicate $Path : \mathcal{L} \times \mathcal{L} \rightarrow \mathbb{B}$ indicates whether a path exists between location pairs with distance $dist_{l_i,l_j} \in \mathbb{R}$. The mission entails a set of tasks $\mathcal{T} = \{t_1, t_2, \ldots, \}$, each to be completed at a given location $TaskLoc : \mathcal{T} \times \mathcal{L} \rightarrow \mathbb{B}$. Each agent can perform actions $\mathcal{A}ct = \{Move, Do\}$, where $Move : \mathcal{A} \times \mathcal{L} \times \mathcal{L} \rightarrow \mathbb{B}$ and $Do : \mathcal{A} \times \mathcal{T} \times \mathcal{L} \rightarrow \mathbb{B}$. The probability of an agent successfully performing a task is given by $PSuccess : \mathcal{A} \times \mathcal{T} \rightarrow [0, 1]$, where 0 means that the agent cannot perform the task. In case of a failure, an agent can retry the task up to a maximum number of possible retries given by $Retry : \mathcal{A} \times \mathcal{T} \rightarrow \mathbb{N}_0$. Finally, the cost incurred by an agent to perform a task is given by $CostT : \mathcal{A} \times \mathcal{T} \rightarrow \mathbb{R}^{+}_{0}$. These functions neatly capture differences between the agents; for example, $PSuccess$ allows specifying that humans are more competent in dexterous tasks, such as picking grapes, requiring fewer retries than a robot, while $CostT$ allows specifying the robots' energy consumption or the increase in a human's fatigue. The CPHS mission includes the constraints C and optimisation objectives O shown in Table II. An optimisation objective defines if a quality attribute should be maximised or minimised provided that all constraints are met.

TABLE II: Mission requirements partitioned into constraints C_i and optimisation objectives O_j.

ID	Description
C_1	An agent can travel between locations iff a path exists.
C_2	An agent must be at the initial location of the path to travel.
C_3	Each location must be occupied by at most one agent at any time.
C_4	Agent a can be allocated task t iff $PSuccess(a, t) \geq \gamma, \gamma \in (0, 1]$
C_5	A plan is feasible iff all tasks $\mathcal{T}$ are completed.
C_6	A plan is feasible iff the probability of succeeding is at least p_{succ}.
O_1	The mission execution cost must be minimised.
O_2	The overall mission success probability must be maximised.

Definition 1 (Problem Specification): A CPHS planning problem is defined by the tuple $\mathcal{M} = (\mathcal{A}, \mathcal{L}, \mathcal{T}, \mathcal{A}ct, C, O, s_0)$, where $\mathcal{A}$ is a set of agents, $\mathcal{L}$ is a set of locations, $\mathcal{T}$ is a set of tasks, $\mathcal{A}ct$ is a set of actions, C and O are the sets of mission constraints and optimisation objectives, respectively, and s_0 is the initial CPHS configuration comprising the set of pending tasks and the initial location of the agents.

Definition 2 (Problem Solution): A solution to the planning problem $\mathcal{M}$ is a plan comprising the list of actions performed per agent a during its (finite) execution horizon H_a of the mission, represented by $\Pi = \left(\left(\mathcal{A}ct^{h}_{a}\right)_{h \in 1 \ldots H_a}\right)_{a \in \mathcal{A}}$.

Definition 3 (Pareto Front Solutions): Given the set of plans Π^S, the planning problem comprises finding the Pareto-optimal front PF induced by the Pareto-optimal set PS of plans that satisfy the C constraints and are Pareto-optimal with respect to the O optimisation objectives. Formally:

$$PS = \{\Pi \in \Pi^S | \bigwedge_{c \in C} B(\Pi, c) \wedge (\nexists \Pi' \in \Pi^S \bullet \Pi' \prec \Pi)\} \quad (1)$$

$$PF = \{(o, \Pi)_{o \in O} | \Pi \in PS\} \quad (2)$$

where $B(\Pi, c) \in \mathbb{B}$ is *True* if constraint c is satisfied by plan Π and *False*, otherwise, and $\prec$: $\Pi^S \times \Pi^S \rightarrow \mathbb{B}$ is the conventional dominance relation from Pareto optimisation [26].

V. Adaptive and Hybrid Task Planning

A. Overview

Our hybrid task planning approach realises a separation of concerns between the probabilistic and non-probabilistic problem elements of $\mathcal{M}$ to address its intrinsic complexity and scalability issues, and solves equations (1) and (2). Fig. 2

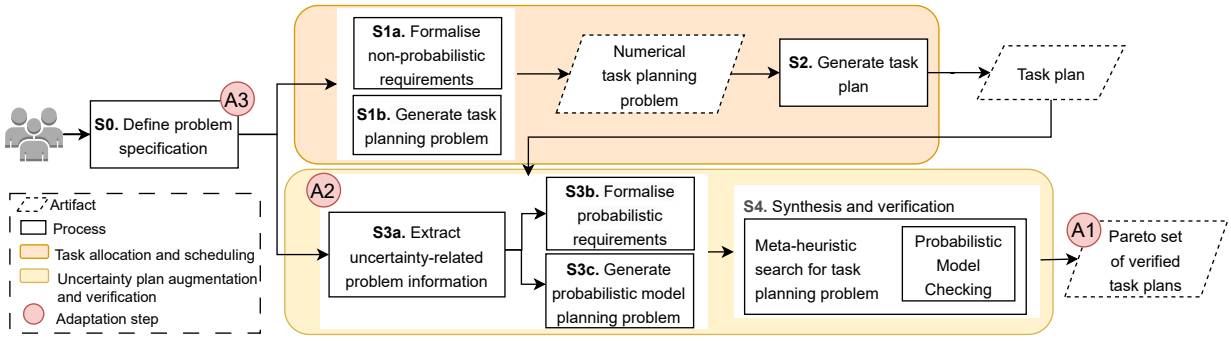


Fig. 2: Overview of our approach showing the required steps and generated artefacts, and adaptation relevant parts.

shows the key steps of our approach. Step *S0* involves defining the problem specification $\mathcal{M}$, typically provided by engineers, with input from stakeholders and domain experts. In step *S1*, this problem specification is transformed into a task planning problem and its non-probabilistic requirements are formalised. In step *S2*, a feasible plan is generated using non-probabilistic numeric planners and constraint solvers. Next, step *S3* extracts uncertainty-relevant information from the problem specification to both devise probabilistic-related requirements and augment the plan produced in step *S2* with such information (e.g., number of retries, the expected probability of success). Finally, step *S4* employs meta-heuristics to synthesise the Pareto front (PF) and the corresponding Pareto set (PS) of revised plans that are robust to uncertainties in the CPHS mission. During this step, probabilistic models of the task planning problem are automatically generated and formally verified using PMC. At runtime, our approach enables incremental adaptation in response to CPHS changes by selecting an alternative plan from the Pareto set (A1), re-executes the probabilistic synthesis (A2) if the original plan is still feasible and only resorts to adaptation from scratch (A3) if it cannot deal with the changes or guarantee a new set of constraints.

B. Approach

Problem Specification (*S0*): Starting at step *S0*, our approach involves defining the problem specification $\mathcal{M}$, encoding information needed for the CPHS mission. This model $\mathcal{M}$ conforms to the JSON format. The following example presents its structure for the vineyard CPHS mission from Section II.

Example 1 (Problem Specification). Figure 3a shows the syntax of the problem specification file for our vineyard case study. This is divided into locations $\mathcal{L}$, paths, tasks $\mathcal{T}$, agents $\mathcal{A}$, and mission constraints C and objectives O. Location elements comprise a unique identifier and an optional description. Path objects contain start and end location identifiers, an optional description, and a travelling cost ($distance = 1$). Task objects define groups of inter-related tasks, comprising a unique identifier (e.g., $t1$), an optional description and a list of tasks with identifiers ($t1l4$, $t1l6a$, $t1l6b$, $t1l7 \in \mathcal{T}$) and locations (e.g., $Taskloc(t1l4) = l4$). Agent objects are defined by a unique identifier, a type, $type \in \{$"worker","robot"$\}$, and a set of tasks the agent can perform. Each task has an identifier (e.g., $t1$ which refers to tasks $t1l4, t1l6a, t1l6b$ and $t1l7$), an expected cost (e.g., $CostT(t1l4) = 3$), a probability of success (e.g., $PSuccess(t1l4) = 1$) and the number of retries allowed per agent (e.g., $Retry(t1l4) = 5$). In our example, $worker1$ can perform tasks $t1$ and $t3$. Lastly, two constraints describe the probability of success (p_{succ}) given by $mission\_probability\_of\_success = 0.95$, and the allocation threshold $\gamma = 0.5$ ("min_assignment_probability").

Task Planning Problem Generation (*S1*): Step *S1* extracts the relevant requirements (*S1a*) for creating the domain $\mathbb{D}$ and the numeric planning problem $\mathbf{P}$ (*S1b*), cf. Section III-A. The relevant requirements from Table II are formalised to inform the execution of feasible actions $\mathcal{A}ct$ for the CPHS agents. Concretely, constraints C_1–C_3 apply to action *Move*,

$$\forall a : \mathcal{A};\ l_i, l_j : \mathcal{L} \mid Path(l_i, l_j) \wedge AgentLoc(a) = l_i \wedge Empty(l_j) \implies Move(a, l_i, l_j) \quad (3)$$

resulting in move actions that are available only when there is a path to the target location (C_1), the agent is at the initial location (C_2), given by $AgentLoc : \mathcal{A} \rightarrow \mathcal{L}$, and the target location is empty (C_3), given by Boolean $Empty : L \mapsto \mathbb{B}$.

For the *Do* action, only constraint C_4 applies, yielding

$$\forall a : \mathcal{A};\ l : L;\ t : T \mid AgentLoc(a) = l \wedge TaskLoc(t) = l \wedge \neg TaskDone(t) \wedge Psucc(a, t) \geq \gamma \implies Do(a, t, l) \quad (4)$$

resulting in do actions executable when an agent is at the task's location, the task is available and not yet done at that location, and provided that the agent's competency exceeds threshold γ.

The numeric planning problem is written in the PDDL language [16]. Figure 3b shows the PDDL domain $\mathbb{D}$, where the non-highlighted parts are generic and reusable. The domain comprises three types $\mathbb{T}$: locations, tasks and agents. Predicates $\mathbb{P}$ are defined for the agent's location (*agent_at*), paths (*path*), empty locations (*empty*), task locations (*task_loc*), and the completion of tasks (*task_done*). The two functions $\mathbb{F}$ specify the probability of an agent completing a task successfully (*p_success*) and track the travelling cost (*travel_dist*).

Actions $\mathbb{A}$ exist to move an agent and perform a task. The *Move* action precondition (Figure 3b, lines 13-15) requires an existing path, an agent at the initial location, and the next location to be empty. The effect of this action (lines 16-19) is that the agent moves to the specified location, the original location becomes empty, the destination location is no longer

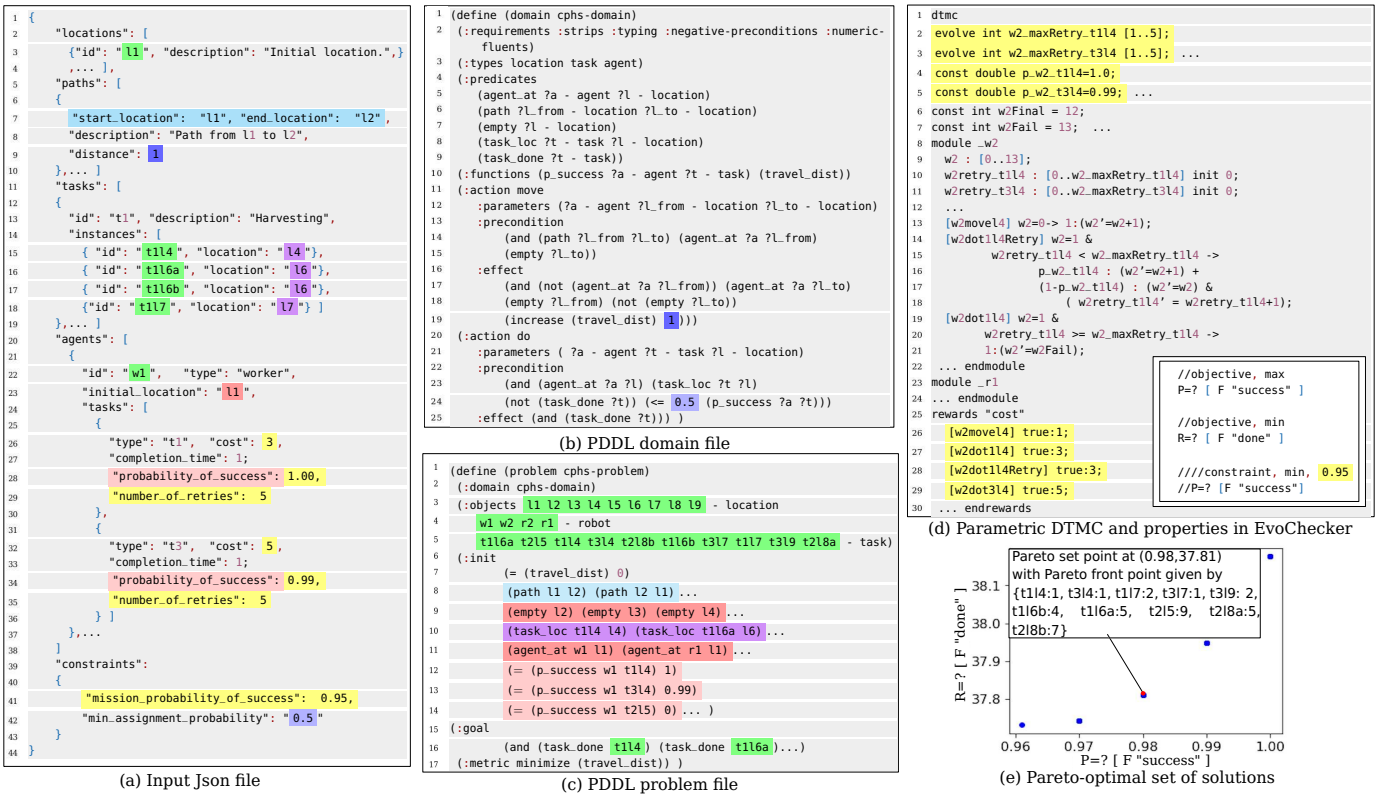


Fig. 3: Input file (a), generated PDDL files (b,c), parametric DTMC with probabilistic properties (d) and synthesis Pareto-optimal set of solutions (plans). The highlighted sections of the PDDL files use a colour palette based on the corresponding parts of the input file. The yellow parts refer to the task planning with uncertainty quantification part (see Figure 2).

empty and the total travel time increases (line 19). The *Do* action precondition (Figure 3b, lines 22-24) specifies that an agent must be present at the task location, the task must be in an undone state, and the agent must have a probability of successfully completing the task $\geq \gamma$. The function *p_success* associates an agent to its probability of task completion. In our example, as $\gamma = 0.5$, only robots with at least 50% chance of completing the task are allowed to take action *Do*. The effect of this action (line 25) results in the task being done.

Figure 3c shows an example of the planning problem $\mathbf{P}$ in PDDL. As before, non-highlighted parts are generic and reusable The problem-typed objects $\mathbf{O}$ are defined by the location, agent and task identifiers from the input file. The initial state s_{init} (lines 6-14) comprises the travel cost initialised to zero (line 7); all existing paths defined by the symmetric relation of paths defined in the input file (line 8); the empty locations comprising all locations except the initial location of agents (line 9); the task locations (line 10) defined from task instances in the input file; the agents initial locations (line 11); and the probabilities of each agent succeeding with the tasks capable of performing (lines 12-14). When an agent cannot perform a task, this is initialised to zero (line 14). Finally, goal g entails that all tasks are done, i.e., constraint C_5, (line 16) is met and the optimisation objective of minimising travel distance (line 17) is achieved.

Example 2 (Task Planning Model). We defined the task planning model in Figures 3b and 3c, showing the domain $\mathbb{D}$ and problem $\mathbf{P}$ in PDDL for our vineyard case study. The highlighted parts follow the colour palette of the corresponding parts in the input file from which these were obtained.

Task Plan Generation (*S2*): Given the task planning formulation, defined in PDDL by a domain $\mathbb{D}$ and a problem $\mathbf{P}$, step *S2* generates a feasible but uncertainty-agnostic task plan using an off-the-shelf numeric planner like ENHSP [25]. The outcome of the planning problem is a plan Π, comprising a sequence of actions that transitions the CPHS from one state to another, ultimately achieving the goal state (g).

Example 3 (Task plan). For our case study, an optimal plan that minimises the travelling cost is the following sequence of robot ($r1$) and worker ($w2$) actions: $\langle Move(w2, l_1, l_4)$, $Do(w2, t1l4, l_4)$, $Do(w2, t3l4, l_4)$, $Move(w2, l_4, l_7)$, $Do(w2, t1l7, l_7)$, $Do(w2, t3l_7, l_7)$, $Move(r1, l_1, l_4)$, $Move(r1, l_4, l_5)$, $Do(r1, t2l5, l_5)$, $Move(w2, l_7, l8)$, $Move(w2, l_8, l_9)$, $Move(r1, l_5, l_8)$, $Do(r1, t2l8a, l_8)$, $Do(r1, t2l8b, l_8)$, $Do(w2, t3l9, l_9)$, $Move(w2, l_9, l_6)$, $Do(w2, t1l6b, l_6)$, $Do(w2, t1l6a, l_6)\rangle$.

Uncertainty Augmentation and Task Plan Refinement (*S3*): Step *S3a* extracts uncertainty information from the problem specification to generate the relevant probabilistic requirements and a probabilistic model of the plan in steps *S3b* and *S3c*, respectively. For the probabilistic model, it extracts the probability of task success ($PSuccess$), costs ($distance$

and $CostT$) and allowed number of task retries ($Retry$). For the probabilistic requirements, it extracts the minimum probability of mission success p_{succ}.

Next, step *S3b* formalises requirements C_6 and O_1, O_2 in PCTL. Thus, it uses information about the task plan generated in step *S2* to define the final states: "success", where all agents completed their allocated tasks successfully, and "done", where no more actions are possible (due to mission success or failure). The PCTL requirements yielded are: C_6: $\mathcal{P}_{\geq p_{succ}}[\mathcal{F}$ "success"]; O_1: minimise the total expected cost (*min* $\mathcal{R}^{cost}_{=?}[\mathcal{F}$ "done"]); and O_2: maximise the probability of mission success (*max* $\mathcal{P}_{=?}[\mathcal{F}$ "success"]).

In step *S3c*, a parametric DTMC model $\mathcal{D}$ of the plan is created, enhanced with uncertainty information. To create this model, we extract the number of actions (n_{act}) assigned to an agent in the task plan from step *S2* and the number of tasks assigned to an agent (n_t) that allow for retries. Model $\mathcal{D}$ contains state variables for each agent in the task plan, defined by the tuple $s = (c, \overline{x},)$, where $c \in [0, n_{act} + 1]$ tracks the agent's actions increasing by one when an action is completed (c =0 to n_{act}), or when a task has failed ($c = n_{act} + 1$); and $\overline{x} = x_1, x_2, ...$ is a collection of variables such that x_i tracks the attempted retries of a task. The upper bound of variable x_i, i.e., $\hat{x_i}$, is a parameter of $\mathcal{D}$ and its value is synthesised in step *S4* to yield a robust plan. The maximum number of retries cannot exceed the limit defined in the problem specification (*step S0*), hence $\hat{x_i} \in [1, Retry(t)]$. The total number of state variables is the sum of agents selected to undertake tasks in the CPHS and the set of tasks with retries per agent.

Transitions in model $\mathcal{D}$ per agent a are defined based on the sequence of actions it must perform. Move actions increase c by one. Attempting a task t with $Retry(t) = 0$ results in $c += 1$ with probability $PSuccess(a, t)$ representing a succeeded attempt, and in $c = n_{act} + 1$ with probability 1-$PSuccess(a, t)$ when failed. Finally, attempting a task t with $Retry(t) > 0$ has two possible transitions depending on the value of $\overline{x}$. Let $x \in \overline{x}$ be the state variable tracking the number of retries for task t,

- if $x < \hat{x}$, then with probability $PSuccess(a, t)$ succeeds and $c += 1$, and with probability 1-$PSuccess(a, t)$ fails increasing the number of retries $x += 1$.
- else, it fails resulting in $c = n_{act} + 1$.

Rewards are defined over transitions so that a *Move* action incurs a travelling cost $distance$, and *Do* action a cost of $CostT(a, t)$ per agent-task pair (a, t). This step yields the model $\mathcal{D}$ and PCTL properties in the EvoChecker language [27], used for the task retries' synthesis in step *S4*.

Example 4 (Probabilistic Model). Figure 3d shows the task plan augmented with uncertainty data. Lines 2-3 define the retry parameters and their range for the meta-heuristic search. For instance, $w2$ assigned with task $t1l4$ which can retry up to five times is shown in line 2. Lines 4-5 capture the probability of success for each task assigned to each agent. Lines 6-7 define two values of state variable c for each agent, named as Final (i.e., robot succeeds with its tasks) and Fail. Each agent's behaviour is modelled as a separate module. Lines 8-22 describe worker *w2* actions. Lines 9-12 initialise the state variables c (line 9) and $\overline{x}$ (lines 10-12). The transition at line 13 shows a move action. Transition at lines 14-18 shows the attempt to complete task $t1l4$. Transition at lines 19-22 shows $w2$ failing to complete the task after the allowed number of retries. The reward structure (lines 25-30) shows the transition rewards with costs specified in the input file. The formalised properties in EvoChecker are shown in the white rectangle, with the minimum success probability value (0.95) extracted from the problem specification in step *S0*.

Synthesis and verification (*S4*): To generate a Pareto-optimal set of verified plan solutions, our approach uses meta-heuristic search to search for combinations of task retries using genetic algorithms (GAs) [26]. Chromosomes represent the number of retries for each task. Internally, at the GA evaluation stage, our approach generates concrete DTMC models of the task plan with the parameters $\hat{x}$ fixed, and employs PMC to analyse the set of extracted PCTL properties (bottom right of Figure 3d). The Pareto-optimal front PF contains the values of the task plan's expected cost and success probability for objectives O_1 and O_2, while the Pareto-optimal solutions set PS contains the values of the number of retries per task.

Example 5 (Pareto Front). Figure 3e shows the Pareto-optimal front found with 25 different combinations of decision points clustered into five regions. A single solution depicted in red shows the task retries allowed as a dictionary. The GA was set to 200 evaluations with a population size of 25.

Incremental Adaptation (*A1–A3*): At runtime, the implemented plan is adjusted in response to changes, leveraging the self-management and adaptation capabilities for CPHS offered by the MAPE-K loop [28]. The Monitor checks for the following changes and disruptions:

- C1: a task failure;
- C2: a change in the minimum success probability p_{succ};
- C3: a change in the minimum assignment probability γ;
- C4: a change in an agent's probability $PSuccess(a, t)$.

The Analyse phase receives the change type C$\in${C1,C2,C3,C4} and analyses the current state of the system compared to its status in the knowledge base. Based on the analysis result, a strategy is selected by the Plan phase.

The adaptation process is shown in Algorithm 1. Given a change C, a deployed plan Π, the time of the change τ and an initial set of verified and robust plans PS (i.e., the Pareto front set), our approach results in an adaptation related to different parts of the task planning approach (A1, A2 or A3) as shown in Figure 2. For all changes, our algorithm follows a similar data flow: (a) reduce the PS set by removing incorrect plans that do not comply with the change, (b) if our current plan is part of this PS, continue without adaptation, (c) otherwise, check if another plan exists ($PS.nonempty$) and deploy a new plan. If no plan exists, re-plan from step *S0*.

Let $\Pi_j \in PS$ be a verified plan. Let $\Pi_j[k], k \in \mathbb{R}^{+0}$ denote the action (starting) at time k in Π_j; $\Pi_j[: k]$ all actions up to time k and $\Pi_j[k :]$ from k onwards. Here, we use the notion

of a time unit as the duration of an action. A change occurs at time τ if it happens at time τ or later, but before $\tau + 1$.

When C1 occurs at time τ (lines 1-2), REDUCE_PS_TF(PS, τ, t) selects a new set of plans $PS' \subseteq PS$ with plans that comply with the current progress ($\Pi_j[: \tau] = \Pi[: \tau]$) and has (a retry of) do task t at the next step $\Pi_j[\tau + 1]$. Similarly, when C2 occurs (lines 3-4), REDUCE_PS_PSUCC(PS, τ, p'_{succ}) selects a new set of plans $PS' \subseteq PS$ with plans that comply with the current progress as in C1 and that have a mission success probability greater than or equal to the new p'_{succ}.

When C3 occurs (lines 5-6), REDUCE_PS_PASSIGN(PS, τ, t, γ') first checks if any (undone) task in $\Pi[\tau :]$ has $PSuccess(a, t) \leq \gamma'$. If this holds, it yields an empty PS' triggering line 15 in Algorithm 1. This change entails that as all $\Pi_j \in PS$ follow the same allocation from *S2*, when the allocation is invalid all plans become invalid. Finally, when C4 occurs, REDUCE_PS_PTASK(PS,t, a, p') first checks whether the new probability p' of succeeding with t by agent a applies to any (undone) tasks in $\Pi[\tau :]$. If it does, the function proceeds similarly to C3, checking if the new probability $p' > \gamma$; if not, the process is reset from step *S0*. If the condition holds, a new set $PS' \subseteq PS$ is generated from *S3* for the tasks in $\Pi[\tau :]$, with the updated probabilistic information p'.

Adaptations A1-A3 occur at various stages of the task planning process, as illustrated in Fig. 2. A minor adaptation A1 might be required by changes C1 or C2, requiring only plans from PS. A medium adaptation A2 might be required by C3 or C4 when plans follow the allocation constraints but new probabilistically verified plans must be obtained. A major adaptation A3 might be required by C3 or C4 when allocations are no longer valid as they violate constraints related to the allocation threshold γ.

Algorithm Correctness. Algorithm 1 terminates for a finite $|PS|$ and without any loops. A finite $|PS|$ is ensured as PS results from the synthesis step *S4* (see Fig. 2), which is set to a finite number of evaluations. The correctness of the solutions produced by the adaptation algorithm is ensured through the following guarantees: (a) changes C1 and C2 only modify the plan by selecting from previously generated correct plans in PS; (b) change C3 solely impacts the allocation process (as the γ threshold only impacts the task allocation), hence adaptations result in a new task planning problem from *S0*, with the new problem specification guarantees a new correct set of solutions; (c) change C4 checks compliance with both planning and probabilistic constraints. This change can proceed as C3, or generate new plans from *S3*. As the input plan in *S3* guarantees compliance with the planning constraints, from *S3*, a new set of verified plans (in *S4*) is then guaranteed to be correct.

Although we can guarantee the correctness of the generated plans through our adaptation process, we cannot guarantee that a plan exists, for example, when too strict requirements are set (e.g., minimum probability of success >0.99) or the planning problem is unsolvable (e.g., when no path exists to reach one of the tasks). Moreover, to reduce complexity, we limit our approach to only one change per time unit for this paper. This assumption holds for agents working in parallel. A plan Π contains the actions of all agents, while an individual agent's plan contains only that agent's actions. The key difference is that multiple actions can be executed simultaneously. By restricting ourselves to one change per time unit, adaptations impact the plans of all agents at once. Future work will investigate ways to relax this constraint.

Algorithm 1 Task plan adaptation algorithm

Require: Monitored change C
Require: Global var. Π ▷ current plan
Require: Global var. τ ▷ time of change
Require: Global var. PS ▷ set of verified plans
Require: Global var. M ▷ Definition 1 (Problem specification)
1: **if** C=C1 **then** ▷ C1: task t failure
2: $PS' \leftarrow$ REDUCE_PS_TF(PS,τ,t)
3: **else if** C=C2 **then** ▷ C2: mission success p'_{succ} change
4: $PS' \leftarrow$ REDUCE_PS_PSUCC(PS,τ, p'_{succ})
5: **else if** C=C3 **then** ▷ C3: prob. of assignment γ' change
6: $PS' \leftarrow$ REDUCE_PS_PASSIGN(PS,t,γ') ▷ via PMC
7: **else if** C=C4 **then** ▷ C4: task t success from a changed to p'
8: $PS' \leftarrow$ REDUCE_PS_PTASK(PS,t, a, p') ▷ via PMC
9: **end if**
10: **if** ($\Pi \in PS'$) **then**
11: **return** Π ▷ No adaptation required (N/A)
12: **else if** ($PS' \neq \emptyset$) **then**
13: **return** SELECT_NEW_PLAN(PS') ▷ A1/A2
14: **else**
15: $PS \leftarrow$ GENERATE_NEW_PLANS(M, Π, τ) ▷ A3
16: **return** SELECT_NEW_PLAN(PS)
17: **end if**
18: Return

VI. Evaluation

A. Research questions

RQ1 (Effectiveness): How effective is our hybrid approach, in terms of solution quality and computational performance, when compared to an optimal solution derived from a full MDP model of the problem specification? To mitigate the state explosion problem, our approach leverages heuristic numerical planners for task partitioning and scheduling, and verifies augmented plans using PMC. We compare the effectiveness of our verified plan solutions against those from a full MDP (optimal) baseline synthesised using PRISM [18].
RQ2 (Adaptation): How effective is our approach in dealing with runtime changes by adapting the plan when necessary and reducing latency costs? We assess the effectiveness of our runtime adaptation to changes C1-C4 and discuss reductions in replanning latency.
RQ3 (Efficiency): How computationally efficient is our hybrid approach when we increase the number of tasks and agents in the problem specification? We investigate the efficiency of our approach when increasing the planning problem in the number of tasks and the number of agents.

B. Experimental setup

We assess the effectiveness, adaptability, and efficiency of our hybrid approach and compare it against a full MDP baseline derived from the problem specification. We use several

TABLE III: Hybrid approach compared to a full-MDP model

Planning problem					Full MDP verification approach				Hybrid approach				
ID	$\|T\|$	$\|L\|$	retries	agents	#states	#trans	PMC execution mean time (SD) [s]	$\|PS\|$	# states*	# trans*	Numerical planner mean time (SD) [s]	Verif. & search mean time (SD) [s]	$\|PS\|$ mean (SD)
$\mathcal{M}_1$	3	9	1	1 human, 1 robot	35,081	172,714	196.69 (85.38)	3	14	17	0.29 (0.02)	104.2 (1.85)	1 (0.00)
$\mathcal{M}_2$	2	6	3	1 human, 1 robot	134,581	541,767	128.53 (53.93)	3	36	48	0.33 (0.02)	105.7 (5.94)	4 (0.00)
$\mathcal{M}_3$	3	9	CS	2 human, 2 robot	270,100,547	1,351,524,022	Timeout	-	398	553	0.33 (0.02)	884.2 (138.06)	47.80 (7.87)
$\mathcal{M}_4$	10	9	CS	2 human, 2 robot	OOM	OOM	-	-	1,825,976	3,822,145	2.02 (0.08)	128.1 (10.31)	39.93 (3.17)

* from DTMC with max. number of task retries per agent (i.e., largest possible model checked).
OOM = out of memory. CS = retries set as in Table I. All tasks in $\mathcal{M}_{1-2}$ are type $t3$. Model $\mathcal{M}_4$ is our vineyard case study (Section II)

problem instances of varying complexity and numbers of tasks and agents, including those (M1, M2, M3, M4) from Table III.

RQ1 (Effectiveness). To evaluate the effectiveness of our approach compared to an optimal baseline, we generated a full MDP from the problem specification in Section IV. By obtaining policies from this full MDP, we compute optimal Pareto fronts that serve as baselines for comparison. We apply our approach to several problem instances of increasing complexity and measure solution quality, execution times (mean and standard deviation), and the number of solutions over 30 runs. We also compare the model sizes of the MDP, and the DTMC generated in step *S4* with parameters set to the maximum number of retries.

RQ2 (Adaptation). To investigate how our approach manages runtime changes (e.g., task failures, probability value changes), similar to [29], we present a scenario where all changes (C1-C4) and adaptations (A1-A3) happen during a typical execution of our industrial vineyard case study. We assessed latency reduction when applying different adaptation strategies at different steps of our hybrid planning approach.

RQ3 (Efficiency). We examine how the computational efficiency of our approach changes as the problem size grows. Thus, we vary the number of tasks (10 to 13) and agents (2, 4, and 6) from our vineyard case study. Tasks were assigned random locations, while agents were deployed at location l_1. We report metrics as those for RQ1; however, given the high variation in the execution time results, we reported the geometric median following standard statistical guidance [30].

Implementation details. All experiments are run on an Intel Core i5 @2.40 GHz machine with 8 GB RAM, under Ubuntu 22.04.5 LTS 64-bit. Our hybrid approach is fully automated. For step *S2*, we use a model-to-model (M2M) transformation implemented in Python to generate the PDDL files. We produce a task plan using the ENHSP numerical solver in the Unified Planning library [31]. A second M2M transformation generates the EvoChecker input files. Multi-objective optimisation at steps *S3-S4* is performed using EvoChecker [27], [32], which orchestrates PRISM and the JMetal [33] search framework with NSGA-II configured to 150 evaluations and a population size of 30 following guidelines [34]. In RQ3, as we were interested in efficiency, we increased the population size to 100. Experiments use PRISM set to 8 GB memory limit. Our open-source code is available at [35].

C. Results and Discussion

RQ1 (Effectiveness). To generate the optimal Pareto front, we tested the multi-objective verification capabilities of the widely-used PMC tools PRISM [18] and Storm [19]. However, neither tool could solve the multi-objective verification problem outlined in Section V-B—reachability rewards for multi-objective properties are not supported. Thus, we verified the bounded cumulative reward $\mathcal{R}_{min}$=?[$\mathcal{C}$<=20] and the reachability property $\mathcal{P}_{max}$=?[$\mathcal{F}$ "success"] simultaneously for both our approach and PRISM/Storm using the full MDP.

For the first two models (M1 and M2) we were able to generate Pareto solutions for both the full MDP and our approach. However, due to an explosion in the size of the MDP models, we could not do the same for M3 and M4—M3 was parsed but could not be checked (Table III). Our hybrid approach successfully generated plans for all models. Since we resolved the allocation and scheduling separately, the probabilistically checked models were reduced by several orders of magnitude. For example, M1 was reduced from 35,081 states to 14, and 172,714 transitions to 17. This also resulted in a lower execution time compared to the full MDP. Figures 4a and 4b show that our approach generates dominated solutions while the full MDP yields the optimal Pareto front. We continue evaluating each model's results separately.

For M1 we only allowed for one retry. Our approach first generated a plan to deploy only one robot. Therefore, we only find one single solution where the robot can retry all three tasks once. In comparison, the full MDP resulted in plans to deploy the robot or the human worker. These are the red diamonds close to $\mathcal{P}max$=?[$\mathcal{F}$ "success"]=0.99. A human has a slightly higher probability of completion (0.99) than the robot (0.97) but incurs a higher expected cumulative cost.

For M2, our approach also deploys a single robot. However, as more retries are allowed, the Pareto front results in four solutions. Calculating the total number of solutions for this allocation results in 3^2 (multiplying the number of retries per task), this is only 9 solutions to search by the GA—five dominated and four non-dominated. Models M1 and M2 are too small to benefit from our hybrid approach implementation resulting in unnecessary latencies. For these smaller problems, exhaustive searches can be considered instead.

For M3 and M4, we obtained several verified plans as shown in their Pareto front plots. Moreover, each of these Pareto front points corresponds to more than one combination of task retries. For example, for model M3 we obtained 8.51 Pareto front points on average (the solution in Fig. 4c shows these clustered near five points), while these correspond to a total of 27.77 different verified plans on average. These results are the basis for our adaptation algorithm: we can change

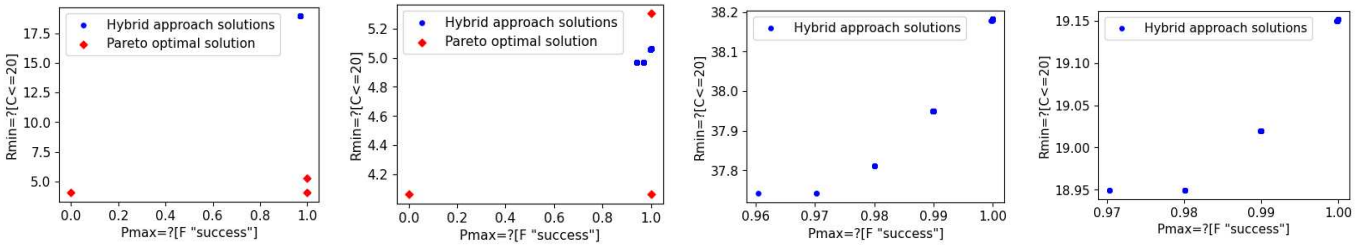


Fig. 4: Pareto front from optimal baseline and our hybrid solution for problem instances. From left to right: $\mathcal{M}_1, \mathcal{M}_2, \mathcal{M}_3, \mathcal{M}_4$.

the plan to one with more retries, if needed, while ensuring plan correctness, even for complex models where full MDP verification is not feasible—though at the cost of optimality.

RQ2 (Adaptation). Our adaptation algorithm accommodates a finite number of monitored changes C1-C4 and adaptation strategies A1-A3. Therefore, these were exhaustively tested in our provided tool [35], where changes are injected along the plan timespan. Here, we present the adaptation results on a continuous run for our industrial case study showing all monitor changes and adaptations (see Fig. 5). Worker 2 starts by travelling to $l4$.

- **Case N/A:** At time 1 (i.e., between 1 and 2 time units as explained in Sec. V-B), worker $w2$ attempts task $t1l4$ and fails. The task failure is already anticipated in the current plan, allowing for a $t1l4$ retry.
- **Case A1:** At time 2, task $t1l4$ fails during a retry attempt. No second retry is allowed. Plan B, which allows for two $t1l4$ retries and is consistent with the current plan A progress, is found in the verified plans. After the second retry, the task succeeds and proceeds with $t3l4$.
- **Case N/A 2:** At time 4, a change in the probability of mission success $p_{succ} = 0.8$ results in no adaptation, as all plans already comply with such constraint update.
- **Case A2:** At time 11, the worker's probability $PSuccess(w2, t3l9)$ changes to 0.89 as tiredness start building up. This modifies the probability of plan success and a new verified plan is synthesised at *S3* for the remaining actions for both $w2$ and $r1$ agents (due to space limitations, we show only $w2$). A new plan C is deployed, which follows the same task and travel sequence —up to this point— as Plans A and B.
- **Case A3:** At time 13, a change $\gamma' = 0.9$ invalidates all verified plans obtained from *S2* , as $w2$ can succeed with $t3l9$ only with 0.89 probability. Hence, a new plan is generated from *S0* and the completion of tasks continues.

This representative scenario shows how the planning adaptation transpires at runtime to avoid a task failure or the violation of a requirement. In RQ1, we discuss the computational costs of the different stages of the generation of plans. Here, we show that some of these costs can be avoided when no adaptation, or adaptations A1 and A2, are performed. At times 1 and 2, no replanning costs were incurred. At time 11, the execution time for numerical planning was avoided. Meanwhile, the execution of the hybrid planning was only required at time 12, emphasizing the efficiency gains achieved via the adaptation of hybrid planning techniques. The benefit of this incremental

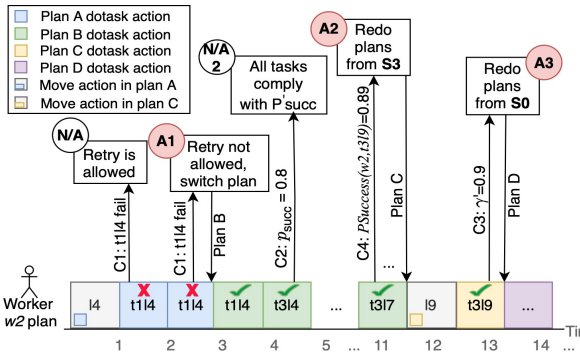


Fig. 5: Sample scenario showing adaptation cases A1-A3.

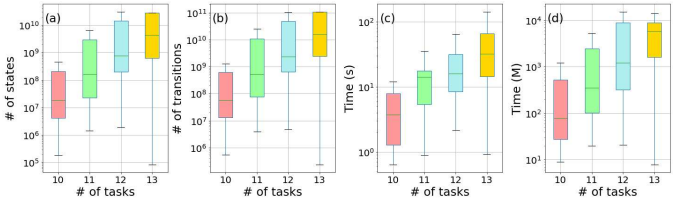


Fig. 6: a) DTMC states, b) DTMC transitions, c) Numerical planner time, and d) Verif. & search time for 10-13 tasks.

adaptation shows that replanning from scratch is not always necessary. In fact, task planning adaptation can leverage results from different stages of the planning process (such as the structure of a plan and the expanded set of verified plans), provided that the stages generating these results can guarantee their correctness after the change is applied.

RQ3 (Efficiency). We conducted experiments for different combinations of the number of tasks and robots, with the results presented in Table IV and Figure 6. The planner's computation time increases consistently for increments in both of these parameters, demonstrating relatively low variation in comparison to the verification and search counterpart. This is expected as verification and search perform multiple GA evaluations of the possible feasible plans. The verification and search times show significant variability depending on the number of tasks—in particular, on the number and locations of the tasks (see Fig. 6c and d). As we randomly generated the task locations, these resulted in multiple planning problems, some easier to solve than others. This diversity explains the consistent increase in reported mean, but also in the standard deviation (SD). For instance, for 13 tasks, we have a variation in model size of $\pm 10.34 \times 10^{11}$ and $\pm 38.20 \times 10^{11}$ geometric SD for states and transitions, respectively. The overall time required for verification and search increases as the total

TABLE IV: Computational times and model sizes for RQ3.

# tasks	# agents	Num. planner time (SD) [s]	Verif. & search time (SD$\times 10^4$) [s]	# states $\times 10^7$ (SD$\times 10^9$)	# transitions $\times 10^9$ (SD$\times 10^{10}$)
10	4	3.71 (11.85)	4,760.4 (0.06)	1.89 (0.43)	0.06 (0.15)
11	4	14.11 (10.35)	21,301.05 (16.2)	18.64 (20.00)	0.55 (5.17)
12	4	16.02 (35.12)	73,001.82 (28.2)	84.39 (547.00)	2.62, (155.00)
13	4	32.43 (49.70)	348,623.4 (28.8)	684.40 (1034.00)	24.09 (382.0)
# tasks	# agents	Num. planner time (SD) [s]	Verif. & search time (SD) [m]	# states (SD)	# transitions (SD)
10	2	0.53 (0.04)	7.54 (11.65)	1,872,856	3,915,885
10	4	2.10 (0.09)	13.92 (2.38)	1,825,976	3,822,145
10	6	9.39 (0.19)	15.31 (1.38)	1,872,856	3,915,885

number of tasks grows, driven by the corresponding increase in plan complexity. The variations in the number of states and transitions follow a similar trend (Fig. 6a and b). This determines the complexity of the probabilistic model, which increases on average with additional tasks.

For agent variations, all agents starting from l_1 resulted in a single plan per robot count. This results in a single probabilistic model size reported with retry parameter values set to the maximum number of retries. Increasing the number of agents increases the allocation complexity which, in turn, increases the heuristic planners' time. This increase is expected, as a greater number of agents introduces additional decision variables and constraints into the planning process. However, the verification and search times do not exhibit a discernible trend, highlighting their dependence on the sequential plan generated by the numerical planner. For instance, 2-agent and 6-agent cases yielded similar states and transitions, as only 2 agents (1 human, 1 robot) were actively assigned tasks in both scenarios. These findings indicate that the complexity of the verification part is driven by the number of robots in the numerical planning solution rather than by the initial number of agents. Furthermore, increasing either the number of tasks or agents leads to a significant rise in computational time.

Threats to validity. We reduce **construct validity threats** due to simplifications in the specifications of the planning problem and adaptation tactics selection by using an industrial case study from our ongoing European-project collaboration. This paper is aligned with their vision of long-term adaptation for their CPHS tasks. We mitigate threats in the adaptation cases of Algorithm 1 by exhaustively testing each of the possible adaptation scenarios automatically through our tool.

We mitigate **external validity threats** that could affect the generalisation of our approach by using off-the-shelf numerical planners and the search-based software engineering tool EvoChecker [27]; the latter using the widely used PRISM model checker internally. This also mitigates introducing errors in the relevant search algorithms underpinning our approach. We use the widely used PDDL2.1 [16] language to define our numerical planning problem, and the PRISM-based EvoChecker language for the multi-objective optimisation part. For consistency, we verified the full MDP in PRISM [18].

VII. Related work

Multiple hybrid approaches have been proposed to address the multi-agent task planning problem and variants (task allocation, task scheduling and motion planning [36]) [37], [38], [39]. Reviews such as [40] present an overview of some of these hybrid approaches. These include combining GAs with mixed integer linear programming [41], Branch and Bound (BnB) [42], Q-learning [43], game-theory [44], clustering [45], [46] and simulated annealing [47]. KANOA [48], [21] combines constraint solving with GA and PMC for the allocation and scheduling of tasks. In contrast, our solution uses GA and PMC for the verification of probabilistic properties and task retry synthesis, while efficiently pre-solving the numerical task planning problem through heuristic methods. PMC has become a prominent technique for ensuring reliability under uncertainty [11], [49], successfully applied in task planning problems [50]. However, its limited adoption remains due to the state explosion problem, motivating our approach.

Prior work on task planning under uncertainty, emphasising cyber-physical systems, has been studied using various approaches [8], [51], [52]. Random resource availability is considered in [5], while [53] considers dynamic task allocation through replanning or task reallocation. For self-adaptive systems, frameworks such as ROSRV, [54], and runtime verification approaches [55], [56] can check the safety and temporal properties during system execution. However, these and many similar approaches [5] do not provide explicit adaptation strategies. Finally, for CPHS, different variants, including human-in-the-loop [15], human-on-the-loop [57], [38], and human-machine-interaction [58] for task planning, have been extensively studied. Specifications such as"*humans can reject plan solutions*" [22] will be explored in future work.

VIII. Conclusions and Future Work

We presented a hybrid adaptive task planning approach to generate correct and verified CPHS plans. By combining numerical planning methods and probabilistic model checking, our approach effectively decomposes task planning into deterministic and uncertainty-augmented stages. Integrating meta-heuristic search enables synthesising Pareto-optimal plans that manage uncertainties while meeting formal probabilistic requirements. Our approach produced plans for larger problems that policy synthesis based on full MDP and verification via probabilistic model checking failed. Incremental adaptation also yields computational time savings when possible.

Future work will extend our approach to support multiple simultaneous changes and other requirements relevant to our CPHS problem, e.g., human fatigue levels or sensor failures instrumented through real-time feedback from agents. We will also investigate using advanced numeric planners [59] and decentralised adaptation [49]. Finally, we will explore using graphical editors for mission specification [60].

Acknowledgements. This research was supported by the Europe Horizon projects AI4Work (101135990) and SOPRANO (101120990), and by the ULTIMATE project funded by the Advanced Research and Invention Agency, UK. We thank Alessandro Valentini, Elisa Tosello and Andrea Micheli from Fondazione Bruno Kessler for their helpful support on the numerical planner, and Quinta Do Castro and the University of Trás-os-Montes for their help with the case study.

REFERENCES

[1] A. Annaswamy, P. Khargonekar, F. Lamnabhi-Lagarrigue, and S. Spurgeon, *Cyber-Physical-Human Systems: Fundamentals and Applications*, ser. IEEE Press Series on Technology Management, Innovation, and Leadership. Wiley, 2023.

[2] P. Tokekar, J. Vander Hook, D. Mulla, and V. Isler, "Sensor planning for a symbiotic UAV and UGV system for precision agriculture," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1498–1511, 2016.

[3] J. Lee, B. Bagheri, and H.-A. Kao, "A cyber-physical systems architecture for industry 4.0-based manufacturing systems," *Manufacturing Letters*, vol. 3, pp. 18–23, 2015.

[4] R. Calinescu, J. Cámara, and C. Paterson, "Socio-cyber-physical systems: Models, opportunities, open challenges," in *2019 IEEE/ACM 5th International Workshop on Software Engineering for Smart Cyber-Physical Systems (SEsCPS)*. IEEE, 2019, pp. 2–6.

[5] R. Sanchez, J. Troya, and J. Camara, "Automated planning for adaptive cyber-physical systems under uncertainty in temporal availability constraints," in *International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2024, pp. 14–24.

[6] J. Camara, S. Hahner, D. Perez-Palacin, A. Vallecillo, M. Acosta, N. Bencomo, R. Calinescu, and S. Gerasimou, "Uncertainty flow diagrams: Towards a systematic representation of uncertainty propagation and interaction in adaptive systems," in *Proceedings of the 19th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2024, pp. 37–43.

[7] D. Weyns, R. Calinescu, R. Mirandola, K. Tei, M. Acosta, N. Bencomo, A. Bennaceur, N. Boltz, T. Bures, J. Camara *et al.*, "Towards a research agenda for understanding and managing uncertainty in self-adaptive systems," *ACM SIGSOFT Software Engineering Notes*, vol. 48, no. 4, pp. 20–36, 2023.

[8] X. Zhao, S. Gerasimou, R. Calinescu, C. Imrie, V. Robu, and D. Flynn, "Bayesian learning for the robust verification of autonomous robots," *Communications Engineering*, vol. 3, no. 1, p. 18, 2024.

[9] M. Zhang, B. Selic, S. Ali, T. Yue, O. Okariz, and R. Norgren, "Understanding uncertainty in cyber-physical systems: A conceptual model," in *Modelling Foundations and Applications*, A. Wąsowski and H. Lönn, Eds. Cham: Springer International Publishing, 2016, pp. 247–264.

[10] R. Calinescu, C. Ghezzi, M. Kwiatkowska, and R. Mirandola, "Self-adaptive software needs quantitative verification at runtime," *Commun. ACM*, vol. 55, no. 9, p. 69–77, Sep. 2012. [Online]. Available: https://doi.org/10.1145/2330667.2330686

[11] M. Kwiatkowska, G. Norman, and D. Parker, "Advances and challenges of probabilistic model checking," in *2010 48th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, 2010, pp. 1691–1698.

[12] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking," in *Formal Methods for Performance Evaluation: 7th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM 2007, Bertinoro, Italy, May 28-June 2, 2007, Advanced Lectures*, M. Bernardo and J. Hillston, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 220–270. [Online]. Available: https://doi.org/10.1007/978-3-540-72522-0_6

[13] M. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic model checking and autonomy," *Annual review of control, robotics, and autonomous systems*, vol. 5, no. 1, pp. 385–410, 2022.

[14] S. Gerasimou, J. Cámara, R. Calinescu, N. Alasmari, F. Alhwikem, and X. Fang, "Evolutionary-guided synthesis of verified pareto-optimal mdp policies," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 842–853.

[15] J. E. Fischer, C. Greenhalgh, W. Jiang, S. D. Ramchurn, F. Wu, and T. Rodden, "In-the-loop or on-the-loop? Interactional arrangements to support team coordination with a planning agent," *Concurrency and Computation: Practice and Experience*, vol. 33, no. 8, p. e4082, 2021.

[16] H. L. Younes and M. L. Littman, "PPDDL1. 0: An extension to PDDL for expressing planning domains with probabilistic effects," *Techn. Rep. CMU-CS-04-162*, vol. 2, p. 99, 2004.

[17] R. Calinescu, S. Gerasimou, K. Johnson, and C. Paterson, "Using runtime quantitative verification to provide assurance evidence for self-adaptive software: advances, applications and research challenges," in *Software Engineering for Self-Adaptive Systems III. Assurances: International Seminar, Dagstuhl Castle, Germany, December 15-19,*. Springer, 2017, pp. 223–248.

[18] M. Kwiatkowska, G. Norman, and D. Parker, "Prism 4.0: Verification of probabilistic real-time systems," in *Computer Aided Verification: 23rd International Conferenc (CAV). Proceedings 23*. Springer, 2011, pp. 585–591.

[19] C. Hensel, S. Junges, J.-P. Katoen, T. Quatmann, and M. Volk, "The probabilistic model checker storm," *Int. J. Softw. Tools Technol. Transf.*, vol. 24, no. 4, p. 589–610, Aug. 2022. [Online]. Available: https://doi.org/10.1007/s10009-021-00633-z

[20] J. Cámara, "Haiq: Synthesis of software design spaces with structural and probabilistic guarantees," in *Proceedings of the 8th International Conference on Formal Methods in Software Engineering*, 2020, pp. 22–33.

[21] G. Vázquez, R. Calinescu, and J. Cámara, "Scheduling of missions with constrained tasks for heterogeneous robot systems," in *Formal Methods for Autonomous Systems (FMAS)*, ser. Electronic Proceedings in Theoretical Computer Science, vol. 371, Berlin, Germany, September 2022, pp. 156–174.

[22] B. Wang, P. Zheng, Y. Yin, A. Shih, and L. Wang, "Toward human-centric smart manufacturing: A human-cyber-physical systems (hcps) perspective," *Journal of Manufacturing Systems*, vol. 63, pp. 471–490, 2022.

[23] M. Ghallab, D. Nau, and P. Traverso, *Automated planning and acting*. Cambridge University Press, 2016.

[24] M. Fox and D. Long, "PDDL2. 1: An extension to PDDL for expressing temporal planning domains," *Journal of artificial intelligence research*, vol. 20, pp. 61–124, 2003.

[25] E. Scala, P. Haslum, S. Thiébaux, and M. Ramirez, "Interval-based relaxation for general numeric planning," in *ECAI*. IOS Press, 2016, pp. 655–663.

[26] C. C. Coello, "Evolutionary multi-objective optimization: a historical view of the field," *IEEE computational intelligence magazine*, vol. 1, no. 1, pp. 28–36, 2006.

[27] S. Gerasimou, G. Tamburrelli, and R. Calinescu, "Search-based synthesis of probabilistic models for quality-of-service software engineering (t)," in *IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 319–330.

[28] J. O. Kephart and D. M. Chess, "The vision of autonomic computing," *Computer*, vol. 36, no. 1, pp. 41–50, 2003.

[29] S. Gerasimou, R. Calinescu, and A. Banks, "Efficient runtime quantitative verification using caching, lookahead, and nearly-optimal reconfiguration," in *Proceedings of the 9th international symposium on software engineering for adaptive and self-managing systems*, 2014, pp. 115–124.

[30] P. J. Fleming and J. J. Wallace, "How not to lie with statistics: the correct way to summarize benchmark results," *Commun. ACM*, vol. 29, no. 3, p. 218–221, Mar. 1986. [Online]. Available: https://doi.org/10.1145/5666.5673

[31] Rovetta, Alberto and Trapasso, Alessandro and Valentini, Alessandro and et al., "Unified planning documentation," 2024, accessed: 2024-12-02. [Online]. Available: https://unified-planning.readthedocs.io/en/latest/index.html

[32] S. Gerasimou, R. Calinescu, and G. Tamburrelli, "Synthesis of probabilistic models for quality-of-service software engineering," *Automated Software Engineering*, vol. 25, no. 4, pp. 785–831, 2018.

[33] J. J. Durillo and A. J. Nebro, "jmetal: A java framework for multi-objective optimization," *Advances in engineering software*, vol. 42, no. 10, pp. 760–771, 2011.

[34] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[35] Project's GitHub: https://github.com/Gricel-lee/EfficientPlanAdaptation.

[36] A. Messing, G. Neville, S. Chernova, S. Hutchinson, and H. Ravichandar, "Grstaps: Graphically recursive simultaneous task allocation, planning, and scheduling," *The International Journal of Robotics Research*, vol. 41, no. 2, pp. 232–256, 2022.

[37] A. Fang and H. Kress-Gazit, "Automated task updates of temporal logic specifications for heterogeneous robots," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 4363–4369.

[38] A. Ham and M.-J. Park, "Human–robot task allocation and scheduling: Boeing 777 case study," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1256–1263, 2021.

[39] Z. Chen, J. Alonso-Mora, X. Bai, D. D. Harabor, and P. J. Stuckey, "Integrated task assignment and path planning for capacitated multi-agent pickup and delivery," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5816–5823, 2021.

[40] H. Chakraa, F. Guérin, E. Leclercq, and D. Lefebvre, "Optimization techniques for multi-robot task allocation problems: Review on the state-of-the-art," *Robotics and Autonomous Systems*, p. 104492, 2023.

[41] X. Zhou, H. Wang, B. Ding, T. Hu, and S. Shang, "Balanced connected task allocations for multi-robot systems: An exact flow-based integer program and an approximate tree-based genetic algorithm," *Expert Systems with Applications*, vol. 116, pp. 10–20, 2019.

[42] J. G. Martin, J. R. D. Frejo, R. A. García, and E. F. Camacho, "Multi-robot task allocation problem with multiple nonlinear criteria using branch and bound and genetic algorithms," *Intelligent Service Robotics*, vol. 14, no. 5, pp. 707–727, 2021.

[43] R. J. Alitappeh and K. Jeddisaravi, "Multi-robot exploration in task allocation problem," *Applied Intelligence*, vol. 52, no. 2, pp. 2189–2211, 2022.

[44] J. G. Martin, F. J. Muros, J. M. Maestre, and E. F. Camacho, "Multi-robot task allocation clustering based on game theory," *Robotics and Autonomous Systems*, vol. 161, p. 104314, 2023.

[45] S. Saeedvand, H. S. Aghdasi, and J. Baltes, "Robust multi-objective multi-humanoid robots task allocation based on novel hybrid metaheuristic algorithm," *Applied Intelligence*, vol. 49, no. 12, pp. 4097–4127, 2019.

[46] F. Janati, F. Abdollahi, S. S. Ghidary, M. Jannatifar, J. Baltes, and S. Sadeghnejad, "Multi-robot task allocation using clustering method," in *Robot Intelligence Technology and Applications 4: Results from the 4th International Conference on Robot Intelligence Technology and Applications*. Springer, 2017, pp. 233–247.

[47] Z. Junwei and Z. Jianjun, "Study on multi-UAV task clustering and task planning in cooperative reconnaissance," in *2014 Sixth International Conference on Intelligent Human-Machine Systems and Cybernetics*, vol. 2. IEEE, 2014, pp. 392–395.

[48] G. Vázquez, "Scheduling of missions with constrained tasks for heterogeneous multi-robot systems," Ph.D. dissertation, University of York, 2024.

[49] R. Calinescu, S. Gerasimou, and A. Banks, "Self-adaptive software with decentralised control loops," in *Fundamental Approaches to Software Engineering*, A. Egyed and I. Schaefer, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 235–251.

[50] J. Cámara, B. Schmerl, and D. Garlan, "Software architecture and task plan co-adaptation for mobile service robots," in *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 125–136.

[51] T. Pan, A. M. Wells, R. Shome, and L. E. Kavraki, "Failure is an option: task and motion planning with failing executions," in *International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 1947–1953.

[52] K. Ye, F. Yan, and S. Gerasimou, "Quantitative assurance and synthesis of controllers from activity diagrams," *arXiv preprint arXiv:2403.00169*, 2024.

[53] S. Alirezazadeh and L. A. Alexandre, "Dynamic task scheduling for human-robot collaboration," *IEEE Robotics and Automation Letters*, vol. 7, no. 4, pp. 8699–8704, 2022.

[54] J. Huang, C. Erdogan, Y. Zhang, B. Moore, Q. Luo, A. Sundaresan, and G. Rosu, "ROSRV: Runtime verification for robots," in *Runtime Verification*, B. Bonakdarpour and S. A. Smolka, Eds. Cham: Springer International Publishing, 2014, pp. 247–254.

[55] A. Ferrando, L. A. Dennis, D. Ancona, M. Fisher, and V. Mascardi, "Verifying and validating autonomous systems: Towards an integrated approach," in *Runtime Verification*, C. Colombo and M. Leucker, Eds. Cham: Springer International Publishing, 2018, pp. 263–281.

[56] S. A. Zudaire, L. Nahabedian, and S. Uchitel, "Assured mission adaptation of UAVs," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 16, no. 3-4, pp. 1–27, 2022.

[57] N. Li, S. Adepu, E. Kang, and D. Garlan, "Explanations for human-on-the-loop: A probabilistic model checking approach," in *Proceedings of the IEEE/ACM 15th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2020, pp. 181–187.

[58] J. Cleland-Huang, A. Agrawal, M. Vierhauser, M. Murphy, and M. Prieto, "Extending mape-k to support human-machine teaming," in *Proceedings of the 17th Symposium on Software Engineering for Adaptive and Self-Managing Systems*, 2022, pp. 120–131.

[59] A. Micheli, A. Bit-Monnot, G. Röger, E. Scala, A. Valentini, L. Framba, A. Rovetta, A. Trapasso, L. Bonassi, A. E. Gerevini *et al.*, "Unified planning: Modeling, manipulating and solving ai planning problems in python," *SoftwareX*, vol. 29, p. 102012, 2025.

[60] I. Predoaia, J. Harbin, S. Gerasimou, C. Vasiliou, D. Kolovos, and A. García-Domínguez, "Tree-based versus hybrid graphical-textual model editors: An empirical study of testing specifications," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, 2024, pp. 80–91.