EPTCS 391

Proceedings of the

Third Workshop on

Agents and Robots for reliable Engineered Autonomy

Krakow, Poland, 1st October 2023

Edited by: Angelo Ferrando and Rafael Cardoso

Published: 30th September 2023
DOI: 10.4204/EPTCS.391
ISSN: 2075-2180
Open Publishing Association

Table of Contents

Preface

This volume encompasses the proceedings of the Third Workshop on Agents and Robots for reliable Engineered Autonomy (AREA 2023), co-located with the 26th European Conference on Artificial Intelligence (ECAI 2023).

The realm of autonomous agents, extensively studied for decades, has been a focal point from both design and implementation standpoints. Nevertheless, the practical utilization of agents in real-world scenarios has predominantly been within software-centric applications, with limited adoption in situations necessitating physical interactions. Concurrently, the utility of robots has transcended narrow industrial contexts and has expanded across various domains. These domains span from robotic assistants to search and rescue operations, wherein the operational context is dynamic and not fully specified. This context often involves intricate interactions between multiple robots and humans.

These circumstances pose notable challenges to conventional software engineering methods. Enhanced autonomy stands as a pivotal avenue for enabling effective functioning of robotic applications in such settings. Autonomous agents and multi-agent systems emerge as promising methodologies for their development. As the levels of autonomy and interaction escalate, ensuring reliable behavior becomes increasingly intricate, not only in robotic applications but also in conventional autonomous agent scenarios. Hence, there exists a demand for research into novel verification and validation approaches that can be seamlessly integrated into the developmental life cycle of these systems.

The primary objective of this workshop is to facilitate collaboration between researchers in the fields of autonomous agents and robotics. By amalgamating insights from these domains, innovative solutions could potentially emerge to tackle intricate challenges associated with the verification and validation of autonomous robotic systems.

In this third iteration of the workshop, a total of 12 submissions were received, of which 7 full papers and 5 short papers were accepted. We extend our gratitude to all authors who contributed their valuable work to the workshop.

Finally, we would like to thank our invited speakers, Viviana Mascardi and Bruno Lacerda. The title and abstract of their presentations can be found below.

We also express our appreciation to the 25 program committee members (the complete list is available below) for their valuable feedback, which contributed to the refinement of the papers. Our thanks also extend to the EPTCS staff for their support in compiling these proceedings.

For additional details about the workshop, kindly refer to our website: `https://areaworkshop.github.io/AREA2023/`.

The AREA 2023 organisers,
Angelo Ferrando and Rafael C. Cardoso

Program Committee

- Tobias Ahlbrecht, Clausthal University of Technology (Germany)
- Gleifer Vaz Alves, Federal University of Technology – Paraná (Brazil)

A. Ferrando, R.C. Cardoso (Eds.): Third Workshop on
Agents and Robots for reliable Engineered Autonomy 2023 (AREA'23)
EPTCS 391, 2023, pp. iii–iv, doi:10.4204/EPTCS.391.0

Talking Agents in the Virtual World

Viviana Mascardi
University of Genoa, Italy
viviana.mascardi@unige.it

What do chatbots and the metaverse have to do with reliability, cognitive agents, and robotic applications? In this talk I will explore the intriguing connections among them, presenting a small scale prototype of (reliable) "talking agents in the virtual world".

A. Ferrando, R.C. Cardoso (Eds.): Third Workshop on
Agents and Robots for reliable Engineered Autonomy 2023 (AREA'23)
EPTCS 391, 2023, pp. 1–1, doi:10.4204/EPTCS.391.1

Model-Based Reasoning under Uncertainty for Reliable Robot Mission Planning

Bruno Lacerda
University of Oxford, United Kingdom
bruno@robots.ox.ac.uk

In this presentation, I will argue that the synergy between three factors is critical for creating reliable mission planning algorithms for autonomous robots operating in uncontrolled environments. These factors are (i) utilising historical data gathered online to enhance decision-making under uncertainty models, (ii) implementing principled planning techniques that explicitly reason about the epistemic uncertainty inherent to these data-driven models, and (iii) incorporating rich specifications that go beyond typical expected reward maximisation problems. Developing frameworks that unify these three factors is an open problem in the field of robotics, which is heavily dependent on the specific application domain. I will offer an overview of various works undertaken at the GOALS lab at the Oxford Robotics Institute that consider these three factors in distinct ways. Moreover, I will describe how these works provide a foundation for creating integrated approaches that enable long-term deployment of robots capable of acquiring parametrised environmental models from historical data, plan considering the epistemic uncertainty of such models, and effectively adapt their behaviour to in-mission observations, continuously refining their estimate over the epistemic uncertainty online.

A. Ferrando, R.C. Cardoso (Eds.): Third Workshop on
Agents and Robots for reliable Engineered Autonomy 2023 (AREA'23)
EPTCS 391, 2023, pp. 2–2, doi:10.4204/EPTCS.391.2

Online Proactive Multi-Task Assignment with Resource Availability Anticipation

Déborah Conforto Nedelmann Jérôme Lacan Caroline Chanel
ISAE-SUPAERO, Université de Toulouse, France
{deborah.conforto-nedelmann,jerome.lacan,caroline.chanel}@isae-supaero.fr

With the emergence of services and online applications as taxi dispatching, crowdsourcing, package or food delivery, industrials and researchers are paying attention to the online multi-task assignment optimization field to quickly and efficiently met demands. In this context, this paper is interested in the multi-task assignment problem where multiple requests (e.g. tasks) arrive over time and must be dynamically matched to (mobile) agents. This optimization problem is known to be NP-hard. In order to treat this problem with a proactive mindset, we propose to use a receding-horizon approach to determine which resources (e.g. taxis, mobile agents, drones, robots) would be available within this (possibly dynamic) receding-horizon to meet the current set of requests (i.e. tasks) as good as possible. Contrarily to several works in this domain, we have chosen to make no assumption concerning future locations of requests. To achieve fast optimized online solutions in terms of costs and amount of allocated tasks, we have designed a genetic algorithm based on a fitness function integrating the traveled distance and the age of the requests. We compared our proactive multi-task assignment with resource availability anticipation approach with a classical reactive approach. The results obtained in two benchmark problems, one synthetic and another based on real data, show that our resource availability anticipation method can achieve better results in terms of costs (e.g. traveled distance) and amount of allocated tasks than reactive approaches while decreasing resources idle time.

1 Introduction

The emergence of new applications dedicated to services such as taxi dispatching [9], ridesharing [12], crowdsourcing [27] or package delivery has generated a lot of interest in the field of online multi-task assignment. What all these services have in common, is that users can make requests and the platforms have to adapt their resources in order to satisfy the demand.

The difference between offline and online assignment is that on the online one, requests (i.e. tasks) arrive over time and are dynamically matched whereas, in the offline case, all the requests are known beforehand [9]. With offline matching, finding an optimal task assignment is doable whereas online matching poses additional challenges. In a majority of online task assignments, resources are disposable and can only be used once, whereas we will consider the case where the resources are reusable in accordance with the following references [25], [9]. In other words, after an agent has received an assignment, it will not be available for a new allocation for a certain period of time before being able to be assigned to new requests.

In order to treat online (multi-)task assignment, there are generally two options to manage the time. The first option consists to divide the time into intervals. At the end of each interval, the requests that have arrived within this interval are assigned to agents that have finished their previous tasks [2], [19]. The second option consists in assigning the requests immediately upon their arrival (i.e. continuously on time) to whatever available agents at that moment. Both of these solutions have drawbacks: the first one might result in a better solution but the waiting time before agents are available to assign tasks might be important and the second one assigns the request immediately but the solution is worse [25].

A. Ferrando, R.C. Cardoso (Eds.): Third Workshop on
Agents and Robots for reliable Engineered Autonomy 2023 (AREA'23)
EPTCS 391, 2023, pp. 3–17, doi:10.4204/EPTCS.391.3

Following the literature, there are several possibilities to describe the requests according to their arrival model [20] but, in this work, we have chosen to make no assumption on any arrival distribution. This choice was motivated by the fact that we chose to focus on the performance of online task allocation algorithms without being distracted by any knowledge of future task appearances.

With a proactive mindset, our contribution to treat this problem is to use a receding-horizon approach to determine which agents would be available at a given time horizon given their current allocated tasks. Inspired by [2] and [19], our approach divides the time in small intervals and accounts for a certain horizon size (some time steps in the future), to determine where the agents would be and if they would have finished their tasks within this horizon. This would lead to the choice to consider them as near-future available resources for current multi-task assignment problem. Hence, our contribution can be seen as a proactive multi-task assignment with resource availability anticipation approach.

In terms of assignment computations, this problem is similar to the Multiple Traveling Salesmen Problem with open path and multiple depot [22]. A way to treat this problem online is to use meta-heuristic approaches, such as Genetic Algorithm, as we will present in next section. Moreover, meta-heuristic approaches allows to use multi-objective criteria costs to evaluate (multi-)task assignments solutions [23].

We compared our proactive approach with a classical reactive approach (no anticipation) using two benchmark problems: a synthetic and a real-data based one. In brief, the results show that our resource availability anticipation method can achieve better results in terms of costs (e.g. traveled distance) and amount of allocated tasks than a reactive approach. Interestingly, the results also demonstrate that our approach decreases resources idle time.

The paper is organized as follows. Section 2 overviews related works. Section 3 formally describes the optimization problem we are addressing. The approach proposed to treat the proactive online multi-task assignment problem is presented in Section 4. Experiments are showed in Section 5, and Section 6 concludes the paper.

2 Related Work

The general problem we address in this paper is the multi-task assignment with reusable resources, where the goal is to coordinate a set of agents (i.e. the resources) in order to accomplish some tasks in an efficient way [17] within a given time period. Here, we focus on the case where multiple agents have to travel, reach and perform their tasks in a way that the overall (traveled) distance is minimized and the number of assigned tasks is maximized. This is reminiscent of the Multiple Traveling Salesmen Problem (MTSP), a well known hard optimization problem, for which literature has proposed several solution search algorithms [22][5]. In particular, our problem is similar to the special case of MTSP with open path and multiple depots. For this last, most common methods use meta-heuristics which include Ant Colony Optimization, Simulated Annealing and Genetic Algorithm [5][22].

In this paper we are focusing on online variants of the classical multi-task assignment approaches [17], [16], [6], where requests arrive dynamically. Compared to classical offline models, online models tend to perform worse due to the uncertain nature of the future requests [3]. In spite of this, several methods have been used to find solutions for the online assignment problem. A common one is the Linear Programming [25] [21]. Evolutionary algorithm have also been used due to their strong performance for multi-criteria objective [23]. Given the nature of the problem we are treating in this paper, this solving approach interests us as we will detail on Section 4. Reference [12] has studied, in the context of online assignment, the performance of variants of the Ant Colony Optimization (ACO) algorithm compared to a Genetic Algorithm (GA) and have found that a variant of ACO would perform at best the same as the GA. This is why GA seems to be an interesting choice.

Additionally, we have observed in the literature (see [1] [10] [9] [25] [14]) that papers tend to focus on using a request arrival model, usually a distribution that describes the likely tasks locations (e.g. requests). Several authors have examined the concept of anticipation by working with the distribution. In [4], the authors divides the time in intervals and with the arrival distribution determines the number of tasks and their location a couple of intervals ahead, enabling them to better position their agents to respond to the future tasks more effectively. In [8], the authors have beforehand an example of the characteristics of the requests which allows them to build offline a distribution about the more likely scenarios. On a more practical example, both [24] and [7] chose to build an anticipation model to try to avoid traffic jams thanks to previous registered data. As said previously, we make no assumption about the requests arrival model, but we work on anticipating the availability of agents given their current tasks (i.e. the end-time of the tasks assigned to a given agent and its corresponding location). As far as we know, there is no work in the literature for online proactive multi-task assignment with resource availability anticipation.

Moreover, to treat the problem online, time has to either be divided in small intervals or requests must be assigned immediately at their arrival. Both have their benefits and drawbacks, because as presented in [26], either the new allocation is sub-optimal (regarding a long-term horizon), due to a lack of available agents at a given moment compared to the fleet, or that, in order to find a better solution, we need to wait for more agents to become available, which can be long. In this work, we have decided to divide the time into intervals and assign the requests that have arrived within the interval to agents that may be available within a given receding-horizon.

3 Problem Statement and Treatment

3.1 Problem statement

Multi-task assignment is a combinatorial optimization problem. In our case and over the entire time horizon, we aim to maximize the amount of allocated requests (i.e. tasks) while also minimizing the overall traveled distance of the agents (i.e. the resources).

We consider that we have a set of M agents denoted by $A = \{a_1, a_2, \ldots, a_M\}$. The location of the agents is denoted by $\mathbf{p_a}$ for all $a \in A$. The tasks are described by their location $\mathbf{p_r}$ and the expected time t_r an agent is supposed to met them. Thus, a single request $r \in R$ can be written as $r = (\mathbf{p_r}, t_r)$. A request can be assigned to only one agent. Thus, each agent $a \in A$ has an associated vector of requests assigned to them: $R_a = (r_a^1, r_a^2, \ldots, r_a^n)$, where n is the number of tasks assigned to a at a given time. To describe if a request has been allocated to an agent, we use the binary variable $x_{a,r}$. If request r was allocated to agent a, we have $x_{a,r} = 1$, otherwise $x_{a,r} = 0$. Thus, we have $R_a = \{\bigcup_{r \in R} r | x_{a,r} = 1\}$ defining the set of tasks from R assigned to a. To an agent a maximum of C_a requests can be assigned. Let L_a be the cost associated to the path length that an agent have to travel to accomplish their assigned tasks at a given time.

The total time horizon T is divided into small intervals called time steps or time windows. The time windows are indexed by τ and their duration is a constant equal to δ. During the time window τ, we read the buffer of requests B and store the new requests in R_τ. It is worth saying that R_a, C_a and L_a may also depend on this time step τ, such as $R_{\tau,a}$, $C_{\tau,a}$ and $L_{\tau,a}$.

Therefore, the general optimization problem we address can be formalized as:

$$\min \left[\sum_{\tau=0}^{T-1} \Big(\alpha \sum_{a \in A} L_{\tau,a} \big(\bigcup_{r_i \in R_\tau} r_i | x_{a,r_i} = 1 \big) + (1-\alpha) \big(|R_\tau| - \sum_{a \in A} \sum_{r \in R_\tau} x_{a,r} \big) \Big) \right] \tag{1}$$

subject to:

$$\sum_{a \in A} x_{a,r} \leq 1, \forall r \in R_\tau, \tag{2}$$

$$\sum_{r \in R_\tau} x_{a,r} \leq C_{\tau,a}, \forall a, \forall \tau \in \{0, ..., T-1\} \tag{3}$$

where the weights $\alpha (\in [0,1])$ and $(1-\alpha)$ define the relative importance of the total path length cost and the total number of assigned tasks, including criteria scaling needs. We do not have any information on the upcoming requests, thus the general optimization problem formalized above is hard (or even impossible) to be solved offline.

3.2 Problem treatment

With a proactive mindset, we propose to use a receding-horizon approach to determine which agent would be available at horizon H given their current allocation. For simplicity, we define this receding-horizon as a multiple of the duration of the time intervals such as, $H(k) = k\delta$, with $k \geq 0$ (e.g $H(5) = 5\delta$).We will call the set of agents that are available within horizon H at time step τ as $A_\tau(H) \in A$. Note that the number of available agents depends on the size of the receding-horizon H and the time step τ.

So to achieve a solution to the general optimization problem presented, we use this receding-horizon to be proactive, and we adapt the optimization problem to be solved at each time window τ. At τ, we know the position of the tasks R_τ and we can check the availability of the agents within this receding-horizon H.

Thus, at the time step τ, the optimization problem we solve is the following:

$$\min \left[\alpha \sum_{a \in A_\tau(H)} L_{\tau,a}\Big(\bigcup_{r_i \in R_\tau} r_i | x_{a,r_i} = 1 \Big) + (1-\alpha)\Big(|R_\tau| - \sum_{a \in A_\tau(H)} \sum_{r \in R_\tau} x_{a,r} \Big) \right] \tag{4}$$

subject to:

$$\sum_{a \in A_\tau(H)} x_{a,r} \leq 1, \forall r \in R_\tau \tag{5}$$

$$\sum_{r \in R_\tau} x_{a,r} \leq C_{\tau,a}, \forall a \in A_\tau(H) \tag{6}$$

In the following, we define how we compute $L_{\tau,a}$, the cost associated to the path length that an agent a has to travel to accomplish its assigned tasks at τ:

$$L_{\tau,a}(\bigcup_{r_i}) = c(\mathbf{p_a}; \mathbf{p_{r_1}}) + \sum_{i=1}^{|\cup_{r_i}|-1} c(\mathbf{p_{r_i}}; \mathbf{p_{r_{i+1}}}), \forall r_i \in R_\tau \tag{7}$$

where c is the cost associated to the distance between two locations and $\bigcup_{r_i} = \{\bigcup_{r_i \in R_\tau} r_i | x_{a,r_i} = 1\}$. This path cost formula was inspired by the definition of cost for MTSP with multi-depot open path proposed in [5] and adapted to our case. $L_{\tau,a}(\bigcup_{r_i})$ effectively quantifies the length of the shortest path between the agent's location and his assigned tasks.

As a feature, and with the aim of maximizing the number of achieved requests within the entire horizon T, at τ, we consider the tasks that have not been allocated in earlier time steps in addition to the current buffer B to form R_τ. Earlier requests are considered with higher priority compared to later tasks. This constraint will be integrated into the design of the solution, explained in Section 4.3.2. The reason we take into account this aspect is that we want to avoid cases where some tasks would potentially wait for a long time before being allocated or even never be allocated.

4 Online Proactive Task-Assignment with Resource Availability Anticipation

4.1 General solving procedure

In accordance with our problem statement and model stated previously, we summarize our general algorithm as the pseudo-code in Algorithm 1. In line 1 and 2, we first define the set of agents and their locations as well as the size of our receding-horizon H. At each time step τ, we consider the tasks that have arrived since the last assignment which are stored in buffer B along with the ones that have not been assigned yet (line 5). Then in line 6, we check the availability of agents at the receding-horizon H. Agents that are available within H are then used for multi-task assignment using a genetic algorithm in line 7 (explained later).

Algorithm 1 General Algorithm

1: Definition of the set of agents A and their locations
2: Definition of horizon H
3: $R_{\tau=0} \leftarrow \emptyset, t_r = 0$
4: **for** each time step τ **do**
5: $R_\tau \leftarrow \text{GetTasksFrom}(B, R_{\tau-1})$
6: $A_\tau(H) = \text{AvailabilityAnticipation}(H, A, \tau)$
7: $\text{Sol}_\tau = \text{GeneticAlgorithm}(R_\tau, A_\tau(H), C_{\tau,a})$

Algorithm 2 Availability Anticipation Process

1: **procedure** AVAILABILITYANTICIPATION(H,A,τ)
2: $A_\tau(H) \leftarrow []$
3: **for** $a \in A$ **do**
4: $R_{\tau,a} \leftarrow \bigcup_{r_i \in R_\tau} r_i | x_{a,r_i} = 1$
5: $t_{R_{\tau,a}} \leftarrow \text{getLastTaskElementTime}(R_{\tau,a})$.
6: **if** $t_{R_{\tau,a}} < \tau + H$ **then**
7: append a to $A_\tau(H)$
8: return $A_\tau(H)$.
9: **end procedure**

4.2 Availability Anticipation

We now define the essential part of our contribution which corresponds to the function on line 6 in Algorithm 1. The main difference with other reactive matching algorithms [25] [14] [9] is we are not using only the agents that are strictly available at τ, but we are computing which agents will be available within a certain horizon.

Using only the instantly available agents at the current time may provide sub-optimal solutions in terms of distance and the overall amount of requests allocated [26]. Note if an agent becomes available right after the time window, it will have to wait almost the time window duration before being assigned to new requests. We believe anticipation may allow to find better solutions in terms of overall traveled distance while being proactive in terms of resource usage. Additionally, in terms of the number of requests completed, if we have more agents available, the number of tasks that can be allocated is automatically higher given an agent can only treat a maximum of $C_{\tau,a}$ requests per round. Another point is we do not wait for the agents to complete their previous tasks before assigning them new ones. With this, we avoid the problem of agents staying idle for almost a round.

A feature of our solution is to try to avoid unbalance of tasks between agents where the requests of busy agents could be completed well after some newer requests, which are assigned to less busy agents. With our solution, if some agents are very busy, they will simply not be picked for a new allocation until they complete some of their previously assigned requests.

The availability anticipation process is detailed in Algorithm 2. For every agent, we check the estimated time $t_{R_{\tau,a}}$ when their last request is supposed to be completed in lines 4 and 5. If this time is within the horizon H with respect to current time τ (line 6), then we consider this agent as available at the receding-horizon and consequently this agent is used for multi-task assignment. The last request completion time $t_{R_{\tau,a}}$ can be predicted by supposing each agent has a constant velocity. Using the current

set of tasks assigned to an agent, $R_{\tau,a}$ we can calculate the time necessary for the agent a to go to them and, also its final location.

4.3 Genetic Algorithm

The Genetic Algorithm (GA) is an evolutionary algorithm and a well known meta-heuristic approach [15] usually used to solve high combinatorial optimization problems [5], which iteratively constructs solutions. A proposition of a solution by a GA is called a chromosome and a list of chromosomes is a population. At each iteration, the population is evaluated and the better chromosomes are used to create a new generation thanks to some operations inspired by natural evolution theories. The new generation will hopefully provide better solutions than the previous one. This process is repeated until some stopping

Algorithm 3 Genetic Algorithm

procedure GENETICALGORITHM($R_\tau, A_\tau(H), C_{\tau,a}$)
  Definition of the probability of mutation p_{muta} and the probability of swapping p_{swap}
  Definition of the size of the population n_{Pop} and the size of a chromosome: $|A_\tau(H)| \times C_{\tau,a}$
  Definition of the timing beginning t_{beg}
  Fill each chromosome *chr* of the population Pop with -1
  $P \leftarrow []$
  for $r_i \in R_\tau$ **do**
    Calculate p_i with the Boltzmann Probability Distribution
    Append p_i to P
  for each $chr \in Pop$ **do**
    while $R_\tau \neq \emptyset$ **or** $count(v \text{ in } chr \mid v = -1) \neq 0$ **do**
      Select r_i from R_τ using P
      Select *pos* position in *chr* randomly
      $chr[pos] = r_{i_{ID}}$
      Remove r_i from R_τ
  Set $min_{now} = \infty$ and $min_{prec} = 0$
  Definition of the current time t_{now}
  while $t_{now} - t_{beg} < \delta$ or $|min_{now} - min_{prec}| > \varepsilon$ **do**
    $S \leftarrow []$
    for each *chr* in Pop **do**
      $s = fitness(chr)$
      Append s to S
    Set $min_{prec} = min_{now}$ and $min_{now} = min(S)$
    Select the best 30% chromosomes $Best_{chr} \subset Pop$ regarding S
    $Pop \leftarrow []$
    Append $Best_{chr}$ to Pop
    while $|Pop| < n_{Pop}$ **do**
      $parent1 = random(Best_{chr})$ and $parent2 = random(Best_{chr})$
      $child_{chr} = Crossover(parent1, parent2)$
      $v = random(0,1)$
      if $v < p_{muta}$ and $v < p_{swap}$ **then**
        $child_{chr} = Swapping(child_{chr})$
      if $v < p_{muta}$ and $v > p_{swap}$ **then**
        $child_{chr} = Inversion(child_{chr})$
      Append $child_{chr}$ to Pop
    Update the current time t_{now}
  Set $\text{Sol}_\tau = chr^*$, such as $chr^* = argmin_{chr \in Pop} S$
  return Sol_τ
end procedure

condition is reached.

In the following, we detail some design choices of the GA that we use in order to find assignment solutions to the R_τ tasks considering the $A_\tau(H)$ agents. This operation corresponds to line 7 of Algorithm 1 and is detailed in Algorithm 3.

4.3.1 Solution representation

Our population has the general aspect illustrated in Fig. 1. Each chromosome represents a proposition of solution (i.e. an multi-task assignment solution given the agents and requests considered). The size of a chromosome is therefore limited to $|A_\tau(H)| \times C_{\tau,a}$ to respect the maximum number of requests assigned to each agent. If this constraint is relaxed, the size is then $|A_\tau(H)| \times |R_\tau|$. Each request has a unique positive identification number (ID) which is used inside the chromosome. If an agent is assigned less than $C_{\tau,a}$ tasks, the remaining values are noted as -1 to differentiate from the ID. For example, in the last chromosome of Figure 1, the GA has assigned requests 8 and 6 to the first agent, requests 12 and 3 to the second agent, request 5 to the third agent, and request 7 to the last agent.

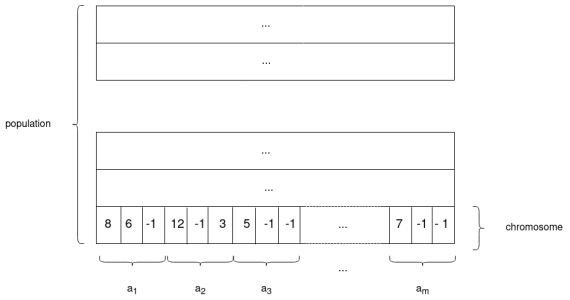


Figure 1: Structure of the population and chromosome.

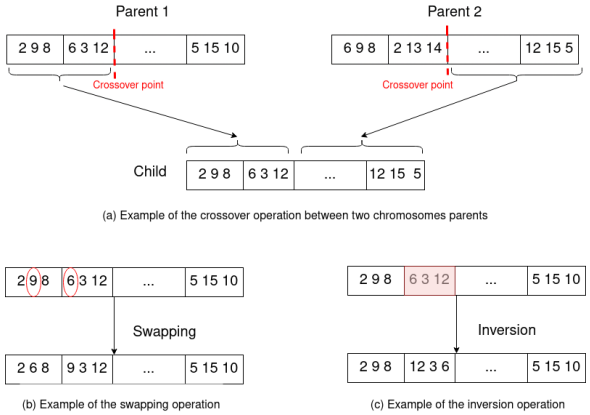


Figure 2: Illustration of the different evolution operations: (a) for crossover and (b) and (c) for the respective swapping and inversion mutation operations.

4.3.2 Solution Initialization

This step corresponds from line 5 to 22 of Alg. 3. In line 5, we fill each chromosome with the value -1 that we will possibly replace with the ID of the requests. As mentioned previously, we want to prioritize the selection of tasks that have been registered earlier. For this, we use the Boltzmann Probability Distribution, which was proposed for thermodynamic field but has since been picked up to be used in the domain of reinforcement learning [11]. We propose the following adaptation:

$$p_i = \begin{cases} \frac{1}{Q}\exp(-\frac{t_{r_i}}{\tau}) & \text{for } \tau > 0 \\ \frac{1}{|R_\tau|} & \text{for } \tau = 0 \end{cases} \tag{8}$$

where $Q = \sum_{r_j=1}^{|R_\tau|} \exp(-\frac{t_{r_j}}{\tau})$. Then, p_i is the probability of the task r_i to be selected, and t_{r_i} is the time when r_i was registered and τ is the current time step. This is done between the lines 7 and 9 of Alg. 3. We select the tasks using this distribution for each chromosome initialization and we order them in the chromosome randomly (from line 10 to line 15).

To create a new generation, we need to evaluate the quality of our previous population chromosomes thanks to a fitness function defined from our optimization objective. The goal is to minimize the result of this fitness function throughout the generations of the GA. The fitness function is defined as follows:

$$\alpha\Big(\sum_{a\in A_\tau(H)} l_a \frac{1}{L_{max}}\Big) + (1-\alpha)\Big(1 - \frac{\sum_{a\in A_\tau(H)}\sum_{r\in R_\tau} x_{a,r}}{|R_\tau|}\Big) \tag{9}$$

where l_a is the distance traveled by agent a from its (current or anticipated) position in order to complete his new list of requests, and r_n is the list of requests assigned to agent a for this chromosome. We normalize the cost concerning the total traveled distance agents $A_\tau(H)$ by L_{max}, which is the maximum distance traveled by all agents in the first generation of the GA. As we assign the selected tasks in a random order in the solution initialization, this maximum distance obtained in the first generation can be considered as a worst-case solution. The scoring of the different chromosomes is done between lines 20 and 22 in Alg. 3.

4.3.3 Evolution operations

Using the fitness function, we select the 30% best chromosomes (line 24) and apply some operations on them to build a new generation. We use crossover operation and two types of mutation operations. These different operations are illustrated in Figure 2.

The crossover operation selects two parent chromosomes and a random index. The elements from the first parent are copied until that index and from that index, we append the elements of the second parent, respecting the condition that a request can only be assigned to one agent. This creates a child chromosome (see line 29 of Alg. 3). After the crossover operation, some chromosomes go randomly through one of the mutations. This operation is usually used to avoid the genetic algorithm getting stuck on a local minimum (see [18]). We call the first type of mutation swapping, which consists in taking two random tasks in a chromosome and swapping their positions. The other mutation is the inversion operation, where the order of the requests between two random indexes is inverted. A chromosome can (randomly) undergo a mutation following two probabilities p_{muta} and p_{swap} (line 30 to 34) which were empirically set.

4.3.4 Stopping condition

Our GA stops if it reaches one of two stopping conditions: (i) if it can not improve its solution value (e.g. not more than $\varepsilon \simeq 0$) from the last couple of generations or (ii) if the time spent since the beginning of the GA calculations has reached the duration of the time window δ (line 18 in Alg. 3). If some requests were not assigned (e.g. due to the constraint on the number of tasks), they will be considered along with the buffer for the next allocation. The best solution in terms of fitness function evaluation is selected and we consider it as the assignment solution (Sol_τ) for the time window τ (line 37).

5 Experiments

5.1 Benchmarks and Metrics

The first aspect we are interested in is how our proactive approach compares with a reactive approach that considers only the immediately available agents. For that, we are going to use two different benchmarks: the first one is a synthetic simulation and the second one is based on the real data of taxi dispatching in New York city[1] obtained from reference [9].

[1] available at `http://www.andresmh.com/nyctaxitrips/`

This is also the step where we have to make some adaptations to fit each scenario. We focused on the cost of the distance l_a present. In our synthetic scenario, we consider the request as completed when the agent reaches it; so l_a is only the distance from the anticipated position of the agent to the first task and the distance between the other tasks. Whereas for taxi dispatching, we have to also take into account the pick-up and the arrival locations of the requests, so here l_a is sum of the distance from the anticipated position of the taxi to the pick-up of the first request, the distance between the pick-up and the arrival for each task and the distance between the arrival location of the previous task and the pick-up of the following request. We can adapt this criterion to whatever scenario by just changing the formula of l_a.

During our synthetic simulation, we vary different parameters to analyze how they impact the performance of our proposal. We will first analyze the impact of changing α through empirical testing. Note that different values of α give more or less importance to either the traveled distance criterion or the percentage of assigned requests criterion. The goal is to find an α value that allows for a compromise between the performance of the different criteria. We will also evaluate how our method performs depending on the number of tasks compared to the number of agents: in one case, at each time window τ there will be fewer tasks to assign than the total number of agents $|R_\tau| < |A|, \forall \tau \in \{0, ..., T-1\}$, and on the other hand, there will be more tasks than agents at each time window $|R_\tau| > |A|, \forall \tau \in \{0, ..., T-1\}$.

In a second moment, we analyze the impact of the receding-horizon H. We denote by $H(k)$ the receding-horizon with a duration equal to k time steps. In variable case, denoted $H(v)$, we compute the solutions for several receding-horizon sizes and take the best solution among them. Note that the $H(0)$ case is equivalent to the classical reactive approach whereas the others constitute the proactive approach. In the variable receding-horizon case $H(v)$ we search a solution for $H(0)$, $H(1), \ldots, H(5)$ and chose the best solution among them regarding the score from the fitness function in the GA. The last parameter we evaluate is the impact of the maximum amount of requests $C_{\tau,a}$ that can be assigned to an agent in each time window. For one case, we fixed $C_{\tau,a} = \frac{1}{3}|R_\tau|$ and for the other case $C_{\tau,a} = \infty$ for all different agents.

For the real data set of taxi dispatching, we want to see how our method scales and performs in a real-data based environment, where there is more ground to be covered by the numerous agents and the number of requests really varying according the time and the activity of the clients.

In terms of metrics we uses to evaluate our model, we are interested in three specific metrics registered during the total time length on our simulations. The first one is the total distance traveled by the agents, the second one is the time agents stay idle, and the last one is the percentage of allocated tasks. We want the distance and the idle time to be low and the percentage of assigned tasks to be high. The idle time is when agents stay idle waiting for their new assignments after completing their tasks. Normally, with the availability anticipation, this only happens to an agent if the timing to complete all their requests is less than τ.

5.2 Results

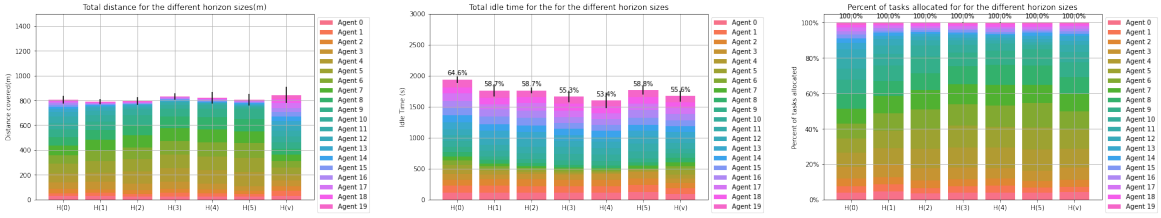
5.2.1 Synthetic benchmark

In this simulation, we have fixed the velocity of each agent at 1 m/s. The generated requests at τ have a random position in a squared grid world of dimensions of $10m \times 10m$. One simulation has a total of 30 time windows and we use 10 different simulations. During one simulation, each agent can travel a maximum of 150m. The results presented are based on the average of our 10 simulations.

Evaluation of the impact of the α parameter. We attributed to α the following values: {0, 0.25, 0.5, 0.75 ,1} for the fixed and variable receding-horizons. We also look at the case where there are fewer requests per time step than agents and the contrary. Results are presented in Table 1 in Appendix. We can observe that, in general, when the α value increases, the traveled distance decreases, which is an

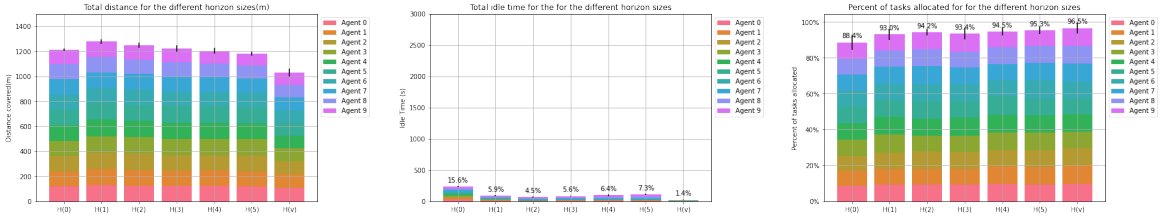
expected result since α is the weight of the distance in the fitness function. When $\alpha = 0$, no attention is given to the way the tasks are ordered and the algorithm does not search for the shortest path. This is also when the idle time is the lowest since the agents have the biggest distance to cover.

With $|R_\tau| > |A|$, when α value increases, the percentage of allocated tasks rises even if the weight of the total number or assigned tasks criterion is less important in the fitness function. With $|R_\tau| < |A|$, percentage of allocated tasks is almost 100 % until $\alpha = 0.75$, when the minimization of total distance becomes the dominant criterion and the percentage drops slightly. In this case, the the agents are underutilized and should be able to complete all the requests. For this reason, we chose to exclude $\alpha = 1$. The value with the best results is $\alpha = 0.75$. This value will used in the deep analysis presented in the following and on the rest of our synthetic benchmark tests.



(a) Total traveled distance (b) Total idle time (c) Percent of assigned tasks

Figure 3: Comparison between different receding-horizon sizes when $|R_\tau| < |A|$.

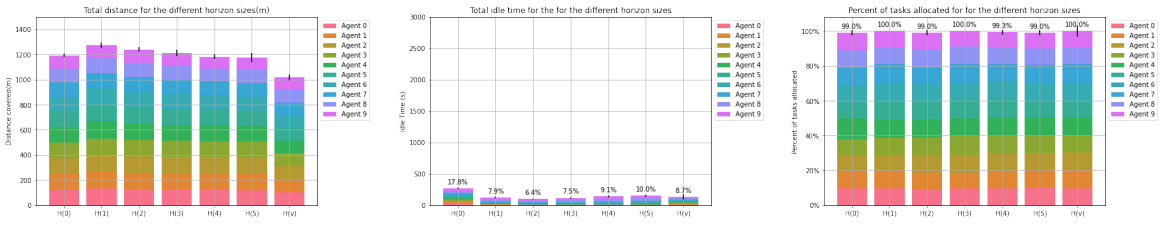


(a) Total traveled distance (b) Total idle time (c) Percent of assigned tasks

Figure 4: Comparison for different receding-horizon sizes when $|R_\tau| > |A|$.

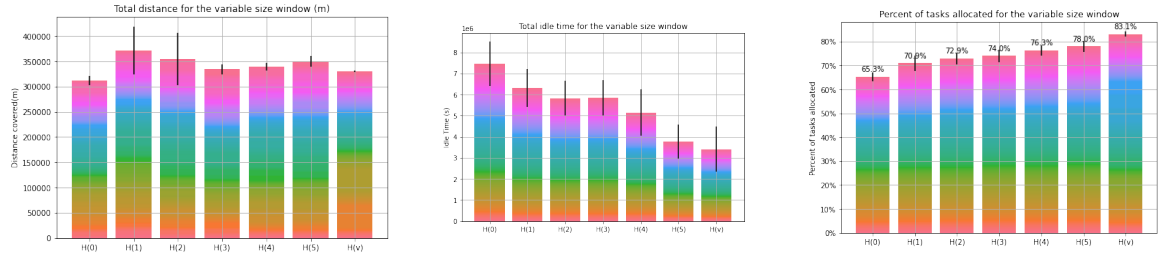
Less tasks than agents In Figure 3, we are paying attention to the case where at each time step the number of tasks is inferior to the number of agents. In particular, we have 20 agents and new 10 tasks at each time window τ. In terms of total distance traveled during the simulation, we can see that it is almost equal for the different receding-horizon sizes. For the total agents idle time, it is maximal when there is no anticipation (i.e. at $H(0)$) and decreases for the following values until it reaches a minimum in $H(4)$ before going slightly up again for $H(5)$. This decrease is explained by the fact that the availability anticipation allows to predict where and when the different agent finishes his tasks. The increase at the end can also be explained. The values of idle time are particularly high because the agents are underused. In terms of percentage tasks assigned, we can see that all of them have been assigned regardless of the size of the receding-horizon. In the case where there are less requests than agents, the anticipation availability method is not necessary as it does not bring advantages in terms of traveled distance and amount of allocated tasks (the two metrics we want to optimize) but can be interesting to lower the time agents stay idle.

More tasks than agents Figure 4 shows the results for the case where, at each round, the number of tasks is superior to the number of agents. Even in this case we maintain the constraint of $C_{\tau,a} = \frac{1}{3}|R_\tau|$. In general, we can see that with a farther receding-horizon, our method allows for a better balance. As previously, when there is no anticipation, the percentage of assigned tasks is the lowest and the agents idle time the highest. Here, with the constraint of maximum number of tasks per agent, we can see a significant improvement in terms of percentage of assigned tasks when anticipation is used. With these results, we can clearly see that the variable receding-horizon is the best among all the receding-horizon cases, as it achieves the lowest distance, the lowest idle time and the higher percentage assigned tasks. This is due to the fact we take the receding-horizon size that is more advantageous, resulting in the best results.



(a) Total traveled distance (b) Total idle time (c) Percent of assigned tasks

Figure 5: Comparison for different receding-horizon sizes when $C_{\tau,a} = \infty$.



(a) Total traveled distance (b) Total drivers idle time (c) Percent of assigned tasks

Figure 6: Our method applied to taxi dispatching dataset to answer the consumers' requests.

Relaxation of the $C_{\tau,a}$ constraint In this set of results we study the case where we relax the constraint $C_{\tau,a}$ such as $C_{\tau,a} = \infty$, with $|R_\tau| > |A|$ (we have 10 agents and 20 new tasks each time window τ). Compared to the previous part, the total traveled distance is higher but it is explained by the fact that the agents are traveling to more tasks and at the same time. It is confirmed by the fact that the idle time is lower since our agents (e.g. resources) are more used. As before, the percentage of assigned requests is close to 100%.

For the different receding-horizon cases, in terms of total distance traveled, we have an increase from $H(0)$ to $H(1)$ and then the value decreases. The increase can be explained by the fact that the idle time with in these cases more important - i.e. less anticipation cases. With the receding-horizon size growing, the traveled distance diminishes because we balance the assigned tasks in a better way. Additionally, in terms of idle time, there is an important difference when there is no anticipation compared to when there is some anticipation, which allows for a better use of the agents. With the anticipation, there is no need to wait for the agents to finish their previous tasks before assigning them new ones.

With these results we can speculate that our proactive online multi-assignment method which anticipates resource availability brings benefits in terms of resource use.

5.2.2 Taxi Dispatching benchmark

We also applied our method to a real-set of data to analyze how our approach scales up. For this we use the dataset, which details the rides of the taxis in New York on the year 2013 (see [9]). To evaluate our method on this dataset we considered the pickup-time instead as the time a request for a ride was registered. We concentrated our test from January 07th to January 09th and during nights from 12:00am to 07:00am. We set a time window lasting 5 minutes. To answer the requests, we considered a constant fleet of 1000 taxis, randomly placed in New York. We assume the taxis to have a constant velocity of 30 mph (the authorized limit in New York). Contrarily to the synthetic benchmark, here the number of requests truly varies as it depends on the habits of the clients. This way, the minimum amount of requests are registered between 3AM and 4AM (most people are sleeping) whereas the maximum is close to 7AM when people start going to work.

The results are showed in Figure 6. We can see that as in the synthetic benchmark simulations, the anticipation availability method allows for a higher percentage of assigned tasks, while also reducing the time the drivers stay idle. We also see that $H(v)$ is the setting with the more allocated tasks and one of the lowest distance, which highlights the potential of using the variable receding horizon. In a curious way, the distance does not steadily decrease like previously but this is probably due to the increase of the assigned requests since the distances are a lot more important in New York compared to our previous (synthetic) simulations.

There are also other practical advantages with our method. First of all, we see that our method is able to scale up well and keep with real-life demands. For comparison, the article [9] which also referenced this dataset used 30 tasks for allocating 550 tasks. With our method, we managed to allocate 1000 agents to around 30 000 tasks per night. We also get rid of human bias: in general, a driver working for a driving platform accepts or declines the request himself. Drivers may refuse some requests due to the distance, which leaves the consumer in the uncertainty if he will be picked up or not. Our method gets rid of this bias and tries to arrange the requests so that it can be combined with other close requests favoring ride-sharing. Moreover, we speculate that our method can also be used to determine fleet size to met demands, adding more agents when too many requests are being left unattended and reducing the number of agents if the idle time becomes important, leading to a variable size of fleet.

6 Conclusion and Future Work

In order to treat the online multi-task assignment problem, we have proposed an alternative approach: online proactive multi-task assignment with resource availability anticipation, where we use a receding-horizon to anticipate which agents would be available with this horizon. Through our synthetic and real dataset benchmark simulations, we have shown that our method allows for generally assign a higher percentage of tasks to agents and that the agents tend to be less idle, leading to a better use of our resources. We have plans to further explore the concept of resource availability anticipation in the future. First of all, we plan to integrate a criterion to take into account the waiting times of the tasks for their completion, in order to try to minimize this waiting time. While the genetic algorithm is effective, it can be time-consuming in terms of calculation so we want to try to find another efficient but lighter allocation method. We are also planning to test our anticipation method with some algorithms that have been used for offline allocation or exploration but not in the context of online allocation.

References

[1] Saeed Alaei, MohammadTaghi Hajiaghayi & Vahid Liaghat (2012): *Online Prophet-Inequality Matching with Applications to Ad Allocation*. In: *Proceedings of the 13th ACM Conference on Electronic Commerce*, EC '12, Association for Computing Machinery, New York, NY, USA, p. 18–35, doi:10.1145/2229012.2229018.

[2] Javier Alonso-Mora, Samitha Samaranayake, Alex Wallar, Emilio Frazzoli & Daniela Rus (2017): *On-demand high-capacity ride-sharing via dynamic trip-vehicle assignment*. *Proceedings of the National Academy of Sciences* 114(3), pp. 462–467, doi:10.1073/pnas.1611675114.

[3] Niv Buchbinder, Kamal Jain & Joseph Seffi Naor (2007): *Online Primal-Dual Algorithms for Maximizing Ad-Auctions Revenue*. In: *Proceedings of the 15th Annual European Conference on Algorithms*, ESA'07, Springer-Verlag, Berlin, Heidelberg, p. 253–264, doi:10.1007/978-3-540-75520-3_24.

[4] Ethan Burns, J. Benton, Wheeler Ruml, Sungwook Yoon & Minh Do (2012): *Anticipatory On-Line Planning*. *Proceedings of the International Conference on Automated Planning and Scheduling* 22(1), pp. 333–337, doi:10.1609/icaps.v22i1.13533.

[5] Omar Cheikhrouhou & Ines Khoufi (2021): *A comprehensive survey on the Multiple Traveling Salesman Problem: Applications, approaches and taxonomy*. *Computer Science Review* 40, p. 100369, doi:10.1016/j.cosrev.2021.100369.

[6] Han-Lim Choi, Luc Brunet & Jonathan P. How (2009): *Consensus-Based Decentralized Auctions for Robust-task Allocation*. *Trans. Rob.* 25(4), p. 912–926, doi:10.1109/TRO.2009.2022423.

[7] Rutger Claes, Tom Holvoet & Danny Weyns (2011): *A Decentralized Approach for Anticipatory Vehicle Routing Using Delegate Multiagent Systems*. *IEEE Transactions on Intelligent Transportation Systems* 12(2), pp. 364–373, doi:10.1109/TITS.2011.2105867.

[8] Allegra De Filippo, Michele Lombardi & Michela Milano (2019): *How to Tame Your Anticipatory Algorithm*. In: *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, IJCAI'19, AAAI Press, p. 1071–1077, doi:10.24963/ijcai.2019/150.

[9] John Dickerson, Karthik Sankararaman, Aravind Srinivasan & Pan Xu (2018): *Allocation Problems in Ride-Sharing Platforms: Online Matching With Offline Reusable Resources*. *Proceedings of the AAAI Conference on Artificial Intelligence* 32(1).

[10] Xiao-Yue Gong, Vineet Goyal, Garud N. Iyengar, David Simchi-Levi, Rajan Udwani & Shuangyu Wang (2022): *Online Assortment Optimization with Reusable Resources*. *Management Science* 68(7), pp. 4772–4785, doi:10.1287/mnsc.2021.4134.

[11] Will Grathwohl, Kuan-Chieh Wang, Jörn-Henrik Jacobsen, David Duvenaud, Mohammad Norouzi & Kevin Swersky (2019): *Your Classifier is Secretly an Energy Based Model and You Should Treat it Like One*. *CoRR* abs/1912.03263, doi:10.48550/arXiv.1912.03263.

[12] Wesam Herbawi & Michael Weber (2011): *Ant Colony vs. Genetic Multiobjective Route Planning in Dynamic Multi-Hop Ridesharing*. In: *Proceedings of the 2011 IEEE 23rd International Conference on Tools with Artificial Intelligence*, ICTAI '11, IEEE Computer Society, USA, p. 282–288, doi:10.1109/ICTAI.2011.50.

[13] Wesam Herbawi & Michael Weber (2012): *The ridematching problem with time windows in dynamic ridesharing: A model and a genetic algorithm.* pp. 1–8, doi:10.1109/CEC.2012.6253001.

[14] Yuya Hikima, Yasunori Akagi, Naoki Marumo & Hideaki Kim (2022): *Online Matching with Controllable Rewards and Arrival Probabilities*. In Lud De Raedt, editor: *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, International Joint Conferences on Artificial Intelligence Organization, pp. 1825–1833, doi:10.24963/ijcai.2022/254. Main Track.

[15] John H. Holland (1992): *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA, doi:10.7551/mitpress/1090.003.0009.

[16] Ahmed Hussein & Alaa Khamis (2013): *Market-based approach to Multi-robot Task Allocation*. pp. 69–74, doi:10.1109/ICBR.2013.6729278.

[17] Alaa Khamis, Ahmed Hussein & Ahmed Elmogy (2015): *Multi-robot Task Allocation: A Review of the State-of-the-Art*, pp. 31–51. 604, doi:10.1007/978-3-319-18299-5_2.

[18] Annu Lambora, Kunal Gupta & Kriti Chopra (2019): *Genetic Algorithm- A Literature Review*. pp. 380–384, doi:10.1109/COMITCon.2019.8862255.

[19] Nixie S Lesmana, Xuan Zhang & Xiaohui Bei (2019): *Balancing Efficiency and Fairness in On-Demand Ridesourcing*. 32, Curran Associates, Inc.

[20] Aranyak Mehta (2013): *Online Matching and Ad Allocation*. *Found. Trends Theor. Comput. Sci.* 8(4), p. 265–368, doi:10.1561/0400000057.

[21] Vedant Nanda, Pan Xu, Karthik Abinav Sankararaman, John P. Dickerson & Aravind Srinivasan (2020): *Balancing the Tradeoff between Profit and Fairness in Rideshare Platforms during High-Demand Hours*. In: *Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society*, AIES '20, Association for Computing Machinery, New York, NY, USA, p. 131, doi:10.1145/3375627.3375818.

[22] Nirav Patel, N. S. Narayanaswamy & Alok Joshi (2020): *Hybrid Genetic Algorithm for Ridesharing with Timing Constraints: Efficiency Analysis with Real-World Data*. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, GECCO '20, Association for Computing Machinery, New York, NY, USA, p. 1159–1167, doi:10.1145/3377930.3389804.

[23] S. Rangriz, M. Davoodi & J. Saberian (2019): *A NOVEL APPROACH TO OPTIMIZE THE RIDESHARING PROBLEM USING GENETIC ALGORITHM*. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-4/W18, pp. 875–878, doi:10.5194/isprs-archives-XLII-4-W18-875-2019.

[24] M. Schreckenberg & J. Wahle (2001): *A Multi-Agent System for On-Line Simulations Based on Real-World Traffic Data*. In: *2014 47th Hawaii International Conference on System Sciences*, 4, IEEE Computer Society, Los Alamitos, CA, USA, p. 3037, doi:10.1109/HICSS.2001.926332.

[25] Hanna Sumita, Shinji Ito, Kei Takemura, Daisuke Hatano, Takuro Fukunaga, Naonori Kakimura & Ken-ichi Kawarabayashi (2022): *Online Task Assignment Problems with Reusable Resources*. *Proceedings of the AAAI Conference on Artificial Intelligence* 36(5), pp. 5199–5207, doi:10.1609/aaai.v36i5.20455.

[26] Hao Wang & Xiaohui Bei (2022): *Real-Time Driver-Request Assignment in Ridesourcing*. *Proceedings of the AAAI Conference on Artificial Intelligence* 36(4), pp. 3840–3849.

[27] Yang Wang, Chenxi Zhao & Shanshan Xu (2020): *Method for Spatial Crowdsourcing Task Assignment Based on Integrating of Genetic Algorithm and Ant Colony Optimization*. *IEEE Access* 8, pp. 68311–68319, doi:10.1109/ACCESS.2020.2983744.

Appendix

Table 1: Average results for the evaluation of the impact of the α parameter given different receding-horizons and the cases where there are more/less agents than requests.

α	$H(0)$	$H(1)$	$H(2)$	$H(3)$	$H(4)$	$H(5)$	$H(v)$
$\|R_\tau\| < \|A\|$							
Total traveled distance (m)							
0	1501	1524	1522	1501	1512	1487	1519
0.25	965	936	934	971	979	965	922
0.5	926	917	920	926	932	911	909
0.75	804	789	794	833	821	805	843
1	867	839	874	886	886	864	859
Idle time (s)							
0	1378.0	1350.0	1326.2	1330	1343.1	1350.9	1108.0
0.25	1715.6	1572.6	1517.5	1500.0	1541.4	1536.4	1766.5
0.5	1736.7	1535.5	1585.2	1610.7	1528.4	1570.0	1733.2
0.75	1936.7	1759.6	1761.1	1659.0	1600.8	1765.4	1668.8
1	1860.1	1706.8	1681.8	1682.1	1646.1	1674.2	1608.3
Percentage of assigned tasks (%)							
0	100	100	100	100	100	100	100
0.25	100	100	100	100	100	100	100
0.5	100	100	100	100	100	100	100
0.75	100	100	100	99.97	99.97	100	100
1	99.63	99.53	99.73	99.77	99.87	99.83	99.73
$\|R_\tau\| > \|A\|$							
Total traveled distance (m)							
α	$H(0)$	$H(1)$	$H(2)$	$H(3)$	$H(4)$	$H(5)$	$H(v)$
0	1355	1421	1426	1437	1440	1420	1273
0.25	1270	1334	1315	1301	1274	1261	1138
0.5	1243	1297	1273	1250	1206	1202	1080
0.75	1211	1277	1247	1221	1203	1811	1031
1	1207	1277	1258	1225	1198	1208	1044
Idle time (s)							
0	97.5	15.6	7.3	9.8	9.7	9.3	4.5
0.25	185.9	61.2	50.5	62.5	85.5	90.1	29.0
0.5	212.3	64.5	54.4	70.4	83.1	92.9	22.9
0.75	233.8	89.1	62.3	83.6	96.6	109.7	21.1
1	240.2	86.7	69.4	97.5	107.3	117.7	30.5
Percentage of assigned tasks (%)							
0	79.5	86.4	81.2	84.3	84.6	80.3	85.9
0.25	89.3	90.5	90.3	92.9	93.4	92.1	92.4
0.5	91.2	93.5	94.3	94.4	93.2	93.3	95.9
0.75	88.8	93.0	94.2	93.4	94.5	95.3	96.5
1	89.5	93.1	94.4	93.7	93.4	94.7	97.9

From Robot Self-Localization to Global-Localization: An RSSI Based Approach.

Athanasios Lentzas
Aristotle University
Thessaloniki, Greece
School of Informatics
Aristotle University of Thessaloniki
Thessaloniki, Greece
alentzas@csd.auth.gr

Dimitris Vrakas
Aristotle University
Thessaloniki, Greece
School of Informatics
Aristotle University of Thessaloniki
Thessaloniki, Greece
dvrakas@csd.auth.gr

Localization is a crucial task for autonomous mobile robots in order to successfully move to goal locations in their environment. Usually, this is done in a robot-centric manner, where the robot maintains a map with its body in the center. In swarm robotics applications, where a group of robots needs to coordinate in order to achieve their common goals, robot-centric localization will not suffice as each member of the swarm has its own frame of reference. One way to deal with this problem is to create, maintain and share a common map (global coordinate system), among the members of the swarm. This paper presents an approach to global localization for a group of robots in unknown, GPS and landmark free environments. The main idea relies on members of the swarm staying still and acting as beacons, emitting electromagnetic signals. These stationary robots form a global frame of reference and the rest of the group localize themselves in it using the Received Signal Strength Indicator (RSSI). The proposed method is evaluated, and the results obtained from the experiments are promising.

1 Introduction

Robotic Swarms is a topic with significant interest, especially with the increased use of large groups of unmanned air or ground vehicles [13]. Self localization and navigation is a crucial task for autonomous robots [3], both on indoor and outdoor navigation. Depending on the application of the robotic swarm, the robots may need the ability to discover their location.

Non robotic applications can also take advantage of indoor localization techniques. In case of emergency evacuation, especially on crowded buildings like malls, airports etc., it is beneficial to know the location of people inside the building as this can help with faster and safer evacuation. In order to achieve that, specific devices or smartphones can be used that take advantage of the proposed localization scheme.

Localization is the ability of a robot to know its position and orientation. The location could be relative to other robots or absolute on a common coordinate system. While operating outdoors, a robot can rely on GPS to localize itself. The aforementioned sensor can not be used on indoor environments. Simultaneous localization and mapping (SLAM) algorithms are extensively used [2], although when small and relatively simple robots are needed, their high complexity and computational cost is a major drawback. Several techniques exploiting beacons and landmarks for localization have been proposed in the literature as well as techniques using inertial navigation systems, magnetic, sound and optic based navigation [14].

Inertial localization is also a common approach. Based on odometers, accelerometers and gyroscopes, one can determine the orientation and direction of the robot [7]. Although the results provided

A. Ferrando, R.C. Cardoso (Eds.): Third Workshop on
Agents and Robots for reliable Engineered Autonomy 2023 (AREA'23)
EPTCS 391, 2023, pp. 18–25, doi:10.4204/EPTCS.391.4

are robust against environmental changes, this technique is prone to error accumulation [15]. Kalman filters have been expensively used in order to improve accuracy [8].

Radio frequency localization is the approach usually preferred. Beacons can cover a wide area, radio waves can penetrate most materials while the installation cost is relatively low [11]. Received Signal Strength Indicator (RSSI) is mainly exploited for robot localization [1, 19, 27]. Time of arrival and time difference of arrival of two signals that are known to have different propagation is also a common approach for localization.

Localization based on RSSI is proposed in the Ladybug algorithm [10]. A robot equipped with sensors able to measure the strength of the received electromagnetic signal, is able to identify the location of the source of the signal and navigate to it. The source could be either a beacon or another robot. The LadyBug Algorithm was effective and had numerous benefits compared to similar approaches, such as I-Bug [22]. RSSI could be an efficient solution when deploying a robotic swarm on GPS denying environment. Our motivation was to propose an approach where a robotic swarm could be able to extract the location of each robot in the same coordinate system using local sensing only, allowing the LadyBug algorithm to be implemented on the swarm. Sharing a common coordinate system is crucial for tasks such as self-assembly.

The benefits of localizing the swarm in a common coordinate system are presented in [18]. The ability of the swarm to self-assemble on a specific shape was realized by placing four robots on a specific layout, marking the coordinate system and using trillateration. While the proposed solution is efficient, it's main drawback is the limitation to two dimensions and the requirement of hand placing the four initial robots. Our incentive was to propose an approach where the robots will use local sensing but will have the ability to localize in a global system without the aforementioned limitations.

Our approach would have numerous benefits such as: a) the swarm will share a common localization scheme, sharing the same map. b) The localization scheme is based on three members of the swarm equipped with a beacon instead of four. c) No manual placement is required. The beacons could be anywhere on the operation area. d) Our approach is able to localize in three dimensions. e) The proposed solution is robust as, in case of a possible failure of a beacon, another robot from the swarm can replace it.

2 Related Work

Radio frequency identification technology (RFID) is used in [23] to localize robots navigating indoors. The passive RFID tags installed, divided the area into a grid. As a robot, equipped with an RFID reader, explores the area, the reader reads the tag. The position is then estimated by correlating the ID of the tag with a map containing the localization information of the tags. Despite the high accuracy, the need for a map containing the ID and location of the tags limits the practical applications.

Pseudolites (i.e. pseudo satellites) have been proposed for indoor localization [24]. The main idea is to receive the GPS signal and transmit it indoors using signal repeaters [26]. The major drawback of this approach is the increased cost of the network installation. Additionally, in case the main signal receiver fails the whole network is unusable.

A localization method based on RSSI of heterogeneous sources (i.e. WLAN, GSM etc) is presented in [21]. By analyzing the fingerprint and strength of the received signal, the robot is able to localize itself by comparing it with a fingerprint map. While exploiting already existing infrastructures, negating the need for a network deployment, the main disadvantage is the need for a fingerprint map covering all the area of interest. RFID sensors can also be used for indoor localization. In [6], authors presented a

simulator that allows modeling environments and testing the deployment of RFID solutions. Tags and antennas are placed and RSSI is exploited to find the location of each tag.

A distributed localization method based on swarm intelligence algorithms is presented in [4]. Particle swarm optimization and an approach based on backtracking search algorithm is proposed. The proposed method operates in three stages. The first stage allows a robot to estimate the distance from a reference node using the Sum-Dist algorithm [9]. The second stage estimates the position of the robot using the min-max technique [9, 20]. Finally the third stage the accuracy improves by re-evaluating the position based on the position of the neighbors.

3 Problem Statement

The localization of the robot is based on RSSI. Given an electromagnetic signal, the coordinates of the source in 3D space can be calculated in the robocentric system and the in the global coordinate system. In order to achieve our goal, three beacons are employed, creating a 3D common coordinate system for all robots. In our work, the following assumptions were made: a) All robots are equipped with sensors capable of reading the RSSI of a received signal. b) Two robots act as anchors. c) A beacon/robot is placed at the center of the coordinate system.

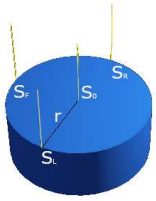


Figure 1: The position of the four sensors on the robot.

The main localization scheme employs three receivers and one transceiver. The transceiver allows the robot to act as a beacon or exchange information with the rest of the swarm. The receivers are placed on the front, right and left of a robot with radius r and the transceiver on the center, as seen in Fig. 1. Each sensor is able to identify both the source of the signal and the signal strength. The signal strength (S) is inversely proportional to the square of the distance (d) between the sensor and the source (1). Using (1) the distance between the source and each sensor can be calculated (1).

$$S = \frac{1}{d^2} \qquad\qquad d = \sqrt{\frac{1}{S}} \tag{1}$$

The coordinates of each sensor, in the robot-centric system, can be seen on Table 1. Let (x_s, y_s, z_s) be the coordinates of the signal source in the robot-centric system. Using (1) the distance between each sensor (center, front, left, right) is calculated.

$$\begin{aligned} d_0^2 &= x_s^2 + y_s^2 + z_s^2 & d_f^2 &= x_s^2 + 2rx + r^2 + y_s^2 + z_s^2 \\ d_l^2 &= x_s^2 + y_s^2 + z_s^2 + 2ry + r^2 & d_r^2 &= x_s^2 + y_s^2 + z_s^2 - 2ry + r^2 \end{aligned} \tag{2}$$

Solving the equation system (2), the coordinates of the source, x_s and y_s respectively, can be calculated. By replacing the values of x_s and y_s on (2) the z_s coordinate can be calculated. Similarly (and

Sensor	Robocentric Coordinates (x,y)	Enchanced Localization Coordinates
S_0	(0,0,0)	(0,0)
S_L	(0,r,0)	(0,-r)
S_R	(0,-r,0)	(0,r)
S_F	(r,0,0)	(r,0)
S_B	-	(-r,0)

Table 1: Position of the sensors

knowing the coordinates of the source) the coordinates of the beacon (x_b, y_b, z_b) and the anchor robots (x_{a1}, y_{a1}, z_{a1}), (x_{a2}, y_{a2}, z_{a2}) can be calculated.

3.1 Enhanced Localization

In order to further enhance the localization procedure, redundant sensors were added. Our implementation uses five sensors. All the receivers are placed on the perimeter of the robot thus creating four sensor groups (triangle formed by the transceiver in the center and two receivers on the perimeter). The updated robot-centric coordinates can be seen on Table 1. The estimates of the four groups are averaged, resulting in more accurate estimation of the source's position.

3.2 Global Localization

As already described in the previous section, the robot is able to calculate the coordinates of the three beacons in the robocentric system. Knowing their relative positions, the global position of the robot can be calculated. Let A be the center of the global coordinate system and B, C the two beacons that define axis X and Y respectively.

Using vectors $\vec{AB}$ and $\vec{AC}$ we define three new vectors using the outer products, as seen in (3), and normalize them.

$$\vec{z} = \vec{AB} \otimes \vec{AC} \qquad \vec{y} = \vec{z} \otimes \vec{AB} \qquad \vec{x} = \vec{y} \otimes \vec{z} \tag{3}$$

Let $\vec{r}$ be the vector between the center of the robot R and A. We can now calculate the coordinates of the robot in the global system defined by $\vec{x}$, $\vec{y}$ and $\vec{z}$. The coordinates of the robot, (x_r, y_r, z_r), are the dot product of $\vec{r}$ and $\vec{x}$, $\vec{y}$, $\vec{z}$ respectively.

4 Experimental Results

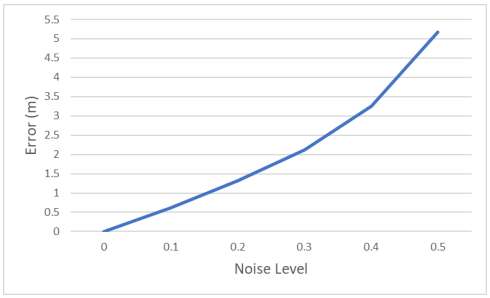
In order to evaluate the performance of our localization approach we implemented the algorithm in the Webots Open Source Robot simulator [25]. A swarm of four identical robots was used, where three of them were serving as beacons. A generic round shaped robot with 20cm radius was employed, equipped with four radio receivers capable of identifying the signal strength on its perimeter and a radio transceiver on the center.

The three robots acting as beacons were constantly emitting a radio signal with a unique ID, allowing the robot to identify the source of the signal. In order to further increase the accuracy, each beacon was transmitting on a different channel reducing the generated noise. For each source the RSSI of 100 packets

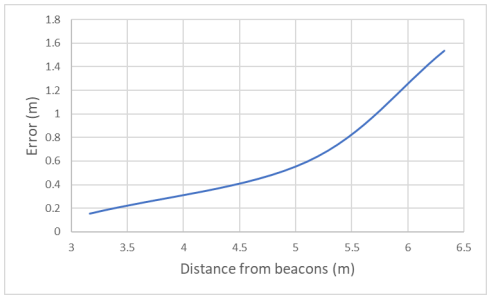
was averaged reducing the noise generated error. The robot was cycling through the three predefined channels allowing it to identify and locate the robocentric position of each beacon. Lastly the global coordinates were calculated and reported.

During the experiment, the robot was static while calculating it's position. After a successfully calculation it was moving randomly and the new position was calculated and reported. This process was repeated 10 times. The error (i.e. euclidean distance between the real and calculated global location) for each location was averaged and used as a metric. In a noiseless environment, the calculated position was the same with the real one, no matter the distance from the beacons or the orientation of the robot.

In order to evaluate our approach in more real-like conditions, noise was gradually introduced into the experiment. In total the process described above was repeated 5 times with 5 different noise levels: 10%, 20%, 30%, 40% and 50%. Noise strength is the standard deviation of the Gaussian noise added to the signal strength. The rest of the conditions were controlled (i.e. the random seed) making sure the experiment would be the same, providing comparable results. As seen in Fig. 2a, for 10% noise the error was 0.6m while for 20% noise the calculated position was 1.4m off. Further increasing the noise had a bigger impact.



(a) Error per noise level



(b) Error per distance from the beacons

Figure 2: Localization error.

As noise is related to the distance of the source, an experiment was performed where the distance between the robot and beacons was gradually increased. The three beacons were hand placed forming an isosceles triangle. This placement guaranteed that the distance between the robot and two beacons would be the same. The side of the triangle formed was 4m and the noise level was set to 10%. The experiment was repeated 10 times and the results were averaged. As seen in Fig. 2b the error was 0.1m when the robot was closer to the source. The best results were observed when the robot was between 3m and 3.5m from the beacons. Increasing the distance between the robot and source results in increased noise level, making it harder to correctly identify the position of the robot. It is worth mentioning that for distances up to almost 6m the calculated position error was less than 1m.

An experiment was also performed using a small swarm of three robots. The first robot was placed away from the beacons, the second close to the two of them while the third close to the three beacon robots. The robots reported their global position 10 times with a time delay added between each calculation. The noise level was set to 10%. Further noise was introduced, as each robot was emitting a random signal, simulating an environment where the robots communicate with each other. In Fig. 3 the position of each robot can be seen. The results show that even with the extra noise (i.e. robots communicating with each other), the localization process is not heavily affected. It is worth mentioning that the distance between the robot and the beacons has an impact on the calculations (as already discussed).

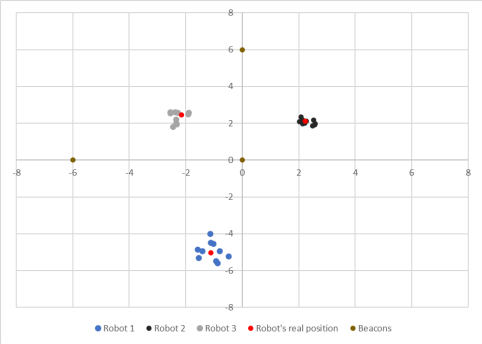


Figure 3: Calculated position of each robot.

5 Conclusion & Future Work

In this paper, a novel global localization method was presented. Our approach exploits RSSI from a signal emitted by three beacons, to localize a robot on the coordinate system formed by those beacons.

Our approach was evaluated in a noiseless environment as well as in an environment with different noise levels. While for lower noise levels the results were accepted, for increased noise levels the error was bigger. This requires further experiments as noise levels in the real world are affected by many factors, such as the materials of the obstacles, reflectance etc.

Additionally, the impact of the distance between the robot and the source of the signal was investigated. Placing the robot close to the beacons negatively affects the ability to localize itself. Increasing the distance up to a certain threshold has low impact. Further increasing the distance produced increased error.

Although a small swarm was simulated, allowing the observation of how multiple robots communicating will impact the performance, the robots were not exchanging any location related information. In the near future our focus will be on investigating ways that will allow the robots to correlate their individual localization information and create a more accurate global map.

Furthermore, the current experiments were performed in a simulated environment. Even with noise added it is not as close as the real world. Experiments will be performed on a real robot. That evaluation is crucial as real world noise can not be reproduced easily (i.e. reflection of the signal, non-uniform noise etc.).

Another important part of our future work is to integrate the localization presented with the LadyBug algorithm [22]. As LadyBug provides robust results, the proposed method will allow its application to robotic swarms.

6 Acknowledgements

This research was carried out as part of the project «Beacon: An Intelligent Dynamic Signage System for Emergency Situations» (Project code: KMP6-0094837) under the framework of the Action «Investment Plans of Innovation» of the Operational Program «Central Macedonia 2014 2020», which is co-funded by the European Regional Development Fund and Greece.

References

[1] Paolo Barsocchi, Stefano Lenzi, Stefano Chessa & Gaetano Giunta (2009): *A Novel Approach to Indoor RSSI Localization by Automatic Calibration of the Wireless Propagation Model*. In: *VTC Spring 2009 - IEEE 69th Vehicular Technology Conference*, pp. 1–5, doi:10.1109/VETECS.2009.5073315.

[2] G. Bresson, Z. Alsayed, L. Yu & S. Glaser (2017): *Simultaneous Localization and Mapping: A Survey of Current Trends in Autonomous Driving*. *IEEE Transactions on Intelligent Vehicles* 2(3), pp. 194–220, doi:10.1109/TIV.2017.2749181.

[3] I.J. Cox (1991): *Blanche-an experiment in guidance and navigation of an autonomous robot vehicle*. *IEEE Transactions on Robotics and Automation* 7(2), pp. 193–204, doi:10.1109/70.75902.

[4] Alan Oliveira de Sa, Nadia Nedjah & Luiza de Macedo Mourelle (2016): *Distributed efficient localization in swarm robotic systems using swarm intelligence algorithms*. *Neurocomputing* 172, pp. 322–336, doi:10.1016/j.neucom.2015.03.099. Available at https://www.sciencedirect.com/science/article/pii/S0925231215010498.

[5] Qian Dong & Waltenegus Dargie (2012): *Evaluation of the reliability of RSSI for indoor localization*. In: *2012 International Conference on Wireless Communications in Underground and Confined Areas*, pp. 1–6, doi:10.1109/ICWCUCA.2012.6402492.

[6] Salvatore D'Avella, Matteo Unetti & Paolo Tripicchio (2022): *RFID Gazebo-Based Simulator With RSSI and Phase Signals for UHF Tags Localization and Tracking*. *IEEE Access* 10, pp. 22150–22160, doi:10.1109/ACCESS.2022.3152199.

[7] Giovanni Fusco & James M. Coughlan (2018): *Indoor Localization Using Computer Vision and Visual-Inertial Odometry*. In Klaus Miesenberger & Georgios Kouroupetroglou, editors: *Computers Helping People with Special Needs*, Springer International Publishing, Cham, pp. 86–93, doi:10.1007/978-3-319-94274-2_13.

[8] Guanghui Hu, Weizhi Zhang, Hong Wan & Xinxin Li (2020): *Improving the Heading Accuracy in Indoor Pedestrian Navigation Based on a Decision Tree and Kalman Filter*. *Sensors* 20(6), doi:10.3390/s20061578. Available at https://www.mdpi.com/1424-8220/20/6/1578.

[9] Koen Langendoen & Niels Reijers (2003): *Distributed localization in wireless sensor networks: a quantitative comparison*. *Computer Networks* 43(4), pp. 499–518, doi:10.1016/S1389-1286(03)00356-6. Available at https://www.sciencedirect.com/science/article/pii/S1389128603003566. Wireless Sensor Networks.

[10] Athanasios Lentzas & Dimitris Vrakas (2020): *LadyBug. An Intensity based Localization Bug Algorithm*. In: *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, 1, pp. 682–689, doi:10.1109/ETFA46521.2020.9212115.

[11] Junjie Liu (2014): *Survey of Wireless Based Indoor LocalizationTechnologies*. *Department of Science and Engineering Washington University*.

[12] O. Michel (2004): *Webots: Professional Mobile Robot Simulation*. *Journal of Advanced Robotics Systems* 1(1), pp. 39–42. Available at http://www.ars-journal.com/International-Journal-of-Advanced-Robotic-Systems/Volume-1/39-42.pdf.

[13] Nadia Nedjah & Luneque Silva Junior (2019): *Review of methodologies and tasks in swarm robotics towards standardization*. *Swarm and Evolutionary Computation* 50, p. 100565, doi:10.1016/j.swevo.2019.100565. Available at https://www.sciencedirect.com/science/article/pii/S2210650217308398.

[14] Huthaifa Obeidat, Wafa Shuaieb, Omar Obeidat & Raed Abd-Alhameed (2021): *A Review of Indoor Localization Techniques and Wireless Technologies*. *Wireless Personal Communications*, doi:10.1007/s11277-021-08209-5.

[15] Enrico Petritoli, Fabio Leccese & Mariagrazia Leccisi (2019): *Inertial Navigation Systems for UAV: Uncertainty and Error Measurements*. In: *2019 IEEE 5th International Workshop on Metrology for AeroSpace (MetroAeroSpace)*, pp. 1–5, doi:10.1109/MetroAeroSpace.2019.8869618.

[16] Joseph A. Rothermich, M. İhsan Ecemiş & Paolo Gaudiano (2005): *Distributed Localization and Mapping with a Robotic Swarm*. In Erol Şahin & William M. Spears, editors: *Swarm Robotics*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 58–69, doi:10.1007/978-3-540-30552-1_6.

[17] Stergios I. Roumeliotis & George A. Bekey (2000): *Distributed Multi-Robot Localization*, pp. 179–188. Springer Japan, Tokyo, doi:10.1007/978-4-431-67919-6_17.

[18] Michael Rubenstein, Alejandro Cornejo & Radhika Nagpal (2014): *Programmable self-assembly in a thousand-robot swarm*. *Science* 345(6198), pp. 795–799, doi:10.1126/science.1254295.

[19] Jirapat Sangthong, Jutamas Thongkam & Sathapom Promwong (2020): *Indoor Wireless Sensor Network Localization Using RSSI Based Weighting Algorithm Method*. In: *2020 6th International Conference on Engineering, Applied Sciences and Technology (ICEAST)*, pp. 1–4, doi:10.1109/ICEAST50382.2020.9165300.

[20] Andreas Savvides, Heemin Park & Mani B. Srivastava (2002): *The Bits and Flops of the N-Hop Multilateration Primitive for Node Localization Problems*. In: *Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*, WSNA '02, Association for Computing Machinery, New York, NY, USA, p. 112–121, doi:10.1145/570738.570755.

[21] M. Stella, M. Russo & D. Begušić (2014): *Fingerprinting based localization in heterogeneous wireless networks*. *Expert Systems with Applications* 41(15), pp. 6738–6747, doi:10.1016/j.eswa.2014.05.016. Available at https://www.sciencedirect.com/science/article/pii/S0957417414002966.

[22] Kamilah Taylor & Steven M. LaValle (2009): *I-Bug: An intensity-based bug algorithm*. In: *2009 IEEE International Conference on Robotics and Automation*, pp. 3981–3986, doi:10.1109/ROBOT.2009.5152728.

[23] R. Tesoriero, R. Tebar, J.A. Gallud, M.D. Lozano & V.M.R. Penichet (2010): *Improving location awareness in indoor spaces using RFID technology*. *Expert Systems with Applications* 37(1), pp. 894–898, doi:10.1016/j.eswa.2009.05.062. Available at https://www.sciencedirect.com/science/article/pii/S095741740900503X.

[24] Xiaoguang Wan & Xingqun Zhan (2011): *The Research of Indoor Navigation System using Pseudolites*. *Procedia Engineering* 15, pp. 1446–1450, doi:10.1016/j.proeng.2011.08.268. Available at https://www.sciencedirect.com/science/article/pii/S1877705811017693. CEIS 2011.

[25] Webots: *http://www.cyberbotics.com*. Available at http://www.cyberbotics.com. Commercial Mobile Robot Simulation Software.

[26] Rui Xu, Wu Chen, Ying Xu & Shengyue Ji (2015): *A New Indoor Positioning System Architecture Using GPS Signals*. *Sensors* 15(5), pp. 10074–10087, doi:10.3390/s150510074. Available at https://www.mdpi.com/1424-8220/15/5/10074.

[27] Zheng Yang, Zimu Zhou & Yunhao Liu (2013): *From RSSI to CSI: Indoor Localization via Channel Response*. *ACM Comput. Surv.* 46(2), doi:10.1145/2543581.2543592.

Safe and Robust Robot Behavior Planning via Constraint Programming

Jan Vermaelen Tom Holvoet
imec-DistriNet, KU Leuven, Belgium
jan.vermaelen@cs.kuleuven.be tom.holvoet@cs.kuleuven.be

The safe operation of an autonomous system is a complex endeavor, one pivotal element being its decision-making. Decision-making logic can formally be analyzed using model checking or other formal verification approaches. Yet, the non-deterministic nature of realistic environments makes these approaches rather troublesome and often impractical. Constraint-based planning approaches such as Tumato have been shown to be capable of generating policies for a system to reach a stated goal and abiding safety constraints, with guarantees of soundness and completeness by construction. However, uncertain outcomes of actions in the environment are not explicitly modeled or accounted for, severely limiting the expressiveness of Tumato.

In this work, we extend Tumato with support for non-deterministic outcomes of actions. Actions have a specific intended result yet can be modeled to have alternative outcomes that may realistically occur. The adapted solver generates a policy that enables reaching the goals in a safe manner, even when alternative outcomes of actions occur. Furthermore, we introduce a purely declarative way of defining safety in Tumato, increasing its expressiveness. Finally, the addition of cost or duration values to actions enables the solver to restore safety when necessary, in the most preferred way.

1 Introduction

Autonomous robotic systems are becoming increasingly popular, both in industry and households. The number and complexity of tasks they are expected to execute are expanding. However, generating a plan providing both productive (goal-oriented) and safe behavior is far from trivial. A plan must be constructed, given the actions that the robot can execute, the information about the environment, and the desired goals. During planning, additional safety constraints have to be taken into account to generate a safe plan while trying to achieve the goals. If such a safe and productive plan can be generated, it is sound by construction.

The main contribution of this paper is to support foreseeable non-deterministic transitions while guaranteeing safety when planning robot behavior. For this purpose, we focus on constraint-based planning and build further upon **Tumato**, a planning framework by Hoang Tung Dinh et al. [5]. Firstly, we support specifying and accounting for foreseeable non-deterministic *alternative effects* of actions rather than assuming a purely deterministic system. Effects of actions in the real world are virtually never fully deterministic. Non-determinism can arise from varying weights of payloads, various ground surfaces to navigate across, and small measuring and actuation errors, to name just a few. Secondly, the extension supports specifying *safety conditions* explicitly. In the original version of Tumato, to obtain a similar result, one has to consider all state-action combinations that could lead to unsafe states separately, which is error-prone and inflexible when the specification of the system evolves. Furthermore, the generated behavior *always* has to adhere to the safety conditions, even when alternative effects occur.

Since the environment, and hence the effects of actions, are most often not deterministic in practical robotic applications, the policy must be sufficiently robust to deal with this kind of uncertainty. Ideally, all

A. Ferrando, R.C. Cardoso (Eds.): Third Workshop on
Agents and Robots for reliable Engineered Autonomy 2023 (AREA'23)
EPTCS 391, 2023, pp. 26–41, doi:10.4204/EPTCS.391.5

contingencies are taken into account. A first step is to pursue a *complete* policy. For each state in which the system could be, the policy must provide the actions to execute next. If the system unexpectedly arrives in an unintended state, the operation can continue. We maintain this powerful feature of Tumato's original approach. In a second step, we take the uncertainty into account by allowing the effects of actions to be modeled in a non-deterministic way. In our approach, we assume that each action has one intended outcome. We call this outcome the *nominal* effect of the action. Additionally, each action can have a number of *alternative* effects. These effects *could* emerge instead of the nominal one, but they are not intentional. For the goal-oriented aspect of planning, only the nominal effect is relevant. Alternative effects are unintentional outcomes and can not reliably be used to achieve a goal. However, when dealing with safety, also the alternative effects must be taken into account as there is a possibility they occur.

Due to external causes, the system might still end up in an *unforeseen* state. Although for such events safety can not be guaranteed (an external force might put the system in an unsafe state directly), the planner will make sure that the policy contains instructions on how to get back on (safe) track to the goal immediately. If multiple such instructions are possible, the planner is capable of selecting the preferred one based on cost values assigned to actions.

The paper is structured as follows. Section 2 discusses the related work to outline the necessary background and provides an overview of Tumato. Section 3 briefly introduces the use case of the robotic system used to illustrate the proposed extension. Section 4 explains and motivates the approach. Section 5 elaborates and analyzes the extension of the specification using the robotic system from the case study. Section 6 discusses the approach and its results. Finally, Section 7 draws conclusions.

2 Background and Related Work

Well-thought-out behavior planning is essential for the safe operation of autonomous safety-critical robots. Furthermore, practical systems often involve a degree of non-determinism that needs to be addressed. In this section, we point to the related work necessary to provide a background for the explored planning approach. However, all (fully observable) non-deterministic planning can be considered related.

Traditionally, the behavior of robots has been defined manually. Finite State Machines (FSMs) are often used to represent the robot's behavior [9, 15]. However, FSMs are known not to cope well with the increasing complexity of the behavior. It is non-trivial to manually specify sound and complete behavior for larger and more complex systems, and one has to rely on simulation and verification approaches to check whether the behavior effectively meets the requirements. This problem can partly be solved by automatically generating the behavior based on a model of the system, along with a representation of the desired requirements.

Different specification languages and planners have been proposed. For example, Linear Temporal Logic (LTL) [8] is often used in robotics. Techniques exist to use (fragments of) LTL to generate FSMs [14, 20], or to compile them into PDDL [1]. Since LTL can take all contingencies into account, the generated behavior will be sound and complete, a property we also value. Further, modeling state-based safety conditions explicitly in LTL should require limited effort. Two other examples are Temporal Action Logic (TAL) [6, 7] and the previously mentioned, more generic Planning Domain Definition Language (PDDL) [11, 13]. Both TAL and PDDL generally rely on replanning at run-time to cope with contingencies. They do not guarantee completeness of the behavior since the replanning could fail due to an unrealizable specification. This lack of completeness would only be detected at run-time. Similarly, in a more practical context, probabilistic planners can be used within the ROSPlan [3] framework directly [2].

To a certain extent, robustness can be obtained by explicitly dealing with non-determinism, as covered in the book *Automated Planning and Acting* by Malik Ghallab et al. [12]. The planning can freely make use of the non-deterministic effects of actions to reach the goal. Unlike this approach, we opt to define one (intended) nominal effect for each action while recognizing alternative (less likely and less desired) outcomes. This is more closely related to practical behavior planning problems. Note that, in the non-deterministic context, the definition of a *Safe Solution* is a policy in which the goal is reachable from the initial state [12]. This definition is different from the additional safety constraints that we are imposing on the system to reach the goal *in a safe manner*.

Tomas Geffner et al. introduce a SAT encoding for fully observable non-deterministic planning [10]. A distinction is made between *fair* and *unfair* non-deterministic actions. In our work, we focus on fair actions and do not consider adversarial actions or agents. We do acknowledge that probabilistic effects of actions are difficult to estimate correctly, while they should not be considered fully non-deterministic either.

We have surveyed existing frameworks combining safe and robust planning before [19]. The use of Markov Decision Processes (MDPs) [17] for probabilistic planning and, to a smaller extent, Simple Temporal Networks (STNs) [4] for temporal scheduling have been explored. Especially their extensions are able to explicitly guarantee safety while dealing with uncertainties. In this work, avoiding the need for (often inherently imprecise) probabilistic values, we investigate and extend the promising constraint programming approach Tumato by Hoang Tung Dinh et al. [5].

Tumato Hoang Tung Dinh et al. [5] obtain sound and complete behavior via constraint programming. As the specification of a system has to contain information about the environment, the actions, and the goals of the system, as well as a set of safety rules, it effectively combines classical planning (using states, actions, and goals) with constraint programming to enforce safety.

Constraint-based planning is achieved by (automatically) translating the entire specification into constraints. Trivially, preconditions of actions constrain whether or not the action can be executed. Furthermore, the effects of actions on the state of the system and whether or not an action is executed in the first place are modeled as constraints. One Constraint Satisfaction Problem (CSP) [18] is generated for each valid starting state of the system. The set of CSPs yielding from the specification can be solved offline. Tumato currently employs Choco-solver [16], yet we try to maintain transparency to the exact solver used. The first execution step found by each individual CSP is selected as the set of actions for the corresponding state. The final result is a mapping from every state to the actions that have to be executed in that state. We will call this mapping the *policy*. If a solution exists, we consider the specification to be realizable. Otherwise, we say the specification is unrealizable. If a sound and complete policy exists, it will be found by the constraint solver. The obtained policy can safely be used at run-time without requiring online re-planning. For further details on the general approach, please refer to the original work on Tumato [5].

In the remainder of this section, we will give an overview of the specification of a model in Tumato. The specification contains information on the state space of the system, the actions that can be executed, the relevant safety constraints, and the goals of the system. Our work will extend this specification where necessary, as described in Section 5.

States A state vector is used to represent the states in which the system can reside. This vector consists of a set of discrete state variables. Each of the state variables represents one aspect of the state. At run-time, the state variables are updated by a monitoring module. This module is responsible for translating

the sensor readings and other input to discrete values in the state variables. An example of a state variable is the location of the robot. A state variable $S_{Location}$ can represent the different discrete locations at which the robot can reside. For example, $\{corridor, charger, workstation\_1, workstation\_2\}$ includes a common corridor, a charging station for the robot, and two interactable workstations. Additional workstations and locations can be added as needed. Alternatively, when more detailed location information is required, a more fine-grained location discretization can be utilized.

Actions Actions represent elemental behavior. They are the smallest unit of execution that we consider during planning. As an example, both *move_one_cell_forward* and *move_to_charging_station* are possible actions, yet they relate to planning at different abstraction levels. Actions can be specified to have *preconditions* that must hold before they can be executed. Actions also can be specified to make use of certain *resources*. The purpose of allocating resources is to prevent two actions from being simultaneously executed if they make use of at least one resource in common. Finally, actions usually have an *effect* on the state. After executing an action, which can take an arbitrary amount of time, the state has changed according to that effect. An example of a simple action is **move_to_workstation_1**, which:

- controls one resource: *motors*,
- has one precondition: $S_{Location} = corridor$,
- has the effect: $S_{Location} = workstation\_1$.

Reaction Rules As their name indicates, reaction rules can be used to specify reactive behavior. They are logical rules on the current state and the executed actions. If a certain condition holds on the current state, either a specific action has to be executed or is not allowed to be executed. For example, *if the robot resides in the corridor with a workstation-sensitive actuator (for example, a conveyor belt) active, then it should deactivate that actuator (for example, by executing the action **stop_conveyor**)*. Let $S_{Conveyor}$ represent whether or not the conveyor is currently active, as $\{on, off\}$.

$$(S_{Location} = corridor \wedge S_{Conveyor} = on) \Rightarrow Exec(\textbf{stop\_conveyor}).$$

We will discuss the use of reaction rules in more detail in Section 5.2.1.

Goals The goals define conditions that have to be achieved by the system. Because of the constraint-based approach, different formats of goals can be specified. For this example, we focus on *prioritized* and *conditional* goals. A conditional goal is a goal that is active only when a specified condition is met. Consider a state variable S_{Load} that can be $\{loaded, free\}$. *If the robot is loaded with an item, the goal is to unload that item* (2) and hence, deliver it. Analogously for picking up an item, a conditional goal is specified (3). Priorities can indicate which conditional goals should be taken into account first. Consider a state variable $S_{Battery}$ that can be $\{low, ok\}$. A conditional goal can be responsible for recharging the battery when the battery level becomes *low* (1).[1] This conditional goal should get priority over the transport goals (2) and (3). Priorities are determined by the order in which the goals are specified.

$$(S_{Battery} = low) \Rightarrow S^{Goal}_{Battery} = ok \quad (1)$$

$$(S_{Load} = loaded) \Rightarrow S^{Goal}_{Load} = free \quad (2)$$

$$(S_{Load} = free) \Rightarrow S^{Goal}_{Load} = loaded \quad (3)$$

[1] For a practical application, it is important that the monitoring module only updates *S_Battery* to *ok* when the battery has been sufficiently charged. An earlier update would result in (undesired) shorter operation cycles.

3 Case Study

As hinted toward in the previous section, we consider a battery-powered Autonomous Mobile Robot (AMR) for validation purposes. The AMR, shown in Figure 1, operates in an automated demo factory. The workstations present in the factory are capable of executing different operations on small, standardized workpieces. Adjacent workstations share a conveyor belt, yet not all workstations are connected. The AMR is responsible for moving around workpieces in the factory. We use this system as an example to paint out shortcomings of the current constraint programming planning approach and to illustrate how to mitigate these shortcomings.

The base of the AMR has three omnidirectional wheels (not visible in the picture), enabling the AMR to move in all directions as well as to turn in place. Combined with information regarding the layout of the factory, the AMR can move to any location and obtain any orientation within the factory. Only 2D movement and orientation are applicable. The AMR can navigate between the different workstations and a charging station, as well as the *corridor* connecting the different locations. The AMR has a camera and LIDAR system for perception, along with a number of proximity sensors. On top of the base, a looping conveyor belt is mounted at a fixed height. This setup is used to dock to workstations and receive or deliver workpieces from or to workstations. Infrared sensors attached near the conveyors help with the alignment when docking. Furthermore, an emergency stop button is mounted on top, as well as a beacon for visual signaling.

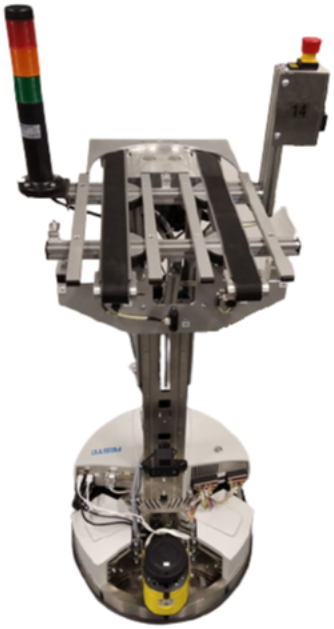


Figure 1: The AMR

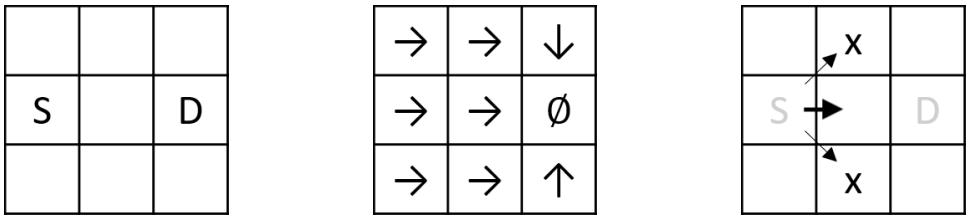
One or more AMRs can be operating simultaneously on the factory floor alongside human agents. The AMRs and humans are not physically separated, increasing the importance of generating safe behavior. An AMR should complete its assigned transport without creating any unsafe situations. As an additional hurdle, the effects of actions can unintentionally vary, even within a relatively controlled environment such as a factory.

4 Uncertainty and Safety

As mentioned in Section 1, we account for uncertainty by adding foreseeable alternative outcomes to actions. Merely to illustrate the approach, this section introduces a grid-based 2D navigation planning example, although the approach can be more fully appreciated with examples from the case study, see Section 5.

4.1 Uncertainty

Assume a 3×3 grid, with starting point S and destination D as illustrated in Figure 2a. We assign each *cell* in the grid to a corresponding state. Further, we define the actions *move_up, move_down, move_left*, and *move_right*. In a deterministic setting, each action always moves the agent exactly one cell in the intended direction of that action. In this example, the shortest plan from starting point S to destination D executes *move_right* twice. An example of a complete policy for this planning problem is shown in Figure 2b.



(a) Starting point *S* and destination *D*

(b) Policy for a deterministic approach

(c) *Move_right* in *S* with two alternative outcomes (X)

Figure 2: Illustrative 3×3 grid example

More realistically, however, in a non-deterministic planning setting, actions are not always successful. Firstly, the action could have no effect. For example, the agent did not move far enough to reach the new cell. Since we are dealing with policy-based execution, the same action will be executed again, presumably leading to a new state eventually. Note that this implicit assumption has to be taken into account when constructing the system. If an action could fail indefinitely, for example, if the agent could have insufficient force to move up a hill, the action should be reconsidered. Either the model is not adequate (the effect does not correspond to the real world), or the (physical) implementation of the action has to be adapted.

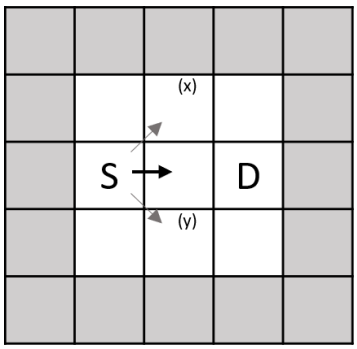
Secondly, deviations can occur. When executing *move_right* at the starting point *S*, two states (cells) could be reached unintentionally, as shown in Figure 2c. Such outcomes could either be observed after deploying and running the system for an extended period or be indicated by experts. These outcomes are less likely than the desired and intentional (nominal) outcome, which is the middle cell in the 3×3 grid. Since these outcomes can occur, we should be aware of them when modeling the system, especially when dealing with safety. However, as the alternative outcomes only occur sporadically, we can not determine accurate probabilistic values to use during planning.[2] For the productive, goal-oriented aspect of planning, we do not take into account alternative outcomes. The constraint solver will assume that the intended effects occur. If, during execution, the system were to arrive in any unintentional state (possibly due to an alternative effect), the completeness of the policy will make sure that the operation can continue to reach the goal. If from such a state, and by extension from *any* state, the goal can not be reached no policy can be found. The planner will then provide such states as feedback to the developer.

Note on non-determinism The presented approach, using nominal and alternative effects, can also be used to model true non-determinism in the system. We illustrate this with the example of a coin flipped at run-time. Two actions (one for each side of the coin) are modeled to eliminate the inherent bias toward the nominal outcome. For each action, the nominal effects take care of one side of the coin while the alternative effects take care of the other side. Regardless of the outcome of an individual toss, the policy ensures that the executed actions are safe, as we define next.

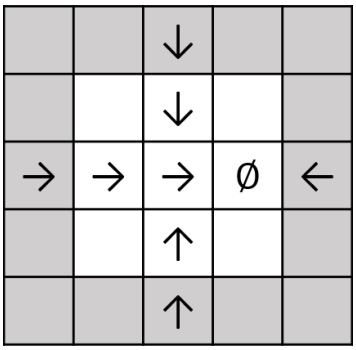
4.2 Safety

We extend the previous grid example to a 5×5 grid. In this example, the outer cells are considered unsafe. These cells could be located next to stairs which the agent could fall off, or a wall it could run into. Although simply defining unsafe states or conditions is a very straightforward way to define (un)safety,

[2] If one is able to determine the probabilistic values for the effects accurately, using an MDP would be a better choice to obtain the policy.



(a) Visualization of the three reachable states by *move_right* from S. A second *move_right* (not visualized) in either *x* or *y* could lead to unsafe (gray) states.



(b) A guaranteed safe policy. This policy should **not** be considered complete unless it is accompanied by the necessary assumptions.

Figure 3: Illustrative 5×5 grid example

it yields a powerful approach. It provides a large improvement over manually identifying which state-action combinations are unsafe, as the original Tumato planner requires using reaction rules.

A *move_right* at the starting point *S* will never lead to an unsafe state, as shown in Figure 3a. This action will be allowed to be used during planning. In turn, in both of the unintentionally reachable states (separately denoted as *x* and *y* in the figure), a similar *move_right* will not be allowed, as alternative outcomes could lead the agent into an unsafe state. A safe policy is shown in Figure 3b. Note that many cells remain empty. Those cells do not allow any safe actions in our current example. To obtain this policy, the specification would explicitly have to contain a number of assumptions to exclude those cells from the valid state space. If those assumptions are not added, no result will be obtained since no complete policy exists. In this paper, we will not look further into assumptions or how they are modeled.

Note on restoring safety The approach mentioned above does not distinguish between starting from a safe or unsafe state: the next state is guaranteed to be safe. Hence, for unsafe states, a solution will only be found when safety can be restored within one step. In addition, when multiple actions are executed, the planner is aware that only the effects of *one* of those actions will take place *first*. As a result, every possible outcome of every action selected for execution must be safe. Furthermore, since actions can be assigned a cost value (see Section 5.1), the planner can choose the set of actions that restores safety in the most preferred way.

5 Specification

In this section, we elaborate on the modifications made to the specification introduced in Section 2. Our contribution lies in the inclusion of alternative effects and costs for actions, as well as the addition of state rules. We use the AMR described in Section 3 as an example for constructing a formal specification. For the complete specification, formatted as supported by Tumato, please refer to Appendix A.

5.1 Actions

Unlike in the original Tumato framework, we define two kinds of effects. The *nominal effects* represent the desired and intended outcome of an action, whereas the *alternative effects* correspond to unintended outcomes that *could* occur instead of the nominal one, but are less likely to. To provide a comprehensive

understanding of both kinds of effects, we present a two-fold perspective. In the context of productive (goal-oriented) planning, only the nominal effects are considered, as mentioned in Section 4.1. Undesired outcomes are deliberately excluded. However, when ensuring safe behavior, *all* effects should be considered. Each possible outcome of every executed action must be safe. This two-fold perspective enables the appreciation of both kinds of effects within their respective formal contexts.

Further, we extend the specifications of actions with a generic cost. In the AMR example, the notion of *duration*[3] is used. If no value is specified, the planner will assume a value of *0*. The planner only considers these values when choosing the best solution to restore safety. For states where safety should only be maintained rather than restored, these values are ignored and can be all considered *0*.

Next, a few examples of actions of the AMR are given to provide further clarification.

The action **stop_conveyor**:

- has a duration of *1*,
- controls one resource: *conveyor*,
- has one precondition: $S_{Conveyor} = on$,
- has the nominal effect: $S_{Conveyor} = off$,
- has no alternative effects.

Since stopping the conveyor happens virtually instantly, the action gets assigned an arbitrarily low duration value. Since there are no alternative effects specified, we assume that the action will never fail.

The action **move_to_workstation_1**:

- has a duration of *10*,
- controls one resource: *motors*,
- has one precondition: $S_{Location} = corridor$,
- has the nominal effect: $S_{Location} = workstation\_1$,
- has one set of alternative effects: $S_{Location} = corridor$.

For moving to a specific location, we assign a relative duration of *10*. The nominal effect is as expected, reaching *workstation_1*, and the alternative effect is expressed as the AMR not reaching the workstation. In practice, this alternative effect can occur when this specific workstation is blocked or occupied. This possibility is workstation-specific, and the effects can be modeled differently for every action.

The action **receive_workpiece**:

- has a duration of *3*,
- controls the resources: *conveyor* and *motors*,
- has the preconditions: $S_{Location} = workstation\_1$ and $S_{Load} = free$,
- has the nominal effects: $S_{Conveyor} = on$ and $S_{Load} = loaded$,
- has two sets of alternative effects:
  $S_{Conveyor} = on$ and $S_{Load} = free$, and
  $S_{Conveyor} = off$ and $S_{Load} = free$.

[3]Please note that *time* is not considered explicitly. The durations are merely used to find the most preferred (fastest) way to restore safety.

A relative duration of *3* is assigned to load transfer actions. During the transfer of a workpiece, the motors are used to hold the AMR in place. In this example, workpieces can only be obtained at *workstation_1*, hence the corresponding precondition. For this action, two sets of alternative effects are considered. The first one represents the outcome where a (for example, oddly shaped) workpiece gets stuck. The second one represents an even worse scenario where the entire conveyor gets blocked by receiving a (for example, too heavy) workpiece.

These three examples show how actions and their effects can be modeled. If only one effect is given (the nominal effect), the outcome of the action is considered deterministic. Alternatively, one or more alternative outcomes can be specified. Recall that the alternative outcomes are not taken into account for the productive aspect of planning. We do not (want to) rely on the alternative effects to reach a particular goal. They are, however, considered when guaranteeing safety.

5.2 Safety Rules

Safety rules are used to add constraints to the behavior of the system to guarantee safe behavior. In this section, we delve deeper into the use of reaction rules and introduce the new state rules.

5.2.1 Reaction Rules

As explained in Section 2, reaction rules are used to specify reactive behavior. If a condition holds on the current state, an action is constrained to be executed or not. We specified that if the AMR is present in the corridor with the conveyor on, the conveyor should be stopped:

$$(S_{Location} = corridor \wedge S_{Conveyor} = on) \Rightarrow Exec(\textbf{stop\_conveyor}).$$

However, we would prefer to guarantee that the conveyor is never on when the AMR is in the corridor (or at the charger[4]) in the first place. Such behavior could lead to dropping workpieces and other dangerous situations for human agents in the factory. Given our knowledge of the system, we specify:

$$(S_{Location} = corridor \vee S_{Location} = charger) \wedge (S_{Conveyor} = off)$$

$$\Rightarrow \neg Exec(\textbf{receive\_workpiece}) \wedge \neg Exec(\textbf{deliver\_workpiece}).$$

This rule has to be updated every time a new action is introduced that could turn on the conveyor. We also have to specify:

$$(S_{Conveyor} = on) \Rightarrow \neg Exec(\textbf{move\_to\_\{x\}})$$

for every location x for which moving toward x could result in arriving at the corridor or charger. This last expression can require a large number of reaction rules to be modeled, depending on the number of locations and how the effects of different move-actions connect them. To considerably reduce the number of safety rules that have to be specified and hence, to make the specification less prone to errors, we introduce the concept of *state rules* next.

[4]The previous reaction rule becomes: $(S_{Location} = corridor \vee S_{Location} = charger) \wedge (S_{Conveyor} = on) \Rightarrow Exec(\textbf{stop\_conveyor})$.

5.2.2 State Rules

This new kind of safety rule enables specifying constraints on reachable states rather than on the actions to execute. We can simplify the previous example to one state rule:

$$(S_{Location} = corridor \vee S_{Location} = charger) \Rightarrow (S_{Conveyor} = off).$$

We introduce state rules in the form of *desired safety conditions*. The specified conditions should, according to the system's ability, always remain *True*. Alternatively, one can specify the unsafe conditions, which should be kept *False*. Translating between the two corresponds to negating the conditions.

The effects of an executed action should never lead to the violation of a state rule. Formally, a state rule expresses a condition that must hold after executing any (set of) action(s) that the policy instructs for a state, regardless of which action effectuates first and regardless of which (nominal or alternative) effects occur. Whether an action a is allowed in a state s can more formally be expressed as follows:

$$allowed(a,s) \Leftrightarrow \forall effect \in effect(a,s) : effect \Rightarrow \{state\_rules\}$$

where $effect(a,s)$ represents all possible effects ($nominal \cup alternative$) of action a in state s and $effect \Rightarrow \{state\_rules\}$ denotes that the effect does adhere to all conditions described by the state rules.

If the current state was to violate a state rule, the next planned (set of) action(s) will always clear that violation (see Section 4.2). Since the state rules do not allow any actions that could knowingly lead to undesired states, the system must have reached that state under the influence of an external force. Further, if during planning for some state no instructions to restore safety exist, the planner notifies the user, indicating for which state no solution can be found. When multiple such instructions are available, the planner selects that set of instructions that restores safety in the most preferred way. For this example, we assigned durations to actions, hinting at the intention of restoring safety as quickly as possible. Alternative approaches are to use costs and find the cheapest solution or to deal with risk explicitly.

6 Discussion

In this section, we first discuss the modifications made to the set of constraints solved during planning. We also present the findings from a preliminary experiment and analyze the resulting policy. Finally, we discuss the challenges and potential limitations of the approach, hinting toward possible future work.

6.1 Constraint-Based Planning

The constraint-based approach providing the basis for this work has been introduced in the original work on Tumato [5]. The constraints that differ are the ones related to the new state rules. While reaction rules could be incorporated directly into the constraint satisfaction problem as a constraint prohibiting or enforcing an action to be executed in a certain state, state rules require more insight. For every execution step, the condition of the state rule is applied as a constraint to each state in the set of *possible next states*. State rules constrain every possible outcome of an action in the current state rather than the actions themselves. This approach results in a number of new constraints, one for every possible outcome, enforcing a state rule conditionally to whether the action gets executed. As a result, if an action *could* violate one or more state rules, this action will be prohibited in the given state. Practically, these constraints replace a (potentially large) number of constraints specified by reaction rules, as illustrated in

Section 5.2.1. Complementary, state rules can *enforce* the execution of actions if their effects are required to maintain safety. Even if the current state is not safe, the next state is guaranteed to be safe, and the planner will minimize the required duration or cost to restore safety using a minimization objective. Since the use of objectives is solver-specific, we refrain from elaborating further and only illustrate their potential use. Finally, when no solution exists, the solver can refer to states for which no behavior can be generated to *explain* why the specification is unrealizable, identical to the original feature of Tumato.

6.2 Preliminary Experimental Results

The example model described throughout this paper contains 32 states and 8 actions, and the planner takes between 2 and 3 seconds to obtain a policy on a *1.6 GHz Dual-Core Intel Core i5*. No significant memory usage was detected. The initial CSP starts off with 518 constraints and ramps up to 5266 (as in Tumato, constraints are added automatically to obtain conflict-free plans to define a policy). The number of decision variables starts at 823 and reaches up to 8487. These numbers are slightly higher compared to using a reaction rule based approach but within the same order of magnitude. For a scalability comparison, a somewhat larger model with 2688 valid states and 18 actions takes about 75 seconds, while memory usage remains negligible. The CSP starts with 1130 constraints and ramps up to 5545. The number of decision variables starts at 1787 and reaches up to 8911.

Practically, all the individual reaction rules have been replaced by the constraints generated from the state rule. However, reaction rules require manual (more error-prone) implementation, and the state rules might cover situations the user did not anticipate. Especially for more complex systems, this approach can be beneficial. Finally, albeit more technical, we want to mention the impact on restoring safety *the most preferred way* when the maximum planning length or number of possible unsafe states grows. Now, the *best* solution has to be found, rather than *any* solution, as otherwise was sufficient. Performance worsens because of the increasing number and complexity of the minimization objectives.

6.3 The Policy

Since the generated policies are complete, every valid state of the system has a corresponding entry in the policy. In this section, a few well-chosen entries from the AMR's policy are presented.[5]

```
"S_Location": "corridor",
"S_Battery": "ok",
"S_Load": "free",
"S_Conveyor": "off",
"Actions": ["move_to_workstation_1"]
```

```
"S_Location": "corridor",
"S_Battery": "low",
"S_Load": "free",
"S_Conveyor": "off",
"Actions": ["move_to_charger"]
```

The first entry (left) instructs the AMR to move to workstation_1, as it is currently not carrying a workpiece and there is sufficient battery power left. The desired productive behavior appears. In the second entry (right), as the battery level holds the value *low*, a different (prioritized) conditional goal is active. The AMR now moves to the charger. In neither entry the state rule condition (specified in Section 5.2.2) is violated, neither in the current state nor any foreseeable possible next state.

```
"S_Location": "workstation_2",
"S_Battery": "ok",
"S_Load": "free",
"S_Conveyor": "on",
"Actions": ["stop_conveyor"]
```

```
"S_Location": "corridor",
"S_Battery": "low",
"S_Load": "free",
"S_Conveyor": "on",
"Actions": ["stop_conveyor"]
```

[5] A short clip of the AMR executing a policy generated by Tumato can be found *online*.

In the next entry (left), the specified state rule is more prominent. Without the state rule, the planner would instruct the AMR to start moving to *workstation_1*. Since the conveyor will be active later on, turning it off now would be redundant. It is clear that the state rule leads to executing *stop_conveyor*. A naive implementation of state rules (or use of reaction rules) would allow simultaneous movement and stopping of the conveyor. However, the adapted Tumato solver recognizes the non-deterministic order in which actions effectuate when multiple actions are executed simultaneously and ensures safety by permitting only the *stop_conveyor* action. In the final presented entry (right), the current state violates the state rule. To restore safety, the AMR is instructed to execute *stop_conveyor*. Although the more productive action *move_to_charger* would also restore safety (as the only foreseeable next state is safe), the planner chooses the most preferred way to restore safety based on the durations of actions.

6.4 Overview and Future Work

Tumato succeeds in connecting *theoretic* agents-based research, specifically on guaranteeing safe behavior, to the field of *robotics*, where non-determinism is inevitable. By including durations and alternative outcomes, we achieve a concise and expressive behavior specification. State rules further enhance expressiveness and result in a stronger as well as less error-prone safety specification. One main limitation of the approach is the use of discrete state variables. For practical applications, a monitoring module has to be present to map the continuous world into a discrete state space.

Despite the promising combination of uncertainty and safety with the constraint-based planning approach, challenges remain open for further investigation. The state rules currently enforce that any foreseeable outcome of the executed actions is safe. When multiple sets of actions exist to *restore* safety, the planner is capable of choosing the best one with regard to a value such as duration or cost. When no actions are available to restore safety immediately, the planner will notify the user, indicating for which state safety can not be restored. Although this is a desirable approach, future research could explore how allowing multiple successive (sets of) actions in unsafe states could be required to restore safety. Furthermore, in practice, different safety constraints relate to different severities. In the same way that actions can be assigned a specific value, the safety rules could be weighted to enable the constraint solver to find the overall best solution. Finally, an empirical study will be conducted concerning the practical use of the adapted planner. This study should consider the actual safety guarantees that are obtained as well as the specification effectiveness and comfort that is achieved. Comparisons with the original version of Tumato, as well as with more traditional approaches, are in order.

7 Conclusion

This paper presents an approach to specify and generate safe and robust robot behavior. For this purpose, we extend the existing constraint-based planning tool Tumato with the notion of uncertainty and state rules. Robustness against uncertainty is achieved by extending actions with alternative, less desired and less likely, but foreseeable outcomes. Further, state rules form a powerful approach for expressing safety rules based on state conditions rather than manually having to specify all situations that could lead to such unsafe states. This new approach requires an order of magnitude fewer rules to be specified and hence is less prone to errors. Tumato translates the declarative specification, including the defined safety rules, automatically into a set of Constraint Satisfaction Problems. Solving the CSPs yields an execution policy that inherently satisfies all the specified rules. This approach enables the detection of unrealizable specifications early on. The obtained policy is sound and complete by construction.

Acknowledgements This research is partially funded by the Research Fund KU Leuven.

References

[1] Alberto Camacho, Eleni Triantafillou, Christian J Muise, Jorge A Baier & Sheila A McIlraith (2016): *Non-Deterministic Planning with Temporally Extended Goals: Completing the Story for Finite and Infinite LTL*. In: *Proceedings of the Workshop on Knowledge-based Techniques for Problem Solving and Reasoning co-located with 25th International Joint Conference on Artificial Intelligence (IJCAI 2016), New York City, USA, July 10, 2016*. Available at `https://ceur-ws.org/Vol-1648/paper10.pdf`.

[2] Gerard Canal, Michael Cashmore, Senka Krivić, Guillem Alenyà, Daniele Magazzeni & Carme Torras (2019): *Probabilistic planning for robotics with ROSPlan*. In: *Towards Autonomous Robotic Systems: 20th Annual Conference, TAROS 2019, London, UK, July 3–5, 2019, Proceedings, Part I 20*, Springer, pp. 236–250, doi:10.1007/978-3-030-23807-0_20.

[3] Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carrera, Narcis Palomeras, Natalia Hurtos & Marc Carreras (2015): *Rosplan: Planning in the robot operating system*. In: *Proceedings of the international conference on automated planning and scheduling*, 25, pp. 333–341, doi:10.1609/icaps.v25i1.13699.

[4] Rina Dechter, Itay Meiri & Judea Pearl (1991): *Temporal constraint networks*. *Artificial intelligence* 49(1-3), pp. 61–95, doi:10.1016/0004-3702(91)90006-6.

[5] Hoang Tung Dinh, Mario Henrique Cruz Torres & Tom Holvoet (2017): *Sound and complete reactive UAV behavior using constraint programming*. Available at `https://lirias.kuleuven.be/retrieve/470086`.

[6] Patrick Doherty, Joakim Gustafsson, Lars Karlsson & Jonas Kvarnström (1998): *Tal: Temporal action logics language specification and tutorial*. *Electronic Transactions on Artificial Intelligence*. Available at `http://www.ep.liu.se/ej/etai/1998/009/`.

[7] Patrick Doherty & Jonas Kvarnström (2001): *TALplanner: A temporal logic-based planner*. *AI Magazine* 22(3), pp. 95–95, doi:10.1609/aimag.v22i3.1581.

[8] E Allen Emerson (1990): *Temporal and modal logic*. In: *Formal Models and Semantics*, Elsevier, pp. 995–1072, doi:10.1016/B978-0-444-88074-1.50021-4.

[9] Maksym Figat, Cezary Zieliński & René Hexel (2017): *FSM based specification of robot control system activities*. In: *2017 11th International Workshop on Robot Motion and Control (RoMoCo)*, IEEE, pp. 193–198, doi:10.1109/RoMoCo.2017.8003912.

[10] Tomas Geffner & Hector Geffner (2018): *Compact policies for fully observable non-deterministic planning as SAT*. In: *Proceedings of the International Conference on Automated Planning and Scheduling*, 28, pp. 88–96, doi:10.1609/icaps.v28i1.13880.

[11] Alfonso E Gerevini, Patrik Haslum, Derek Long, Alessandro Saetti & Yannis Dimopoulos (2009): *Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners*. *Artificial Intelligence* 173(5-6), pp. 619–668, doi:10.1016/j.artint.2008.10.012.

[12] Malik Ghallab, Dana Nau & Paolo Traverso (2016): *Automated Planning and Acting*, 1st edition. Cambridge University Press, USA, doi:10.1017/CBO9781139583923.

[13] Erez Karpas & Daniele Magazzeni (2020): *Automated planning for robotics*. *Annual Review of Control, Robotics, and Autonomous Systems* 3, pp. 417–439, doi:10.1146/annurev-control-082619-100135.

[14] Spyros Maniatopoulos, Philipp Schillinger, Vitchyr Pong, David C Conner & Hadas Kress-Gazit (2016): *Reactive high-level behavior synthesis for an atlas humanoid robot*. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, pp. 4192–4199, doi:10.1109/ICRA.2016.7487613.

[15] Hai Nguyen, Matei Ciocarlie, Kaijen Hsiao & Charles C Kemp (2013): *Ros commander (rosco): Behavior creation for home robots*. In: *2013 IEEE International Conference on Robotics and Automation*, IEEE, pp. 467–474, doi:10.1109/ICRA.2013.6630616.

[16] Charles Prud'homme & Jean-Guillaume Fages (2022): *Choco-solver: A Java library for constraint programming*. *Journal of Open Source Software* 7(78), p. 4708, doi:10.21105/joss.04708. Available at `https://choco-solver.org`.

[17] Martin L Puterman (2014): *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, doi:10.1002/9780470316887.

[18] Edward Tsang (1993): *Foundations of constraint satisfaction*. Academic Press Limited.

[19] Jan Vermaelen, Hoang Tung Dinh & Tom Holvoet (2020): *A survey on probabilistic planning and temporal scheduling with safety guarantees*. In: *ICAPS Workshop on Planning and Robotics*. Available at `https://icaps20subpages.icaps-conference.org/wp-content/uploads/2020/10/12-PlanRob_2020_paper_12.pdf`.

[20] Tichakorn Wongpiromsarn, Ufuk Topcu & Richard M Murray (2013): *Synthesis of control protocols for autonomous systems*. *Unmanned Systems* 1(01), pp. 21–39, doi:10.1142/S2301385013500027.

A The Specification

In this appendix, we use the AMR example to illustrate the extended format of specification supported by Tumato.

```
BEGIN STATE VECTOR
state S_Location can be corridor, charger, workstation_1, workstation_2
state S_Battery can be low, ok
state S_Load can be loaded, free
state S_Conveyor can be on, off
END STATE VECTOR

BEGIN RESOURCES
resource MOTORS
resource CONVEYOR
END RESOURCES

BEGIN ACTIONS
action move_to_workstation_1
duration: 10
controlled resources: MOTORS
preconditions: S_Location is corridor
nominal effects: S_Location is workstation_1
alternative effects: S_Location is corridor

action move_to_workstation_2
duration: 10
controlled resources: MOTORS
preconditions: S_Location is corridor
nominal effects: S_Location is workstation_2

action move_to_charger
duration: 10
controlled resources: MOTORS
preconditions: S_Location is corridor
nominal effects: S_Location is charger
```

```
action move_to_corridor
duration: 2
controlled resources: MOTORS
preconditions: NOT S_Location is corridor
nominal effects: S_Location is corridor

action receive_workpiece
duration: 3
controlled resources: MOTORS, CONVEYOR
preconditions: S_Location is workstation_1, S_Load is free
nominal effects: S_Conveyor is on, S_Load is loaded
alternative effects: S_Conveyor is on, S_Load is free
alternative effects: S_Conveyor is off, S_Load is free

action deliver_workpiece
duration: 3
controlled resources: MOTORS, CONVEYOR
preconditions: S_Location is workstation_2, S_Load is loaded
nominal effects: S_Conveyor is on, S_Load is free
alternative effects: S_Conveyor is on, S_Load is loaded

action stop_conveyor
duration: 1
controlled resources: CONVEYOR
preconditions: S_Conveyor is on
nominal effects: S_Conveyor is off

action charge
duration: 50
controlled resources: MOTORS
preconditions: S_Location is charger
nominal effects: S_Battery is ok
END ACTIONS

BEGIN REACTION RULES // Please note, comments start with "//".
//rule: IF (S_Location is corridor OR S_Location is charger) AND S_Conveyor is on
//  THEN executing stop_conveyor
//rule: IF (S_Location is corridor OR S_Location is charger) AND S_Conveyor is off
//  THEN NOT executing receive_workpiece AND NOT executing deliver_workpiece
//   //AND NOT any future action that could turn the conveyor on
//rule: IF S_Conveyor is on THEN NOT executing move_to_corridor
//  AND NOT executing move_to_charger AND NOT executing move_to_workstation_1
//   //AND NOT any future action that could move the
//   //AMR away from NOT(corridor OR charger).
//And probably more rules
END REACTION RULES

BEGIN STATE RULES
rule: IF S_Location is corridor OR S_Location is charger THEN S_Conveyor is off
END STATE RULES

BEGIN GOALS
goal type: priority
when S_Battery is low then goal: S_Battery is ok
when S_Load is loaded then goal: S_Load is free
when S_Load is free then goal: S_Load is loaded
```

```
END GOALS

BEGIN CONFIG
max_plan_length: 5
END CONFIG
```

Reasoning about Intuitionistic Computation Tree Logic

Davide Catta
Università di Napoli "Federico II"
Naples, Italy

Vadim Malvone
Télécom Paris
Palaiseau, France

Aniello Murano
Università di Napoli "Federico II"
Naples, Italy

In this paper, we define an intuitionistic version of Computation Tree Logic. After explaining the semantic features of intuitionistic logic, we examine how these characteristics can be interesting for formal verification purposes. Subsequently, we define the syntax and semantics of our intuitionistic version of CTL and study some simple properties of the so obtained logic. We conclude by demonstrating that some fixed-point axioms of CTL are not valid in the intuitionistic version of CTL we have defined.

1 Introduction

Classical modal and temporal logics are extensions of classical logic, in which some new operators (usually called modalities) qualify the truth of classical formulae. For instance, in a classical or temporal modal logic, one can express that a certain formula is *necessarily* true, *possibly* true, that *it will be true* in some future moment of time and so on. In particular, temporal logics are a family of modal logics in which the modalities permit to express, as the name suggests, temporal properties of formulae. Temporal logics originated in the philosophical work of Prior [11] in the 50s and were rediscovered and adapted by Pnueli [10] who defined the Linear Temporal Logic (LTL) and showed how interesting program properties could be expressed using temporal logic and, more importantly, automatically verified on mathematical models of the executions of such programs. Time flow in LTL is linear, meaning that each instant of time has exactly one successor. In 1981, Edmund M. Clarke and E. Allen Emerson first introduced Computation Tree Logic (CTL) [4]. CTL is a type of branching-time logic, where time is represented as a tree-like structure with an undetermined future. This logic was first used to reason about abstractions of concurrent programs and subsequently became a milestone in the automatic verification of cyber-physical models.

Intuitionistic Logic. Intuitionism is a mathematical school developed in the early 1900s by the Dutch mathematician L.E.J. Brouwer. Intuitionism rejects the idea that the truth value of a mathematical statement is independent of our ability to *know or verify* it. Put differently: an intuitionist believes that the truth conditions of a mathematical statement are its provability conditions. As a result, intuitionism rejects the validity of the principle of excluded middle. In fact, given any mathematical statement φ, it is not always possible to have knowledge of (i.e., prove or verify) φ or its negation $\neg\varphi$. In the 1930s, Heyting developed intuitionistic logic [7], a logic embodying the underlying principles of intuitionistic reasoning. Intuitionistic logic has found many applications in computer science, e.g., the Curry-Howard isomorphism relating intuitionistic proofs and typed lambda terms [13] and the constructive type theory of Per Martin-Löf [8]. The semantics of intuitionistic logic was first specified by means of topological spaces, and later, by Saul Kripke in terms of Kripke models [6]. This latter semantics is particularly interesting for our means. Kripke's idea is that a model of intuitionistic logic represents the dynamics

A. Ferrando, R.C. Cardoso (Eds.): Third Workshop on
Agents and Robots for reliable Engineered Autonomy 2023 (AREA'23)
EPTCS 391, 2023, pp. 42–48, doi:10.4204/EPTCS.391.6

of an *idealized agent* (or idealized mathematician) that is expanding her knowledge about mathematical statements over time. In this temporal process, she creates new elements while observing the fundamental facts in her universe. Moving from one moment to the next, she freely decides how to continue her activity, resulting in a partially ordered set of possible stages, known as possible worlds. In this particular interpretation, the truth of an intuitionistic formula φ in a given moment of time w depends upon the future of w, i.e, upon the moments of time coming after w. For instance, to conclude that a formula $\varphi \rightarrow \psi$ is true at a given moment w, the agent must be certain that for every future moment w', if there is a proof of φ at w', it is always possible to obtain a proof of ψ as well.

Intuitionistic Modal Logic. Intuitionistic logic can be extended with modalities in different ways (for an overview see [12]): while in classical logic axioms involving only $\Box$ provide also description of the behavior of $\Diamond$, for intuitionistic logic this is no more the case since the duality of the two modalities does not hold anymore. This leads to different approaches. *Constructive modal logics* consider minimal sets of axioms to guarantee the definition of the behaviors of the $\Box$ and $\Diamond$ modalities. A second approach, referred to as *intuitionistic modal logic*, considers additional axioms in order to validate the Gödel-Gentzen translation [3]. This second approach has led to the definition of a class of Kripke models (called birelational models) in which two distinct relations of accessibility are considered: one representing the aforementioned preorder and the other representing the "standard" accessibility relation of a Kripke model. In this paper, for the sake of simplicity, we will follow this second approach. This approach is used, for instance, for the intuitionistic version of LTL studied in [2].

Intuitionistic Computational Tree Logic. In this paper, we aim to define an intuitionistic version of the aforementioned Computation Tree Logic. The intuition we seek to formalize is as follows: we distinguish between two different temporal evolutions within a CTL model. One represents the agent's knowledge about the system, while the other represents the possible evolutions of the system itself based on the given knowledge. The agent's goal is to conclusively verify that certain properties hold with respect to the possible evolutions of the model and the potential evolution of its knowledge. In other words, given a property of interest φ, it wants to ascertain that, in any state of its knowledge regarding the model, and regardless of the evolution of the model itself, φ is satisfied. We think this type of intuition allows for considering the verification of CTL properties in the imperfect information context.

Structure of the work In Section 2 we provide the syntax and semantic of Intuitionistic Computation Tree Logic. Then, in Section 3 we provide some properties of our logic. Finally, we conclude in Section 4 with some future directions.

2 Syntax and Semantics of Intuitionistic Computation Tree Logic

In this section we provide the syntax and semantics of our logic. We follow [9] for the definition of models that will be used in the following. We fix a countable set $\mathcal{P}$ of *atomic propositions* or *atoms*.

Definition 1. *Formulae of Intuitionistic Computation Tree Logic (ICTL for short) are defined by the following grammar:*

$$\varphi, \psi := p \mid \bot \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \varphi \rightarrow \psi$$

$$\mathsf{E}\,\mathsf{X}\varphi \mid \mathsf{E}(\varphi \mathsf{U} \psi) \mid \mathsf{E}(\varphi \mathsf{R} \psi) \mid \mathsf{A}\,\mathsf{X}\varphi \mid \mathsf{A}(\varphi \mathsf{U} \psi) \mid \mathsf{A}(\varphi \mathsf{R} \psi)$$

where $p \in \mathcal{P}$ and $\bot$ is the absurdity symbol. We define the negation of a formula φ as $\neg\varphi \equiv \varphi \to \bot$. Formulae whose first operator is E *are called* existential *formulae, while those whose first operator is* A *are called* universal *formulae.*

Given the syntax of ICTL, we can now provide the definition of birelational frame.

Definition 2. *A* ***birelational frame*** *$\mathfrak{F}$ is a triple $\langle W, P, R\rangle$ where W is a non-empty countable set of worlds, P is a preorder on W (i.e., a reflexive and transitive relation) and R is a binary serial relation on W (i.e., for every $x \in W$ there is a $y \in W$ such that xRy) in which the following conditions are satisfied, for all $x, y, z \in W$:*

(C_1) *if xRy and yPz, then there is a $u \in W$ such that xPu and uRz (see below left);*

(C_2) *if xPz and xRy, then there is a $u \in W$ such that yPu and zRu (see below right).*

A *path* in a frame $\mathfrak{F}$ is an infinite sequence of worlds $\rho = \rho_0, \rho_1, \ldots$ such that for any $i \in \mathbb{N}$ we have that $\rho_i R \rho_{i+1}$. If ρ is a path then ρ_i denotes its $(i+1)$-th element, $\rho_{\leq i}$ the finite prefix $\rho_0, \ldots, \rho_i$ of ρ and $\rho_{\geq i}$ the infinite suffix $\rho_i, \rho_{i+1}, \ldots$ of ρ starting at ρ_i.

Lemma 1. *Let $\mathfrak{F}$ be a frame and w and w' two worlds of $\mathfrak{F}$ such that wPw'. For every path ρ such that $\rho_0 = w$ there is a path τ such that $\tau_0 = w'$ for which holds that $\rho_i P \tau_i$ for every natural number i.*

Proof. Let $A = (w_i)_{i \in \mathbb{N}}$ be an enumeration of the worlds of $\mathfrak{F}$. Given ρ, we define τ by induction on $\mathbb{N}$. We define $\tau_0 = w'$ and for any $i \geq 1$ we let τ_i be the smallest element in A such that $\tau_{i-1} R \tau_i$ and $\tau_i P \rho_i$. Remark that τ_i exists because of condition C_2 above: in fact, suppose that for $j < i$ it holds that $\rho_j P \tau_j$, in particular, this means that $\tau_{i-1} P \rho_{i-1}$. Since $\rho_{i-1} R \rho_i$, then by C_2 there is (at least one and possibly an infinite countable number of) $u \in W$ such that $\tau_i R u$ and $\rho_i P u$. □

Given the definition of frames, we are able to define our models.

Definition 3. *A* ***birelational model*** *(model from now on) is a tuple $\mathfrak{M} = \langle W, P, R, \mathcal{V}\rangle$ where $\langle W, P, R\rangle$ is a birelational frame and $\mathcal{V} : W \to 2^{\mathcal{P}}$ is a* ***valuation function*** *sending each world w to the subset of atomic propositions that are true at w and that is subject to the* ***monotonicity condition****, that is: if wPw' then $\mathcal{V}(w) \subseteq \mathcal{V}(w')$.*

Now, we have all the ingredients to define the semantics of ICTL.

Definition 4. *The satisfaction relation $\mathfrak{M}, w \models \varphi$ between a model $\mathfrak{M}$, a world w of $\mathfrak{M}$, and an ICTL formula φ is inductively defined as follows:*

- $\mathfrak{M}, w \models p$ *iff* $p \in \mathcal{V}(w)$;
- $\mathfrak{M}, w \models \bot$ *never;*
- $\mathfrak{M}, w \models \psi \wedge \theta$ *iff* $\mathfrak{M}, w \models \psi$ *and* $\mathfrak{M}, w \models \theta$;
- $\mathfrak{M}, w \models \psi \vee \theta$ *iff* $\mathfrak{M}, w \models \psi$ *or* $\mathfrak{M}, w \models \theta$;
- $\mathfrak{M}, w \models \psi \to \theta$ *iff for every w' such that wPw' we have that* $\mathfrak{M}, w' \models \psi$ *implies* $\mathfrak{M}, w' \models \theta$;
- $\mathfrak{M}, w \models \mathsf{E}\,\mathsf{X}\psi$ *iff there is a path ρ whose first element ρ_0 is w and* $\mathfrak{M}, \rho_1 \models \varphi$;

- $\mathfrak{M}, w \models \mathsf{E}(\psi \mathsf{U} \theta)$ *iff there is a path* ρ *whose first element* ρ_0 *is* w *and there is a* $j \geq 0$ *such that* $\mathfrak{M}, \rho_j \models \theta$ *and for all* $0 \leq i < j$ *we have that* $\mathfrak{M}, \rho_i \models \psi$*;*
- $\mathfrak{M}, w \models \mathsf{E}(\psi \mathsf{R} \theta)$ *iff there is a path* ρ *whose first element* ρ_0 *is* w *and either* $\mathfrak{M}, \rho_i \models \theta$ *for all* $i \in \mathbb{N}$ *or there is a* $j \geq 0$ *such that* $\mathfrak{M}, \rho_j \models \psi$ *and* $\mathfrak{M}, \rho_i \models \theta$ *for all* $0 \leq i \leq j$*;*
- $\mathfrak{M}, w \models \mathsf{A}\mathsf{X}\psi$ *iff for every* w' *such that* wPw' *and for every path* ρ *whose first element* ρ_0 *is* w' *we have that* $\mathfrak{M}, \rho_1 \models \psi$*;*
- $\mathfrak{M}, w \models \mathsf{A}(\psi \mathsf{U} \theta)$ *iff for every* w' *such that* wPw' *and for every path* ρ *whose first element* ρ_0 *is* w' *we have that there is a* $j \geq 0$ *such that* $\mathfrak{M}, \rho_j \models \theta$ *and for all* $0 \leq i < j$ *we have that* $\mathfrak{M}, \rho_i \models \psi$*;*
- $\mathfrak{M}, w \models \mathsf{A}(\psi \mathsf{R} \theta)$ *iff for every* w' *such that* wPw' *and for every path* ρ *whose first element* ρ_0 *is* w' *we have that either* $\mathfrak{M}, \rho_i \models \theta$ *for all* $i \in \mathbb{N}$ *or there is a* $j \geq 0$ *such that* $\mathfrak{M}, \rho_j \models \psi$ *and* $\mathfrak{M}, \rho_i \models \theta$ *for all* $0 \leq i \leq j$*.*

We write $\mathfrak{M}, w \not\models \varphi$ *when* w *does not satisfy* φ*. We say that a formula* φ *is* ***valid in a model*** $\mathfrak{M}$ *iff it is satisfied in every* $w \in W$*. A formula* φ *is* ***valid in a frame*** $\mathfrak{F} = \langle W, P, R \rangle$ *iff it is valid in* $\langle W, P, R, \mathcal{V} \rangle$ *for any valuation* $\mathcal{V}$*. A formula* φ *is* ***valid*** *iff it is valid in every frame.*

Remark that, by the above definition, a formula $\neg\varphi = \varphi \rightarrow \bot$ is satisfied at w if for any w' such that wPw' we have that w' satisfies φ implies w' satisfies $\bot$. Since $\bot$ is *never* satisfied, this is equivalent to say that *no state* w' bigger than w satisfies φ. Given the above definition of satisfaction and validity, it is fairly easy to show that the operators are not dual, e.g., we do not have that $\neg\mathsf{A}\mathsf{X}\neg\varphi \rightarrow \mathsf{E}\mathsf{X}\varphi$.

3 Main Properties

In this section, we show that CTL and ICTL differs in some important properties. First, we show that the satisfaction relation is monotonous with respect to the preorder. This properties, that is shared by all intuitionist modal logics, intuitively says that the agent's knowledge can only grow.

Proposition 1. *Let* $\mathfrak{M}$ *be a model,* φ *a formula, and* w *and* w' *a pair of worlds of* $\mathfrak{M}$*. if* $\mathfrak{M}, w \models \varphi$ *and* wPw' *then* $\mathfrak{M}, w' \models \varphi$

Proof. By induction on the structure of φ using Lemma 1 when we consider existential formulae. □

Given a model $\mathfrak{M}$ and a formula φ, we write $[\![\varphi]\!]^{\mathfrak{M}}$ to denote the set of worlds of $\mathfrak{M}$ satisfying φ, that is $[\![\varphi]\!]^{\mathfrak{M}} = \{w \in W \mid \mathfrak{M}, w \models \varphi\}$. Whenever the model $\mathfrak{M}$ is contextually given and no confusion can arise, we omit the superscript $\mathfrak{M}$. If w is a world of $\mathfrak{M}$, we denote by $w^{\uparrow}$ the set of worlds that are greater of w with respect to P. Given $X \subseteq W$, we let $Pre^{\exists}(X) = \{w' \mid w'Rw \text{ for some } w \in X\}$ and $Pre^{\forall}(X) = \{w' \mid w'Rw \text{ implies } w \in X\}$. If $Y \subseteq W$ is a set then $Y^{\uparrow}$ denotes the set of elements of w whose upward closure is in Y, that is $Y^{\uparrow} = \{w \in W \mid w^{\uparrow} \subseteq Y\}$. If Y is a set $Y^{\mathbf{c}}$ denotes its complement.

Proposition 2. *Given a model* $\mathfrak{M}$ *and a world* w*, we have that:*

1. $\mathfrak{M}, w \models \varphi \rightarrow \psi$ *iff* $w \in ([\![\varphi]\!]^{\mathbf{c}} \cup [\![\psi]\!])^{\uparrow}$
2. $\mathfrak{M}, w \models \mathsf{E}\mathsf{X}\varphi$ *iff* $w \in Pre^{\exists}([\![\varphi]\!])$
3. $\mathfrak{M}, w \models \mathsf{A}\mathsf{X}\varphi$ *iff* $w \in (Pre^{\forall}([\![\varphi]\!]))^{\uparrow}$

Proof. We only prove (1) and (3).

1. For the ($\rightarrow$)-direction we reason by contraposition: suppose that there is w' such that $w' \in [\![\varphi]\!] \cap [\![\psi]\!]^{\mathbf{c}}$ and wPw'. This means that $\mathfrak{M}, w' \models \varphi$ and $\mathfrak{M}, w' \not\models \psi$ thus we conclude that $\mathfrak{M}, w \not\models \varphi \rightarrow \psi$. For the converse direction: suppose that $w \in ([\![\varphi]\!]^{\mathbf{c}} \cup [\![\psi]\!])^{\uparrow}$. Thus given any w' such that wPw' we have that $w' \in [\![\varphi]\!]^{\mathbf{c}}$ or $w' \in [\![\psi]\!]$. From this fact we deduce that $w' \in [\![\varphi]\!]$ (that is $w \notin [\![\varphi]\!]^{\mathbf{c}}$) implies $w' \in [\![\psi]\!]$ and we can conclude.

3. For the ($\rightarrow$)-direction: suppose that $\mathfrak{M}, w \models \mathsf{A}\,\mathsf{X}\varphi$. By definition, this means that given any w' such that wPw' all successors of w' are in $[\![\varphi]\!]$. This proves that $w^{\uparrow} \subseteq Pre^{\forall}([\![\varphi]\!])$ and thus $w \in (Pre^{\forall}([\![\varphi]\!]))^{\uparrow}$. For the converse direction, suppose that $s \in (Pre^{\forall}([\![\varphi]\!]))^{\uparrow}$. This means given any w' such that wPw' we have that all succesors of w' are in $[\![\varphi]\!]$. Thus any path starting at the given w' will satisfy φ on its second component. We thus deduce that $\mathfrak{M}, w \models \mathsf{A}\,\mathsf{X}\varphi$.

□

Proposition 3. *Define $\varphi \leftrightarrow \psi$ as $(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$. The following formulae are valid:*

1. $\mathsf{E}(\varphi \mathsf{U} \psi) \leftrightarrow \psi \vee (\varphi \wedge \mathsf{E}\,\mathsf{X}\mathsf{E}(\varphi \mathsf{U} \psi))$
2. $\mathsf{E}(\varphi \mathsf{R} \psi) \leftrightarrow \psi \wedge (\varphi \vee \mathsf{E}\,\mathsf{X}\mathsf{E}(\varphi \mathsf{R} \psi))$
3. $\psi \vee (\varphi \wedge \mathsf{A}\,\mathsf{X}\mathsf{A}(\varphi \mathsf{U} \psi)) \rightarrow \mathsf{A}(\varphi \mathsf{U} \psi)$
4. $\psi \wedge (\varphi \vee \mathsf{A}\,\mathsf{X}\mathsf{A}(\varphi \mathsf{R} \psi)) \rightarrow \mathsf{A}(\varphi \mathsf{R} \psi)$

Proof. We prove (2) and (3). Let $\mathfrak{M}$ be any model and w any of its worlds.

2. For the ($\rightarrow$)-direction, suppose that $\mathfrak{M}, w \models \mathsf{E}(\varphi \mathsf{U} \psi)$ and let w' be a world such that wPw'. We must check that $w' \models \psi \vee (\varphi \wedge \mathsf{E}\,\mathsf{X}\mathsf{E}(\varphi \mathsf{U} \psi))$. From the fact that w satisfies $\mathsf{E}(\varphi \mathsf{U} \psi)$, we deduce that either $w \in [\![\psi]\!]$ (in this case we conclude by Proposition 1), or that $w \in [\![\varphi]\!]$ and there is a path ρ such that $\rho_0 = w$, $\rho_i \in [\![\psi]\!]$ for some $i \geq 1$ and $\rho_j \in [\![\varphi]\!]$ for all $1 \leq j < i$, we thus conclude that $w \in [\![\psi \vee (\varphi \wedge \mathsf{E}\,\mathsf{X}\mathsf{E}(\varphi \mathsf{U} \psi))]\!]$ and, again by Proposition 1, that $w' \in [\![\psi \vee (\varphi \wedge (\mathsf{E}\,\mathsf{X}\mathsf{E}(\varphi \mathsf{U} \psi)))]\!]$. For the converse direction: suppose that $w \in [\![\psi \vee (\varphi \wedge \mathsf{E}\,\mathsf{X}\mathsf{E}(\varphi \mathsf{U} \psi))]\!]$ and let w' be a world such that wPw'. Since $w \in [\![\varphi \wedge (\psi \vee \mathsf{E}\,\mathsf{X}\mathsf{E}(\varphi \mathsf{U} \psi))]\!]$ either $w \in [\![\psi]\!]$ or $w \in [\![\varphi \wedge \mathsf{E}\,\mathsf{X}\mathsf{E}(\varphi \mathsf{U} \psi)]\!]$. In both cases, we deduce that $w \in [\![\mathsf{E}\varphi \mathsf{U} \psi]\!]$ and we conclude using Proposition 1.

3. Suppose that $\mathfrak{M}, w \models \psi \vee (\varphi \wedge \mathsf{A}\,\mathsf{X}\mathsf{A}(\varphi \mathsf{U} \psi))$ and let w' be a world such that wPw'. We must show that $w' \in [\![\mathsf{A}(\varphi \mathsf{U} \psi)]\!]$. If $w \in [\![\psi]\!]$ we conclude using Proposition 1 since any path starting at any world bigger than w will immediately satisfy ψ (and thus $(\varphi \mathsf{U} \psi)$). Otherwise, $w \in [\![\varphi \wedge \mathsf{A}\,\mathsf{X}\mathsf{A}(\varphi \mathsf{U} \psi))]\!]$ this means that $w \in [\![\varphi]\!]$ and given any world w' bigger than w, we will have that $w'Ru$ implies $u \in [\![\mathsf{A}(\varphi \mathsf{U} \psi)]\!]$. Since wPw', Proposition 1 allows us to conclude that $w' \in [\![\varphi]\!]$ and since given any u such that $w'Ru$ we have that $u \in [\![\mathsf{A}(\varphi \mathsf{U} \psi)]\!]$, we conclude that $w' \in [\![\mathsf{A}(\varphi \mathsf{U} \psi)]\!]$ as we wanted.

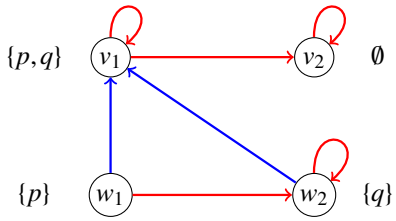
□

Note that, unlike CTL, in ICTL in Proposition 3.3 and 3.4 we have an implication. So, to prove that the other directions do not hold, we provide a counterexample in the following proposition.

Proposition 4. *The two formulae below are* ***not valid***

1. $\mathsf{A}(p \mathsf{U} q) \rightarrow q \vee (p \wedge (\mathsf{A}\,\mathsf{X}\mathsf{A}(p \mathsf{U} q))$
2. $\mathsf{A}(q \mathsf{R} p) \rightarrow p \wedge (q \vee (\mathsf{A}\,\mathsf{X}\mathsf{A}(q \mathsf{U} p))$

Proof. For both formulae, consider the model $\mathfrak{M}$ depicted below in which the preorder P is represented by the blue arrows, the relation R is represented by the red arrows and the valuation function is specified next to each node.



we have that $\mathfrak{M}, w_1 \models \mathsf{A}(p\mathsf{U}q)$ but neither $\mathfrak{M}, w_1 \models q$ nor $\mathfrak{M}, w_1 \models p \wedge \mathsf{AXA}(p\mathsf{U}q)$. In particular w_1 does not satisfy $\mathsf{AXA}(p\mathsf{U}q)$ because $w_1 P v_1$ and given the path $\tau = v_1 \cdot v_2^{\omega}$ we have that there is no $i \geq 1$ such that $\mathfrak{M}, \tau_j \models q$. Similarly, $\mathfrak{M}, w_1 \models \mathsf{A}(q\mathsf{R}p)$ but w_1 does not satisfy neither $p \wedge q$ nor $p \wedge \mathsf{AXA}(q\mathsf{R}p)$. □

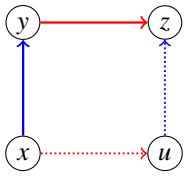
4 Conclusions and Future Works

We have sketched an Intuitionistic variant of CTL (ICTL) and proved some basic properties about this logic. There is still a lot, practically everything, to be done about this logic. Below, we outline some directions we would like to explore, without any specific hierarchical order.

Formal verification. In addition to its purely theoretical interest, the study of CTL has been fundamental for the development of applications in formal software verification. This is because Kripke models on which this logic is interpreted are particularly suitable for modeling the evolution of reactive systems. In order to make ICTL appealing, we would like to identify a class of reactive systems, or specific problems within these systems, that lend themselves well to being modeled using birelationals models.

Model Checking. The model checking problem for ICTL is the same as the one of CTL: given a (finite) birelational model $\mathfrak{M}$, a formula φ, and a world w can we decide whether $\mathfrak{M}, w \models \varphi$? This problem is P-space hard. For instance, given a CTL model $\mathfrak{R}$ one can see it as a birelational model by setting uPu' iff $u = u'$ for any world of $\mathfrak{R}$, and thus reduce the model-checking problem for CTL to the one of ICTL. Furthermore, even though the CTL fix points axioms do not hold in ICTL, we think we can characterize the semantics of the release and until operators by the upward-closure of the usual (classical) fix-point characterization.

Axiomatization & fix-points. The fix-point axioms are not valid in the intuitionistic variant of CTL that we have defined. It would be interesting to find out whether an axiomatization of ICTL can be obtained by other means. Possibly, one could think of adding another condition on birelational models in order to validate the axioms. For instance the following, for all $x, y, z \in W$: if xPy and yRz then there exist an $u \in W$ such that xRu and uPz. Diagrammatically this gives:



Multiagent Systems. It would be natural to extend our intutionistic intepretation of CTL to Alternating-time Temporal Logic (ATL) [1]. A natural interpretation of Intuitionistic ATL would be to consider that agents try to verify a temporal formula by executing some coordinate action on the base of a *shared knowledge* at a given time. We think that this interpretation would have much in common with the epistemic interpretation of ATL [5].

References

[1] Rajeev Alur, Thomas A. Henzinger & Orna Kupferman (2002): *Alternating-time temporal logic*. *J. ACM* 49(5), pp. 672–713, doi:10.1145/585265.585270.

[2] Philippe Balbiani, Joseph Boudou, Martín Diéguez & David Fernández-Duque (2020): *Intuitionistic Linear Temporal Logics*. *ACM Trans. Comput. Log.* 21(2), pp. 14:1–14:32, doi:10.1145/3365833.

[3] Anupam Das & Sonia Marin: *Brouwer meets Kripke: constructivising modal logic*. https://prooftheory.blog/2022/08/19/brouwer-meets-kripke-constructivising-modal-logic/. Posted on August 19 2022.

[4] E. Allen Emerson & Edmund M. Clarke (1982): *Using Branching Time Temporal Logic to Synthesize Synchronization Skeletons*. *Sci. Comput. Program.* 2(3), pp. 241–266, doi:10.1016/0167-6423(83)90017-5.

[5] Wiebe van der Hoek & Michael J. Wooldridge (2003): *Cooperation, Knowledge, and Time: Alternating-time Temporal Epistemic Logic and its Applications*. *Stud Logica* 75(1), pp. 125–157, doi:10.1023/A:1026185103185.

[6] Saul A. Kripke (1965): *Semantical Analysis of Intuitionistic Logic I*. In J.N. Crossley & M.A.E. Dummett, editors: *Formal Systems and Recursive Functions*, *Studies in Logic and the Foundations of Mathematics* 40, Elsevier, pp. 92–130, doi:10.2307/2270547.

[7] Paolo Mancosu (1997): *From Brouwer to Hilbert: The Debate on the Foundations of Mathematics in the 1920S*. Oxford, England: Oxford University Press USA.

[8] Per Martin-Löf (1984): *Intuitionistic type theory*. *Studies in proof theory* 1, Bibliopolis.

[9] Gordon D. Plotkin & Colin Stirling (1986): *A Framework for Intuitionistic Modal Logics*. In Joseph Y. Halpern, editor: *Proceedings of the 1st Conference on Theoretical Aspects of Reasoning about Knowledge, Monterey, CA, USA, March 1986*, Morgan Kaufmann, pp. 399–406.

[10] Amir Pnueli (1977): *The Temporal Logic of Programs*. In: *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, IEEE Computer Society, pp. 46–57, doi:10.1109/SFCS.1977.32.

[11] Arthur N. Prior (1955): *Time and Modality*. Westport, Conn.: Greenwood Press.

[12] Alex K. Simpson (1994): *The proof theory and semantics of intuitionistic modal logic*. Ph.D. thesis, University of Edinburgh, UK. Available at https://hdl.handle.net/1842/407.

[13] Morten Heine Sørensen & Pawel Urzyczyn (2006): *Lectures on the Curry-Howard Isomorphism*. Elsevier, doi:10.1016/s0049-237x(06)x8001-1. Available at https://doi.org/10.1016%2Fs0049-237x%2806%29x8001-1.

Runtime Verification for Trustworthy Computing

Robert Abela Christian Colombo Axel Curmi Mattea Fenech Mark Vella
Department of Computer Science, Faculty of ICT, University of Malta, Msida, Malta
name.surname@um.edu.mt

Angelo Ferrando
Department of Informatics, Bioengineering, Robotics and Systems Engineering, University of Genoa, 16145 Genova, Italy
name.surname@unige.it

Autonomous and robotic systems are increasingly being trusted with sensitive activities with potentially serious consequences if that trust is broken. Runtime verification techniques present a natural source of inspiration for monitoring and enforcing the desirable properties of the communication protocols in place, providing a formal basis and ways to limit intrusiveness. A recently proposed approach, RV-TEE, shows how runtime verification can enhance the level of trust to the Rich Execution Environment (REE), consequently adding a further layer of protection around the Trusted Execution Environment (TEE).

By reflecting on the implication of deploying RV in the context of trustworthy computing, we propose practical solutions to two threat models for the RV-TEE monitoring process: one where the adversary has gained access to the system without elevated privileges, and another where the adversary gains all privileges to the host system but fails to steal secrets from the TEE.

1 Introduction

The challenge of secure software execution is ultimately a game of cat and mouse where for every step forward in security, the attackers likewise launch increasingly sophisticated attacks. Suffice to consider the all too frequent examples[1] from recent history. Given this state of affairs, software architectures need to take a risk-based approach where progressively higher price for security is paid for the correspondingly sensitive components of a system (just like a traditional physical bank puts more hurdles the closer one gets to the vault where all the cash is). As robots are becoming more ubiquitous, they are naturally increasingly becoming likely targets of attacks; motivating more investment in their security [11].

In the security community, the idea of a trusted execution environment (TEE) is well known and is the ultimate objective whenever executing security-critical tasks [27], such as cryptographic protocol steps. Trusted computing finds its origin in trusted platform modules (TPM) that comprise tamper-evident hardware modules and enable secure boots [7]. However, TPM constitute just one component of a complete TEE solution as depicted in Fig. 1. In fact, the cornerstone of TEE lies in the isolated execution of critical code segments in a way that they become unreachable by malware infections of the non-trusted operating system and application code. A secure monitor, which is part of the TEE's trusted computing base (TCB), performs thorough checking of the dynamically provisioned code and the parameters of flows that call into the TEE.

[1] https://securityintelligence.com/heartbleed-openssl-vulnerability-what-to-do-protect/, https://github.com/openssl/openssl/issues/353, https://blog.trailofbits.com/2018/08/01/bluetooth-invalid-curve-points/, https://info.keyfactor.com/factoring-rsa-keys-in-the-iot-era,

A. Ferrando, R.C. Cardoso (Eds.): Third Workshop on
Agents and Robots for reliable Engineered Autonomy 2023 (AREA'23)
EPTCS 391, 2023, pp. 49–62, doi:10.4204/EPTCS.391.7

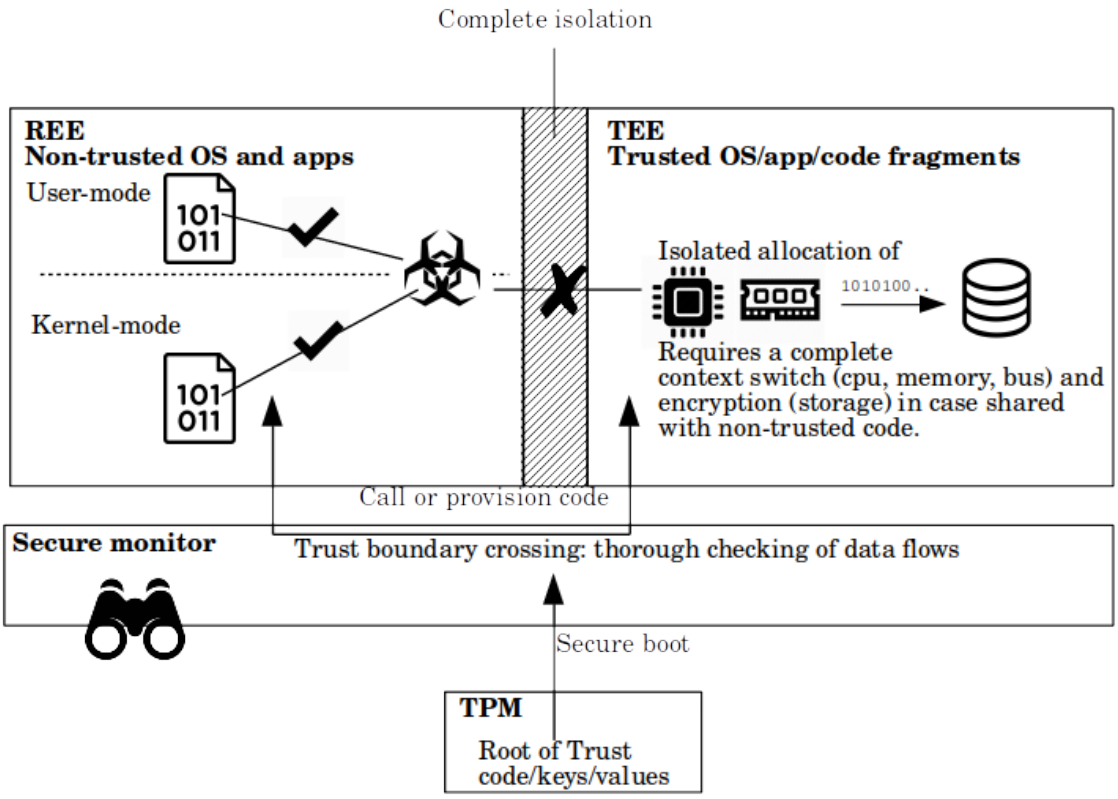


Figure 1: An overview of TEE components.

In previous works [40, 4, 13], we have proposed RV-TEE: A TEE which is supported by runtime verification techniques. The RV component complements the TEE services to elevate trust also inside the rich execution environment (REE)[2]. Even though the TEE's isolated context protects trusted application (TA) components, the rich application (RA) components executing inside the REE may still be required to demonstrate increased trust. In effect, RV-TEE establishes an intermediate level of trust, somewhere in between the levels offered by the TEE and the REE, since i) a clean un/trusted split of an application is far from simple in practice; ii) static verification techniques do not always scale and require complementary dynamic approaches. RV-TEE makes it a point to not be specific to common CPU-mode TEE implementations [23], whose security-efficiency trade-off may still not satisfy the levels of trust of specific security-critical applications. Rather, it considers TEE in its broadest sense possible [16], i.e., any platform realisation that splits runtime execution into trusted and rich execution modes. In sensitive applications such as military and governmental ones, the input/output overheads introduced by a removable hardware security module (HSM) of choice could be acceptable as long as the TEE employs a trusted hardware component.

Robotic systems are far from immune to vulnerabilities [14] and the independent use of TEE's [37] and RV [20, 15] for robotic applications is not new. However, to our knowledge, the proposal of com-

`https://labs.sentinelone.com/how-trickbot-hooking-engine-targets-windows-10-browsers`, `https://meltdownattack.com/`

[2]REE refers to execution which does not take place within a TEE.

bining the two in this context is novel. Interestingly, although one could simply introduce the two independently in a system, we show how the monitor can be further secured through the introduction of the HSM. While the RV-TEE approach contributes to the trustworthiness of the monitored process, the monitor itself does not run in a trusted environment, making it a potential target for attacks. In highly sensitive contexts [26], it is not enough to design for the prevention of attacks. Rather, one has to also design for handling situations where parts of the system have been taken over by the adversary. Applying this approach to the security aspects of the monitor itself: What guarantees do we have that the monitor has not been compromised? How can we be sure that the logs the monitor consumes and generates are actually authentic?

To answer these salient questions, we consider different threat models reflecting different levels of attack success. The first threat model considers the case where the adversary has gained access to the monitor-hosting system without elevated privileges, e.g., through an unpatched OS vulnerability the attacker manages to execute a malicious process. While this threat model doesn't directly compromise the monitoring process, it could potentially gather sensitive information and/or interfere with system resources and processes, e.g., the monitor log file in the filesystem. We handle this threat scenario by isolating the monitor through containerisation and consider the challenges that this brings about. The second threat model goes further by assuming that the adversary has gained all privileges to the host system but fails to steal secrets from the HSM. This gives the adversary full control over the system, including the monitor. The best we can aim for in such a scenario is that the attack is detected via tamper-evident logs. We outline the algorithm of an adaptation of SealFS— a filesystem employing cryptographic techniques to expose any modification of saved data.

In the next section, we introduce the notion of trusted execution, followed by an overview of how we have employed RV to enhance trust in Sec. 3. After elaborating on the two threat models under consideration in Sec. 4, we propose two practical solutions in each context in Sec. 5 and Sec. 6 respectively. Next, in Sec. 7, we give an update of the ongoing work to apply RV-TEE within robotics. We hope that as we conclude in the final sections, this paper offers a novel way of seeing and employing RV in secure contexts such as robotics, highlighting lessons learnt along with practical solutions for varying scenarios of compromise.

2 Trusted Execution Environment

A number of prominent TEE extensions to CPUs (CPU-TEE) have already reached industry level maturity. Intel's SGX [23] and AMD's SVM [18] technologies are primary examples. These constitute hardware extensions allowing an operating system to fully suspend itself, including interrupt handlers and all the code executing on other cores, in order to execute the trusted domain code within a code enclave. Another wide-spread example is ARM's TrustZone [25] that provides a CPU-TEE for mobile device platforms. Several other ideas also originate from academia, such as the suggestion to leverage existing hardware virtualisation extensions to implement TEE without having to resort to further specialised hardware [22]. Other works [9, 39, 30, 43] focus on providing practical solutions to port existing applications to a CPU-TEE.

Despite all these efforts, it is important to note that CPU-TEEs are not attack-proof since practical threats targeting all the aforementioned hardware have already been demonstrated [41, 28, 31, 21]. The root cause of these attacks stems from the overall design of CPU-TEEs. Their architecture follows an on-chip security subsystem approach [16], favouring TEE/REE context switching speed at the expense of having a shared micro-architecture, which ends up exposing a significant attack surface. However, the

architecture of a TEE is not constrained to the widely-available hardware that mainly follows the CPU-TEE design. Instead, the level of isolation offered by a TEE and the hardware components involved in its implementation are highly configurable, possibly to fit specific application requirements. For example, a TEE component may be fully implemented as an external security System-on-Chip (SoC) [16], trading efficiency with increased trust by eliminating shared micro-architectural components and bringing in trustworthy hardware of choice.

3 RV-TEE

Circumventing the need of TEE's to execute sensitive code on specific commodity CPU-TEE, we have proposed RV-TEE [40] to achieve a similar benefit by combining RV with any hardware security module of choice — whether a high-speed bus adapter [38], or a commodity USB stick [42]. More specialised options exist, including multi-chip modules that combine a security-enhanced microprocessor with a security controller, with the possibility of hardware-accelerated cryptography [10]. Compatibility-wise, if the design of the software to be secured already supports HSMs, e.g., PKCS#11, deployment even comes close to 'plug-and-play'. Ultimately, the level of protection with respect to tampering and resistance to side-channel attacks of the adopted HSM is carried forward to RV-TEE.

Overall, RV-TEE aims to be compatible with any physical TEE implementation — its primary goal being that of offering an intermediate level of trust to code executing inside the REE. It might be tempting to push more of the REE on the TEE so that the boundary between the REE and the TEE handles less sensitive elements. However, this approach risks turning the TEE into yet another REE in terms of potential attack surfaces, and which therefore would be counter productive. In the absence of a clean split between the TEE and REE, the result is a set of RA components that process sensitive derivatives of TA computations, e.g., plaintext derived from TA decryption. These RA components would benefit from the trust boundary monitoring for the provision of intermediate trust. The concerned trust boundaries comprise both that between the RA and the TEE as well as that between the RA and the rest of the REE (see Fig. 2). This additional trust boundary monitoring is RA-centric, and complements the existing security monitoring shown in Fig. 1 which rather is TA-centric.

The RV community has traditionally distinguished RV as control-flow or data-flow oriented monitoring (see for example [5]). Following this lead, at each boundary, we can loosely distinguish between control flow, i.e., triggering of code execution, usually through method calls, and data flow, i.e., passing of data through the stack or heap. In what follows we consider each one in turn.

Monitoring Control Flows Employing RV techniques to monitor the control flow is useful both as a means of detecting bugs and also to reduce the attack surface: if we know a priori how the code is expected to be used, then any deviations are either due to bugs, or due to malicious use of the codebase. This is useful both in RA-TEE as well as RA-REE control flows. Monitoring RA-TEE calls may uncover insecure usage of the HSM, while monitoring of RA-REE calls could expose attempts to execute external malicious code belonging to the attacker.

Specifying and monitoring of control flows is a well studied area in RV. In fact, our experience [40, 4, 13] has shown that this part of the RV-TEE instantiation is indistinguishable from traditional RV (see for example [8, 44, 33, 34]): A security protocol is analysed, properties are extracted and encoded in the specification language of choice, and subsequently synthesised into monitoring code using the preferred RV tool. More details are provided in the next section.

Given that RV is monitoring a boundary, the RV monitor itself could potentially be hosted (executed) by either side of the boundary. This is not an easy choice because on the one hand, it is

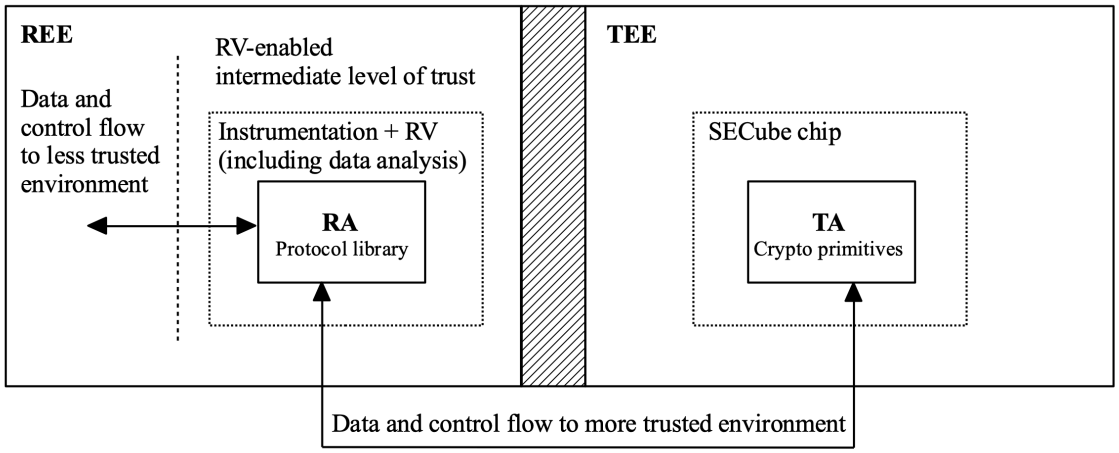


Figure 2: RV-TEE instantiation.

desirable to keep the size of the TA minimal while on the other hand, the monitor by its nature is a sensitive part of the system requiring protection. For all three past works [40, 4, 13], we have opted to run the RV code within the REE, while taking additional precautions to cater for the threat models considered in the following sections. We leave the exploration of deploying the monitor on the TEE side as future work, where the challenges of working with limited resources shouldn't be underestimated.

Monitoring Data Flows Monitoring the control flow, typically also gives access to the data flowing through the function arguments and return values. In this section, we are however particularly interested in the analysis of data which could be used to attack the system (inbound) or ex-filtrated out of the system (outbound), e.g., data leaving the TEE which should never include the keys, and data leaving the REE which should never leak the plaintext version (of sensitive information).

Checking for such flows can be done using dynamic taint tracking where data is followed through the system to ensure that it (or derivatives thereof) are not leaked. While this constitutes a precise approach, it is generally very expensive to deploy [19]. A cheaper alternative is to use taint inference [32], where rather than following data at every step of the way, the outflows are monitored for any sensitive data. This comes with several limitations: if the data is manipulated in any way, a simple string matching approach would immediately fail to flag issues where there might be. Therefore, an approximate string matching approach would be preferable while also lending itself amenable to speedup optimisations. Initial experiments in this regard [40] indicate that finetuning a number of parameters could establish a compromise of efficient execution and avoid accidental matching, while running the process asynchronously (possibly on separate resources) could also make the processor-intensive algorithm affordable.

4 Threat models

Being implemented within the REE, RV monitors constitute an attack surface which could particularly attract the adversaries' attention given its ability to raise intrusion alarms. One way of limiting the

monitor exposure to attacks (such as process injection using a debugging API) is to deploy it offline, but this of course limits the timeliness of the detection mechanism. In any case, instrumentation and recording of the events in a log file still need to happen within the REE and somehow need to be made accessible to the monitor. In this context, we consider two threat models, ordered in increasing severity:

Non-privileged access In this threat model, we consider the presence of user-space malware without root privileges. We assume that while such processes do not have elevated privileges, they still have sufficient privileges to perform malicious actions to interfere with the RV monitor and the monitored app through their data artefacts (e.g., log files, backups) or directly by tracing executing processes.

Successful privilege escalation In the event of an elevated malware infection, the possibilities are much wider, including access to entire filesystems, all devices and even the OS kernel. In other words, the only thing we assume under this threat model is that the secrets held inside the HSM have not been stolen, i.e., either the HSM is still operational and any attacks directed at it have been unsuccessful, or the HSM has been tampered with and became nonoperational with the secrets remaining safe.

Corresponding to these two threat models, a two-fold strategy is being proposed (refer to Fig. 3 which will be described further below: (i) the first involving process isolation to address attack vectors used for RV tampering without privilege escalation and (ii) employing tamper-evident techniques on logs (through an authentication scheme) are able to detect escalation attempts.

5 Isolated Monitoring Process

Namespaces [3] are a feature of the Linux kernel that partitions kernel resources such that a set of processes running in the same namespace are restricted to a corresponding set of resources. This has a similar effect to what chroot [2] does at the filesystem level. Common examples of namespace usage includes container software (e.g., Docker) to isolate processes, and Google Chrome to isolate its own browser tab processes hosting non-trusted code. Contrary to the typical use case of sandboxing non-trusted code, our aim is to use process isolation to safeguard the RV monitor and the instrumented monitored applications from a compromised OS. This setup provides protection from the *Non-privileged access* threat model through custom containers.

We consider two well-known containerisation tools: `runc`[3] and Docker[4]. `runc` is a tool for spawning and running containers on Linux according to OCI specifications. Docker is a software platform which allows developers to build, share, and deploy applications using container technology to separate the application from the rest of the infrastructure. The difference between the two is that Docker is at a higher-level, making use of `runc` underneath. Docker, consisting of a command-line interface tool and a daemon process named `dockerd`, utilises `runc` through `containerd`[5], which provides additional features to the lower-level tool such as shareable images, storage, and networking. While convenient, Docker tooling adds a significant attack surface[6] which we opted to avoid, and therefore made direct use of `runc`. As for code instrumentation, we opted for source-level function hooking aiming for minimal

[3] `https://github.com/opencontainers/runc`
[4] `https://github.com/docker`
[5] `https://github.com/containerd/containerd`
[6] `https://www.cvedetails.com/product/28125/Docker-Docker.html?vendor_id=13534`

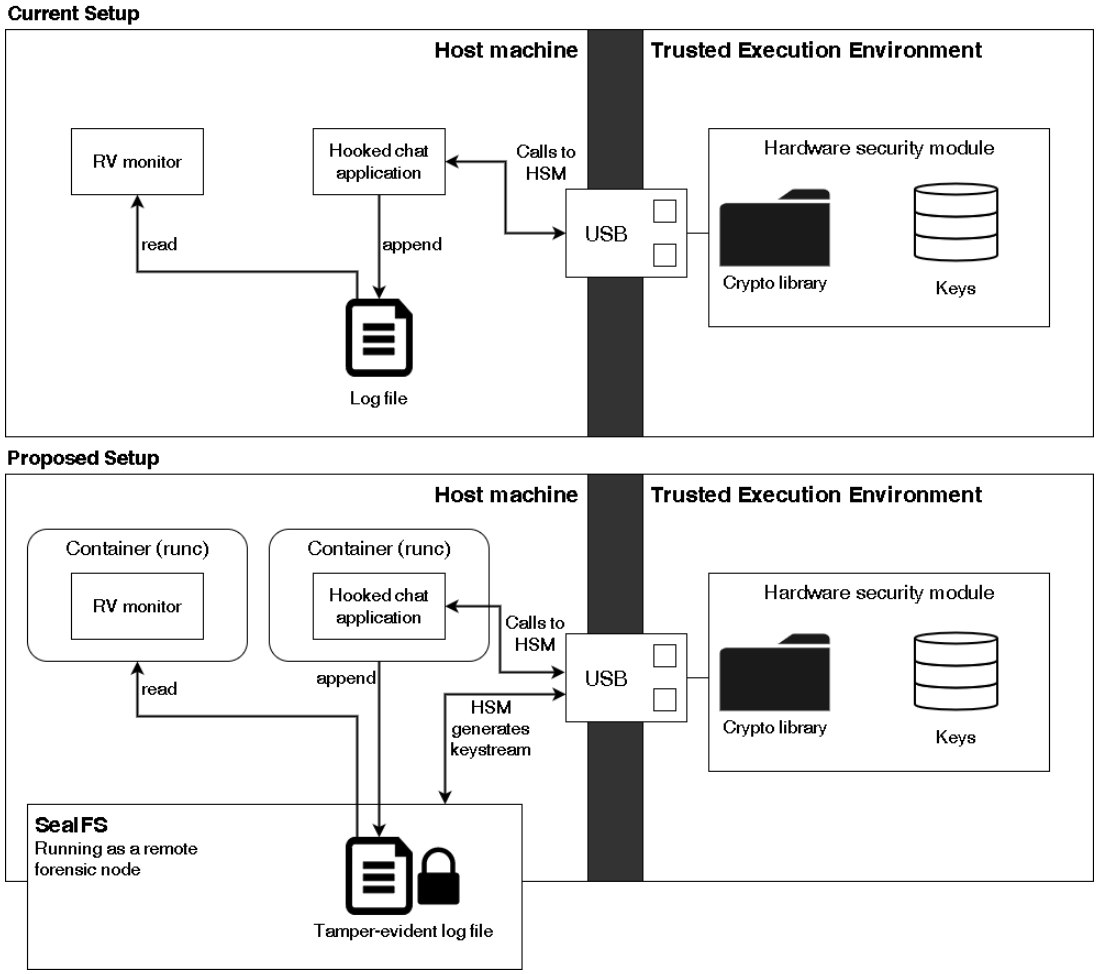


Figure 3: The proposed setup with isolated, tamper-evident monitoring.

impact on runtime overheads. Funchook[7], an API hook library, was deemed suitable for this task. The bottom left quadrant of Fig. 3 shows the RV monitor process and the instrumented application running in separate `runc` containers created through the combined use of namespaces and chroot.

The namespace/chroot-based isolation, along with function hooking-level instrumentation, is not expected to impact significantly on runtime overheads. Yet we made sure that this is the case with an empirical investigation considering the two scenarios of a chat application used in our previous publication [4]. Although we have yet to perform a case study directly on ROS, we expect that the message exchange mechanism in ROS will share several significant characteristics of the chat application case study.

These two testing scenarios involved a number of chat client applications connecting to one server, performing the protocol handshake to establish a secure session and exchanging some text messages between them. The client application was extended making it accept scripted session input in order to allow for automate testing. Artificial pauses were also introduced to better simulate a typical user's

[7] `https://github.com/kubo/funchook`

Table 1: Runtime overheads (in seconds).

Time (s)	No Instrumentation		Instrumentation		Increase
Scenario	**A**	**B**	**A**	**B**	**A & B**
Non-Containerised	20.042	13.028	20.044	13.026	0%
Runc	20.04	13.034	20.042	13.04	0.02%

interaction with the chat application. In both scenarios, only the chat client with `id=1` was instrumented, and all the other clients and server were running on the same machine.

Specifically, the testing scenarios were as follows:

- Scenario A: 3 clients involved, with client `id=1` creating a room (following the protocol steps for an initiator participant U_0).
- Scenario B: 3 clients involved, with client `id=1` joining the room (following the protocol steps for a non-initiator participant $U_{1\leq i\leq n}$).

The experiments were carried out on a Hetzner Cloud VM having two virtual Central Processing Units (vCPU) on an Intel Xeon Gold Processor with 4GB of RAM. All experiments were run 10 times and the results reflect their average running time. Results in Tbl. 1 confirm minimal overheads, not even close to 1%. However, there are other considerations of containerisation, namely that additional work will have to be done if the isolated application makes use of resources isolated via non-default namespaces (e.g., makes use of network or inter-process communication). In such cases the monitored application will have to account for the isolated setup by emulating/virtualising the missing devices and kernel resources through network proxies over virtual network interfaces. Such scenarios are expected to introduce further runtime overheads, and therefore further experimentation is needed.

6 Tamper-Evident Logging

In this section we now consider the *Privileged access* threat model. In this case, the adversary has full control of the system, possibly including physical access to the hardware. Our only assumption will be that the adversary cannot compromise the HSM without breaking it, i.e., the keys stored inside it remain secret.

Log analysis is an important tool for forensic investigation and similarly, most monitoring tools depend on log files both as their source of input and also to record monitoring verdicts. Logs can however be forged by intruders to hide or fake evidence. Sending logs to a remote system might mitigate this risk, but it can be seen as simply shifting the problem to another location on the network.

While it is not possible to stop a fully privileged adversary from tampering with the logs, we adopt the SealFS filesystem [35] whereby any modification doesn't go unnoticed. SealFS implements a scheme that authenticates local log files based on a forward integrity model, i.e., log data from boot time to the instant the malicious code elevates privileges can be authenticated. It does not depend on specialised security hardware or securing a distributed system. An intuitive summary of the procedure is as follows (refer to the bottom left quadrant of Fig. 3):

Generation of keystream A random keystream is generated, in our case by the HSM in order to have more entropy, prior to loading SealFS. This keystream is used in the following steps and a copy is stored on the forensic node (or safe external storage[8]) for the purposes of verification.

[8]We acknowledge that communication to a remote forensic node or external storage might not be an option during operation

Setting up The SealFS module creates an offset on the HSM (initialised at zero) representing the number of bytes consumed from the key and creates a file, $SEAL_{log}$, within the forensic node to store the authentication data and metadata for the logs.

Execution Referring to Alg. 1, when some data D of size D_{sz} is to be appended to a log file L at offset L_{off}, the following operations are executed in the HSM[9]: A chunk C of the key is read and the corresponding zone is "burnt" (lines 2–3), leaving no trace of it. An HMAC of the log concatenation, uniquely identifying the log file, the offset in the log, the data length, the key offset, and data D is generated (line 4). The key chunk is removed from memory (line 5). The record is sent to the SealFS module and appended to $SEAL_{log}$ (line 6). Finally, the offsets are updated accordingly (lines 7–8).

input : System event/monitor verdict D of length D_{sz}
input : Log file L
input : Log file offset L_{off}
input : HSM-stored key K
input : HSM-stored key offset K_{off}
input : Fixed key chunk size C_{sz}
input : Authentication data log file SEAL_{log}

1 append D to L at offset L_{off}; // add data to log file
2 $C \leftarrow K[K_{off} \ldots (K_{off}+C_{sz}-1)]$; // copy key chunk
3 $K[K_{off} \ldots (K_{off}+C_{sz}-1)] \leftarrow RANDOM()$; // burn key chunk
4 $H \leftarrow HMAC(C, L\|L_{off}\|D_{sz}\|K_{off}\|D)$; // generate HMAC using C
5 remove C from memory ;
6 append $(L, L_{off}, D_{sz}, K_{off}, H)$ to SEAL_{log}; // create record in SEAL_{log}
7 $L_{off} \leftarrow L_{off} + D_{sz}$; // update log file offset
8 $K_{off} \leftarrow K_{off} + C_{sz}$; // update key offset

Algorithm 1: Appending tamper-evident logfile (adapted from [35])

When it comes to verifying that the log is intact, all the records of $SEAL_{log}$ are verified sequentially using the safe copy of the key stored as in the first step above. If the adversary removes any log records from $SEAL_{log}$, or if any log file is truncated or shortened, the verification fails. Similarly, if the adversary modifies any of the fields of any record in the log file, the verification fails because the HMAC would not match. The verification process can either be carried out on-demand, i.e., whenever the system auditor decides to, or on particular events, e.g., at regular time intervals, after a specific number of log entries, or when suspected malicious actions have taken place.

We note that our proposal depends on the HSM being used as the root of trust of the whole scheme. An attestation protocol (e.g., see [6]) could be used to provide assurance to a remote observer that the HSM is still being used by the system, and by extension that the guarantees it affords are still in place. However, in our proposal, since the HSM is burning parts of the key which is stored in its entirety in safe storage, by verifying the log file, one would also be indirectly verifying that the system is still using the HSM (keeping in mind our only assumption that the adversary fails to steal secrets from the HSM).

of autonomous robots. In such cases, the key stream could be generated and safely stored *before* the start of the robot's operation.

[9] Given the limited resources of the HSM, the process described here could be optimised through techniques such as ratcheting (see SealFSv2 [24]) which can work with less memory requirements.

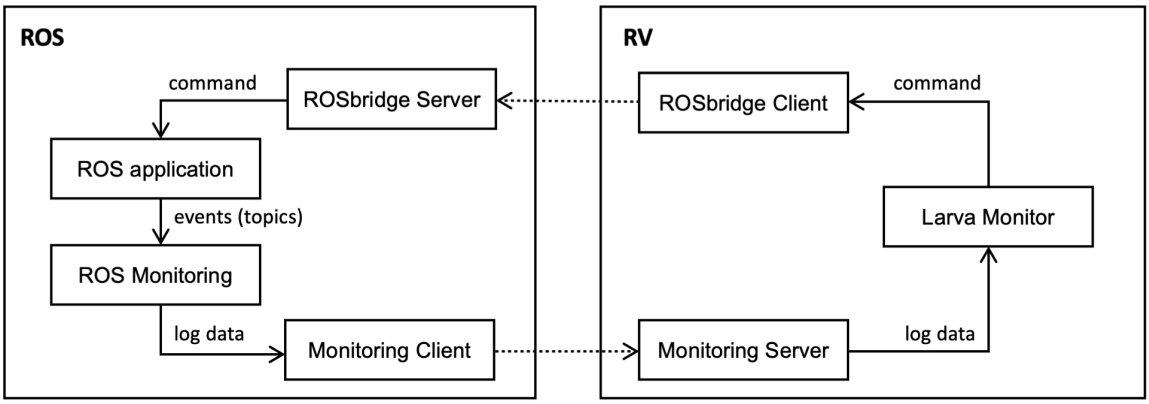


Figure 4: ROS monitoring using Larva as an oracle.

7 Implementation for Robotic Systems

As a step towards deploying RV-TEE within robotic systems, we have developed a prototype which combines the RV tool which we have used in our previous works, Larva [12], with ROSMonitoring [15] to successfully monitor a ROS-based system.

Fig. 4 shows how ROSMonitoring listens for relevant events (also known as *topics*) occurring within the ROS application. These are then forwarded to the Larva monitor, which in turn can send back commands to the system being monitored. As ROSMonitoring is agnostic to the chosen verification system (also referred to as *oracle*) by design, it was not difficult to combine it with Larva.

At the time of writing this paper, the implementation of the rest of the proposed secured RV-TEE setup (described in this paper) is underway. Depending on the robotic case study which would be considered in the future, we expect the Larva process to run in a separate container, the forensic node.

It is important to note that, as for the software systems discussed previously, robotic applications may also be the target of attacks. In case of robotic applications developed in ROS[10], security is a big concern. Indeed, ROS was not born to be exploited in industrial applications and security was not taken into consideration in its development. As mentioned before, ROS nodes can communicate through messages. Such messages are shared over channels, that in ROS are named topics. In ROS, such topics are not protected whatsoever; that is, one cannot protect the data exchanged (except by encrypting the data before sending them). In fact, any ROS node can be the publisher (resp., subscriber) of a topic and hence there is no way to guarantee an attacker node will not intercept the messages on our topics (by simply subscribing to them). One can solve this issue by deploying the robotic system through ROS2 (the newer version of ROS), which offers security mechanisms to forbid attacker nodes from intercepting private messages. However, even with ROS2, the protection against attackers with privileged access is limited.

In both ROS and ROS2, the exploitation of RV-TEE would be of great impact. Thanks to the Larva component currently under development, it is possible, through ROSMonitoring, to intercept and verify the messages exchanged on the topics. By doing so, it is possible to implement the bridge (as partially shown in Figure 4) which would connect the TEE node with the rest of the system. Moreover, since ROS is node-based, our approach could exploit such distribution by deploying the TEE component as a node

[10] https://www.ros.org/

in the net. The rest of the nodes would be considered non-protected nodes that could be the target of malicious attacks. In such a scenario, RV-TEE would be deployed through ROSMonitoring and would be used to protect the information exchange between the secure node (TEE) and the rest of the robotic system.

Most importantly, it is relevant to observe that the exploitation of RV-TEE with ROSMonitoring would be applicable both in ROS and ROS2 (since ROSMonitoring is supported in both ROS versions). Moreover, both ROS and ROS2 would gain from such integration, since the security techniques natively deployed in ROS2 would not protect the system from attackers with privileged access.

While the additional forensic node can assure adherence to some security policy established for the ROS2 computational graph, RV-TEE can also secure communications between nodes on different machines. Secure inter-machine communication in ROS2 is provided by the underlying Data Distribution Service (DDS) [36], which is the programmatic abstraction enabling the publish/subscribe-based communication. Once secure communication is enabled in DDS, the security plugins provided by the specific implementation, e.g., Eclipse Cyclone DDS [1], provide node authentication, data encryption, and integrity services. Any such implementation executing on a robot-controlling PC is prone to threats related to incorrect cryptographic protocol implementation and malware attacks. Thus, DDS security plugin implementations through RV-TEE can offer enhanced resilience, similar to how RV-TEE has secured both classic and post-quantum cryptography in previous works (hence the relevance of the chat application case study presented above).

8 Conclusions

While there are numerous accounts in the literature of the application RV techniques to the area of security (see for example [8, 44, 33, 34]), the challenging task of securing the monitor implementation itself seems not to be so well studied. In fact, the survey of RV challenges in 2019 [29] leaves this aspect out. There are of course several other considerations to achieving "high-assurance" RV [17], but securing and protecting the monitoring code under various threat models cannot be left out if RV is to be deployed in real-life, high-security scenarios such as robotics.

By bridging the gap between the REE and the TEE, RV-TEE presents a flexible way of creating an intermediate level of trust without being restricted to specific specialised hardware. Yet, apart from the usual concerns of monitor correctness and non-intrusiveness, the context requires the monitor itself to be adapted for adversarial conditions. Considering two incrementally compromising threat models, we have thus first isolated the monitoring process to make it harder for attackers to tamper with. Initial results in this regard show that any overheads introduced by containerisation are not of the processing kind but rather due to potential proxying of resources. To cater for the second threat model, we have proposed the integration of a tamper-evident filesystem to protect system and monitor logs from modification. Though an adversary might have been successful in penetrating into the heart of the system, we can be sure that evidence of system log modification cannot be concealed.

9 Future Work

There are still several questions to be answered in the context of RV-TEE. Here are a few of these organised under the following headings:

Further experimentation In this paper we have presented a proof of concept for securing RV monitors. Next, we plan to explore the practical implications of the current setup. In particular, we need to

answer questions such as: What is the impact on the HSM given that it will also be used to encrypt log entries (apart from the other tasks assigned to it)?

RV within the TEE? It could be interesting to explore the possibility of deploying elements of RV as part of the TCB of the TEE itself. However, apart from the practical challenge of further loading the already resource constrained TEE, there is also a conceptual objection: The code deployed on the TEE usually consists of well established primitives which are deployed within the TEE precisely because they are trusted. Therefore, it is yet to be seen whether this is something worth investigating. As a first step, one would need to consider a number of interesting properties at this level and note their cost-benefit analysis. For example, the property concerning the quality of the randomness, which is at the core of cryptographic primitives, is far from straightforward to monitor.

Taint inference The string matching algorithm implemented for taint inference has several set thresholds (e.g., when to trigger fine-grained string matching) and a number of parameters which could also be fine-tuned (e.g., by how much to move the window during coarse-grained matching). These are also dependant on the size of the buffer under consideration, giving rise to various possible experiments, not least on how to efficiently use the hardware available for speedups. Furthermore, selection of taint sinks to make taint inference resilient to high-entropy transformations e.g., compression and encryption, needs further study.

References

[1] *Eclipse Cyclone DDS*. `https://github.com/eclipse-cyclonedds/cyclonedds`. Accessed: 2023-07-25.

[2] *Linux chroot*. `https://man7.org/linux/man-pages/man2/chroot.2.html`. Accessed: 2023-05-17.

[3] *Linux namespaces*. `https://man7.org/linux/man-pages/man7/namespaces.7.html`. Accessed: 2023-05-17.

[4] Robert Abela, Christian Colombo, Peter Malo, Peter Sýs, Tomás Fabsic, Ondrej Gallo, Viliam Hromada & Mark Vella (2021): *Secure Implementation of a Quantum-Future GAKE Protocol*. In: *Security and Trust Management - 17th International Workshop, STM 2021, Darmstadt, Germany, October 8, 2021, Proceedings*, *Lecture Notes in Computer Science* 13075, Springer, pp. 103–121, doi:10.1007/978-3-030-91859-0_6.

[5] Wolfgang Ahrendt, Jesús Mauricio Chimento, Gordon J. Pace & Gerardo Schneider (2017): *Verifying data- and control-oriented properties combining static and runtime verification: theory and tools*. *Formal Methods Syst. Des.* 51(1), pp. 200–265, doi:10.1007/s10703-017-0274-y.

[6] Muhammad Naveed Aman, Mohamed Haroon Basheer, Siddhant Dash, Jun Wen Wong, Jia Xu, Hoon Wei Lim & Biplab Sikdar (2020): *HAtt: Hybrid remote attestation for the Internet of Things with high availability*. *IEEE Internet of Things Journal* 7(8), pp. 7220–7233, doi:10.1109/JIOT.2020.2983655.

[7] Ross Anderson, Mike Bond, Jolyon Clulow & Sergei Skorobogatov (2006): *Cryptographic processors-a survey*. *Proceedings of the IEEE* 94(2), pp. 357–369, doi:10.1109/JPROC.2005.862423.

[8] Andreas Bauer & Jan Jürjens (2010): *Runtime verification of cryptographic protocols*. *Computers & Security* 29(3), pp. 315–330, doi:10.1016/j.cose.2009.09.003.

[9] Andrew Baumann, Marcus Peinado & Galen Hunt (2015): *Shielding applications from an untrusted cloud with haven*. *ACM Transactions on Computer Systems (TOCS)* 33(3), pp. 1–26, doi:10.1145/2799647.

[10] Blu5 Labs (2020): *SEcube – Reconfigurable silicon*. `https://www.secube.eu/site/assets/files\/1145/secube_datasheet_-_r7.pdf`. Accessed: 2022-05-02.

[11] Alessio Botta, Sayna Rotbei, Stefania Zinno & Giorgio Ventre (2023): *Cyber security of robots: A comprehensive survey*. *Intelligent Systems with Applications* 18, p. 200237, doi:10.1016/j.iswa.2023.200237.

[12] Christian Colombo, Gordon J. Pace & Gerardo Schneider (2009): *LARVA — Safer Monitoring of Real-Time Java Programs (Tool Paper)*. In: *Seventh IEEE International Conference on Software Engineering and Formal Methods (SEFM)*, IEEE Computer Society, pp. 33–37, doi:10.1109/SEFM.2009.13.

[13] Axel Curmi, Christian Colombo & Mark Vella (2022): *RV-TEE-Based Trustworthy Secure Shell Deployment: An Empirical Evaluation*. *Journal of Object Technology* 21(2), doi:10.5381/jot.2022.21.2.a4.

[14] Gelei Deng, Guowen Xu, Yuan Zhou, Tianwei Zhang & Yang Liu (2022): *On the (In)Security of Secure ROS2*. In Heng Yin, Angelos Stavrou, Cas Cremers & Elaine Shi, editors: *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, ACM, pp. 739–753, doi:10.1145/3548606.3560681.

[15] Angelo Ferrando, Rafael C. Cardoso, Michael Fisher, Davide Ancona, Luca Franceschini & Viviana Mascardi (2020): *ROSMonitoring: A Runtime Verification Framework for ROS*. In Abdelkhalick Mohammad, Xin Dong & Matteo Russo, editors: *Towards Autonomous Robotic Systems - 21st Annual Conference, TAROS 2020, Nottingham, UK, September 16, 2020, Proceedings*, *Lecture Notes in Computer Science* 12228, Springer, pp. 387–399, doi:10.1007/978-3-030-63486-5_40.

[16] GlobalPlatform (2018): *TEE System Architecture Version 1.2*. *Doc ref: GPD_SPE_009*.

[17] Alwyn Goodloe (2016): *Challenges in High-Assurance Runtime Verification*. In Tiziana Margaria & Bernhard Steffen, editors: *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques - 7th International Symposium, ISoLA 2016, Imperial, Corfu, Greece, October 10-14, 2016, Proceedings, Part I*, *Lecture Notes in Computer Science* 9952, pp. 446–460, doi:10.1007/978-3-319-47166-2_31.

[18] David Kaplan, Jeremy Powell & Tom Woller (2016): *AMD memory encryption*. *White paper*.

[19] Vasileios P Kemerlis, Georgios Portokalidis, Kangkook Jee & Angelos D Keromytis (2012): *libdft: Practical dynamic data flow tracking for commodity systems*. *Acm Sigplan Notices* 47(7), pp. 121–132, doi:10.1145/2365864.2151042.

[20] Yunus Sabri Kirca, Elif Degirmenci, Zekeriyya Demirci, Ahmet Yazici, Metin Ozkan, Salih Ergun & Alper Kanak (2023): *Runtime Verification for Anomaly Detection of Robotic Systems Security*. *Machines* 11(2), doi:10.3390/machines11020166.

[21] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz & Yuval Yarom (2018): *Spectre attacks: Exploiting speculative execution*. *arXiv preprint arXiv:1801.01203*, doi:10.1109/SP.2019.00002.

[22] Jonathan M McCune, Yanlin Li, Ning Qu, Zongwei Zhou, Anupam Datta, Virgil Gligor & Adrian Perrig (2010): *TrustVisor: Efficient TCB reduction and attestation*. In: *Security and Privacy (SP), 2010 IEEE Symposium on*, IEEE, pp. 143–158, doi:10.1109/SP.2010.17.

[23] Frank McKeen, Ilya Alexandrovich, Ittai Anati, Dror Caspi, Simon Johnson, Rebekah Leslie-Hurd & Carlos Rozas (2016): *Intel® software guard extensions (intel® sgx) support for dynamic memory management inside an enclave*. In: *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*, ACM, pp. 1–9, doi:10.1145/2948618.2954331.

[24] Gorka Guardiola Muzquiz & Enrique Soriano-Salvador (2023): *SealFSv2: combining storage-based and ratcheting for tamper-evident logging*. *Int. J. Inf. Sec.* 22(2), pp. 447–466, doi:10.1007/s10207-022-00643-1.

[25] Sandro Pinto & Nuno Santos (2019): *Demystifying Arm trustzone: A comprehensive survey*. *ACM Computing Surveys (CSUR)* 51(6), pp. 1–36, doi:10.1145/3291047.

[26] Scott Rose, Oliver Borchert, Stuart Mitchell & Sean Connelly (2020): *Zero Trust Architecture*, doi:10.6028/NIST.SP.800-207. Special Publication (NIST SP), National Institute of Standards and Technology.

[27] Mohamed Sabt, Mohammed Achemlal & Abdelmadjid Bouabdallah (2015): *Trusted execution environment: what it is, and what it is not*. In: *14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications*, pp. 57–64, doi:10.1109/Trustcom.2015.357.

[28] Mohamed Sabt & Jacques Traoré (2016): *Breaking into the keystore: A practical forgery attack against Android keystore*. In: *European Symposium on Research in Computer Security*, Springer, pp. 531–548, doi:10.1007/978-3-319-45741-3_27.

[29] César Sánchez, Gerardo Schneider, Wolfgang Ahrendt, Ezio Bartocci, Domenico Bianculli, Christian Colombo, Yliès Falcone, Adrian Francalanza, Srdan Krstic, João M. Lourenço, Dejan Nickovic, Gordon J. Pace, José Rufino, Julien Signoles, Dmitriy Traytel & Alexander Weiss (2019): *A survey of challenges for runtime verification from advanced application domains (beyond software)*. *Formal Methods Syst. Des.* 54(3), pp. 279–335, doi:10.1007/s10703-019-00337-w.

[30] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz & Mark Russinovich (2015): *VC3: Trustworthy data analytics in the cloud using SGX*. In: *2015 IEEE Symposium on Security and Privacy*, IEEE, pp. 38–54, doi:10.1109/SP.2015.10.

[31] Mark Seaborn & Thomas Dullien (2015): *Exploiting the DRAM rowhammer bug to gain kernel privileges*. *Black Hat* 15.

[32] R Sekar (2009): *An Efficient Black-box Technique for Defeating Web Application Attacks.* In: *Proceedings of the 16th Annual Network and Distributed System Security Symposium.*

[33] Konstantin Selyunin, Stefan Jaksic, Thang Nguyen, Christian Reidl, Udo Hafner, Ezio Bartocci, Dejan Nickovic & Radu Grosu (2017): *Runtime Monitoring with Recovery of the SENT Communication Protocol*. In: *Computer Aided Verification - 29th International Conference, CAV*, pp. 336–355, doi:10.1007/978-3-319-63387-9_17.

[34] Jinghao Shi, Shuvendu Lahiri, Ranveer Chandra & Geoffrey Challen (2018): *VeriFi: Model-Driven Runtime Verification Framework for Wireless Protocol Implementations*. *CoRR* abs/1808.03406, doi:10.48550/arXiv.1808.03406.

[35] Enrique Soriano-Salvador & Gorka Guardiola Muzquiz (2021): *SealFS: Storage-based tamper-evident logging*. *Comput. Secur.* 108, p. 102325, doi:10.1016/j.cose.2021.102325.

[36] OMG Available Specification (2015): *Data distribution service for real-time systems version 1.4*. *Object Management Group (OMG)* (formal/2015-04-10).

[37] Mariacarla Staffa, Giovanni Mazzeo & Luigi Sgaglione (2018): *Hardening ROS via Hardware-assisted Trusted Execution Environment*. In: *27th IEEE International Symposium on Robot and Human Interactive Communication, RO-MAN 2018, Nanjing, China, August 27-31, 2018*, IEEE, pp. 491–494, doi:10.1109/ROMAN.2018.8525696.

[38] Thales (2020): *High Assurance Hardware Security Modules*. `https://cpl.thalesgroup.com/encryption/hardware-security-modules/network-hsms`. Accessed: 2020-08-10.

[39] Chia-Che Tsai, Donald E Porter & Mona Vij (2017): *Graphene-sgx: A practical library OS for unmodified applications on SGX*. In: *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pp. 645–658.

[40] Mark Vella, Christian Colombo, Robert Abela & Peter Špaček (2021): *RV-TEE: secure cryptographic protocol execution based on runtime verification*. *Journal of Computer Virology and Hacking Techniques*, pp. 1–20, doi:10.1007/s11416-021-00391-1.

[41] Rafal Wojtczuk & Joanna Rutkowska (2009): *Attacking Intel trusted execution technology*. *Black Hat DC* 2009.

[42] Yubico (2020): *Protect your digital world with YubiKey*. `https://www.yubico.com/`. Accessed: 2020-08-10.

[43] Fengzhe Zhang, Jin Chen, Haibo Chen & Binyu Zang (2011): *Cloudvisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization*. In: *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pp. 203–216, doi:10.1145/2043556.2043576.

[44] X. Zhang, W. Feng, J. Wang & Z. Wang (2016): *Defensing the malicious attacks of vehicular network in runtime verification perspective*. In: *2016 IEEE International Conference on Electronic Information and Communication Technology (ICEICT)*, pp. 126–133, doi:10.1109/ICEICT.2016.7879666.

The Impact of Strategies and Information in Model Checking for Multi-Agent Systems

Vadim Malvone
LTCI, Télécom Paris, Institut Polytechnique de Paris,
Palaiseau, France
vadim.malvone@telecom-paris.fr

System correctness is one of the most crucial and challenging objectives in software and hardware systems. With the increasing evolution of connected and distributed systems, ensuring their correctness requires the use of formal verification for multi-agent systems. In this paper, we present a summary of certain results on model checking for multi-agent systems that derive from the selection of strategies and information for agents. Additionally, we discuss some open directions for future research.

1 Introduction

The problem of assuring systems correctness is particularly felt in hardware and software design, especially in safety-critical scenarios. When we talk about a safety-critical system, we mean the one in which failure is not an option. To face this problem, several methodologies have been proposed. Among these, Model Checking (MC) [11, 10] results to be very useful. This approach provides a formal-based methodology to model systems, to specify properties via temporal logics, and to verify that a system satisfies a given specification.

Notably, first applications of model checking just concerned closed systems, which are characterized by the fact that their behavior is completely determined by their internal states. Unfortunately, model checking techniques developed to handle closed systems turn out to be quite useless in practice, as most of the systems are open and are characterized by an ongoing interaction with other systems. To overcome this problem, model checking has been extended to Multi-Agent Systems (MAS). In the latter context, temporal logics have been extended to temporal logics for the strategic reasoning such as Alternating-time Temporal Logic (ATL) [2] and Strategy Logic (SL) [21].

Given the logics under exam, there are two key aspects in MAS to determine the model checking complexity: the type of strategies and the agents' information. A strategy is a generic conditional plan that prescribes an action at each step of the MAS. There are two main classes of strategies: *memoryless* and *memoryfull*. In the former case, agents choose an action by considering only the current state, while in the latter case, agents choose an action by considering the full history of the MAS. Regarding information, we distinguish between *perfect* and *imperfect* information MAS. The former corresponds to a basic setting in which every agent has full knowledge about the MAS. However, real-life scenarios often involve situations where agents must act without having all relevant information at hand. In computer science, these situations occur, for example, when some variables of a system are internal/private and not visible to an external environment. In MAS models, imperfect information is usually modeled by setting an indistinguishability relation over the states of the MAS. This feature deeply impacts on the MC complexity. For example, ATL and SL become undecidable in the context of imperfect information and memoryfull strategies [12].

A. Ferrando, R.C. Cardoso (Eds.): Third Workshop on
Agents and Robots for reliable Engineered Autonomy 2023 (AREA'23)
EPTCS 391, 2023, pp. 63–70, doi:10.4204/EPTCS.391.8

In this paper, we present a review of results on model checking for multi-agent systems concerning strategy classes and agent visibility, and explore potential directions for future research.

Outline In Section 2, we recall the definition of concurrent game structures and the syntax of ATL and SL. Then, in Section 3, we review the main classes of strategies and the concept of information. In Section 4, we revisit the model checking complexities for the aforementioned contexts. Finally, we conclude by providing some future directions in Section 5.

2 Model and logics for MAS

In this section, we recall the definition of a formal model for MAS and the syntax of two well-known logics for strategic reasoning: ATL and SL.

We start by recalling the definition of concurrent game structures [2].

Definition 1 *Given sets Ag of agents and AP of atoms, a* concurrent game structure (CGS) *is a tuple $M = \langle S, s_0, \{Act_i\}_{i\in Ag}, \{\sim_i\}_{i\in Ag}, d, \delta, V\rangle$ such that:*

- *S is a finite, non-empty set of* states*, with* initial state *$s_0 \in S$.*
- *For every $i \in Ag$, Act_i is a finite, non-empty set of* actions*.*
  Let $Act = \bigcup_{i\in Ag} Act_i$ be the set of all actions, and $ACT = \prod_{i\in Ag} Act_i$ the set of all joint actions*, i.e., tuples of actions.*
- *For every $i \in Ag$, $\sim_i$ is a relation of* indistinguishability *between states, that is, an equivalence relation on S. Given states $s, s' \in S$, $s \sim_i s'$ iff s and s' are said to be* observationally indistinguishable *for agent i.*
- *The* protocol function *$d : Ag \times S \to (2^{Act} \setminus \emptyset)$ defines the availability of actions so that for every $i \in Ag$, $s \in S$, (i) $d(i,s) \subseteq Act_i$ and (ii) $s \sim_i s'$ implies $d(i,s) = d(i,s')$.*
- *The* transition function *$\delta : S \times ACT \to S$ assigns a successor state $s' = \delta(s, \vec{\alpha})$ to each state $s \in S$, for every joint action $\vec{\alpha} \in ACT$ such that $a_i \in d(i,s)$ for every $i \in Ag$, that is, $\vec{\alpha}$ is* enabled *at s.*
- *$V : S \times AP \to \{\top, \bot\}$ is a* two-valued labelling function*.*

According to Def. 1, a CGS describes the interactions of a group Ag of agents, starting from the initial state $s_0 \in S$, following the transition function δ. The latter is constrained by the availability of actions to agents, as specified by the protocol function d. A history $h \in S^+$ is a finite (non-empty) sequence of states.

Now, we recall the syntax of Alternating-time Temporal Logic [2].

Definition 2 (ATL^*) *The state (φ) and path (ψ) formulas in ATL^* are defined as follows, where $p \in AP$ and $\Gamma \subseteq Ag$:*

$$\begin{aligned} \varphi &::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \langle\langle\Gamma\rangle\rangle\psi \\ \psi &::= \varphi \mid \neg\psi \mid \psi \wedge \psi \mid X\psi \mid (\psi U \psi) \mid (\varphi R \varphi) \end{aligned}$$

Formulas in ATL^* are all and only the state formulas.

ATL extends Computation Tree Logic (CTL) [9] in which the existential E and the universal A path quantifiers are replaced with strategic modalities of the form $\langle\langle\Gamma\rangle\rangle$ and $[\![\Gamma]\!]$, where Γ is a set of agents. A formula $\langle\langle\Gamma\rangle\rangle\psi$ is read as *"the agents in coalition Γ have a strategy to achieve ψ"*. The meaning of temporal operators *next* X and *until* U is standard [11]. Operators *unavoidable* $[\![\Gamma]\!]$, *release* R, *finally* F, and *globally* G can be introduced as usual.

The formulas in the ATL fragment of ATL* are obtained from Def. 2 by restricting path formulas ψ to the temporal operators.

To conclude this section, we recall the syntax of Strategy Logic [21].

Definition 3 (SL Syntax) *Given the set AP of atoms, variables Var, and agents Ag, the formal syntax of SL is defined as follows, where $p \in AP$, $x \in Var$, and $a \in Ag$:*

$$\varphi \quad ::= \quad p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid X\varphi \mid \varphi U \varphi \mid \varphi R \varphi \mid \exists x\varphi \mid \forall x\varphi \mid (a,x)\varphi$$

SL syntactically extends Linear-time Temporal Logic (LTL) [23] with two *strategy quantifiers*, the existential $\exists x$ and universal $\forall x$, along with an *agent binding* (a,x), where a is an agent and x a variable. Intuitively, these additional elements can be respectively interpreted as *"there exists a strategy x"*, *"for all strategies x"*, and *"bind agent a to the strategy associated with the variable x"*.

3 Classes of Strategies and Information

In this section, we recall some definitions of strategies and of agents' information.

In Definition 1, we have defined an indistinguishability relation for each agent involved in the model. When every $\sim_i$ is the identity relation, *i.e.*, $s \sim_i s'$ iff $s = s'$, we obtain a CGS with *perfect information* [2]. When the latter is not true, we assume that every agent i has *imperfect information* about the exact state of the system. That is, in any state s, i considers all states s' that are indistinguishable for i from s to be epistemically possible [13]. The indistinguishability relations are extended to histories in a synchronous, pointwise way, *i.e.*, histories $h, h' \in S^+$ are *indistinguishable* for agent $i \in Ag$, or $h \sim_i h'$, iff (i) $|h| = |h'|$ and (ii) for all $j \leq |h|$, $h_j \sim_i h'_j$.

Now, we have all the ingredients to present the different definitions of strategies. First, we start with a class of strategies in which the agents determine their actions by considering only the current state of the MAS.

Definition 4 (Memoryless Strategy) *A* memoryless strategy *for agent $i \in Ag$ is a function $f_i : S \to Act_i$ such that for each state $s \in S$, (i) $f_i(s) \in d(i,s)$; and (ii) $s \sim_i s'$ implies $f_i(s) = f_i(s')$.*

By Def. 4, any strategy for agent i must return actions that are enabled for i (i.e. condition (i)). Additionally, whenever two states are indistinguishable for i, the same action is returned (i.e. condition (ii)). This latter introduces the concept of *uniformity*, where an agent can select a strategy that adheres to its visibility. Notice that, for the case of perfect information, condition (ii) is satisfied by any function $f_i : S \to Act_i$.

The notion of memoryless strategy is considered too weak for an agent. For this reason, the concept of memoryfull strategy has been introduced.

Definition 5 (Memoryfull Strategy) *A* memoryfull strategy *for agent $i \in Ag$ is a function $f_i : S^+ \to Act_i$ such that for all histories $h, h' \in S^+$, (i) $f_i(h) \in d(i, last(h))$; and (ii) $h \sim_i h'$ implies $f_i(h) = f_i(h')$.*

As for the memoryless case, memoryfull strategies must adhere to the protocol function and indistinguishability relation.

Between these two approaches, i.e. memoryless and memoryfull, several different classes of bounded strategies have been proposed, including works by Ågotnes and Walther [1], Brihaye et al. [8], Vester [27], and Belardinelli et al. [6]. In this work, we focus our attention to natural strategies [18]. The idea behind natural strategies is to adopt the view of bounded rationality, and look at "simple" strategies in specification of agents' abilities. This notion has been introduced in both ATL and SL in the context of perfect [17] and imperfect information [5, 19]. Here, we focus on the definition provided in [19].

A natural strategies is an *ordered list of guarded actions*, i.e., sequences of pairs (ϕ_i, α_i) such that: ϕ_i is a condition, and α_i is an action. That is, a natural strategy is a rule-based representation in which the first rule whose condition holds in the current execution of the MAS is selected, and the corresponding action is executed.

With respect to the nature of the conditions, it is possible to define different classes of natural strategies. We start by recalling the notion of *uniform natural memoryless strategy*.

Definition 6 *In an uniform natural memoryless strategy for any agent a, the conditions are defined over epistemic formulas as follows:*

$$\psi ::= \top \mid K_a\varphi \mid \neg\psi \mid \psi \wedge \psi$$
$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid K_b\varphi$$

where p is an atomic proposition, b an agent, and K_i is the knowledge operator for any agent i. Intuitively, a formula $K_i\varphi$ can be interpreted as "the agent *i* knows φ".

So, we have formulas that are prefixed by K_a and then possibly combined by boolean operators. In other words, the formulas are always boolean conditions on a's knowledge. As discussed in [17], to define *natural memoryless strategy* in the perfect information case, we can replace epistemic formulas with boolean formulas only.

To improve the abilities of the agents, natural strategies have also been defined with recall. In particular, to evaluate properties over histories instead of states, a way to define conditions is to use regular expressions with the standard constructors $\cdot, \cup, *$ representing concatenation, nondeterministic choice, and finite iteration, respectively. Thus, to define a *natural strategies with recall* in the perfect information context, we can use regular expressions over boolean formulas. Similarly, to define a *uniform natural strategies with recall* in the imperfect information context, we can use regular expressions over epistemic formulas.

In the next section, we will provide the main model checking results for the above mentioned classes.

4 Model Checking Complexities

Here, we discuss the model checking complexities for ATL and SL in terms of memoryless, memoryfull, and natural strategies in the perfect and imperfect information context.

First, we can analyze ATL. As you can see in Table 1, in the worst case, that is imperfect information and perfect recall strategies, the problem becomes undecidable, while in the perfect information case, the problem is polynomial. An interesting point of ATL is that memoryless and memoryfull strategies are equivalent in the perfect information case. This is because ATL is too weak in expressive power.

To overcome this problem, there is ATL*. In this logic, strategic and temporal operators are decoupled to express more complex strategic objectives. As you can see in Table 2, the model checking complexity becomes solvable in polynomial space in the memoryless case, double-exponential time in

ATL	**perfect information**	**imperfect information**
memoryless	$PTIME$-complete [2]	Δ_2^P-complete [26]
memoryfull	$PTIME$-complete [2]	undecidable [12]

Table 1: Model checking complexities for ATL.

ATL*	**perfect information**	**imperfect information**
memoryless	$PSPACE$-complete [26]	$PSPACE$-complete [26]
memoryfull	$2EXPTIME$-complete [2]	undecidable [12]

Table 2: Model checking complexities for ATL*.

the memoryfull and perfect information case, and again undecidable in the memoryfull and imperfect information case. To address the latter problem, some works have either focused on an approximation to perfect information [3], developed notions of bounded memory [6], or employed hybrid techniques [14, 15]. Despite its expressiveness, ATL* suffers from the strong limitation that strategies are treated only implicitly in the semantics of strategic operators. This restriction makes the logic less suited to formalize several important solution concepts, such as the Nash Equilibrium [22].

To gain expressive power, we need to move to Strategy Logic. As shown in Table 3, the model checking problem becomes intractable in the memoryfull case. Given the relevance of this logic, several fragments have been proposed [21, 4]. Among others, we would like to mention Strategic Logic One Goal that has the same model checking complexity as ATL* but more expressive power, and Strategic Logic Simple Goal that has the same model checking complexity as ATL but more expressive power.

In all the above-mentioned logics, the model checking problem is undecidable in the worst case. In the last few years, a natural way to represent strategies has been studied. From the definition of natural strategies, two variants of ATL and SL have been proposed. In these variants, called NatATL [18] and NatSL [5], the strategic operators are equipped with graded modalities that represent the complexity of the natural strategies in achieving the temporal objectives. As you can see in Tables 4 and 5 the model checking problem has a complexity less than or equal $PSPACE$.

Can this approach solve all the problems related to model checking for MAS? Unfortunately (or fortunately for researchers), there are several open problems related to what we have summarized in this work. We will discuss some of them in the following section.

5 Future Directions

As promised throughout the paper, in this section, we will discuss some directions for future research. We will summarize these aspects with respect to the three main features related to the formal verification

SL	**perfect information**	**imperfect information**
memoryless	-	$PSPACE$-complete [20]
memoryfull	non-elementary [21]	undecidable [12]

Table 3: Model checking complexities for SL.

NatATL	**perfect information**	**imperfect information**
memoryless	Δ_2^P-complete [18]	Δ_2^P-complete [19]
with recall	*PSPACE*-complete [18]	*PSPACE*-complete [19]

Table 4: Model checking complexities for NatATL.

NatSL	**perfect information**	**imperfect information**
memoryless	-	*PSPACE*-complete [5]
with recall	-	*PSPACE*-complete [5]

Table 5: Model checking complexities for NatSL.

of multi-agent systems, namely: strategies, information, and logics.

About strategies: find the good representation. As we briefly discussed along the paper, both memoryless and memoryful strategies are not suitable choices. In fact, memoryless and memoryfull are too weak and strong, respectively, to describe agents' abilities. The weakness of memoryless strategies is a gain in terms of complexity, and the strength of memoryfull strategies is paid in terms of complexity. In addition, memoryfull strategies cannot be used to implement a model checker due to their domain over histories. As we mentioned earlier, there are some bounded versions between memoryless and memoryfull, but there is a lot of work to do to standardize the good choice. For instance, in the context of natural strategies, there are various aspects that require attention, such as defining the appropriate notion of complexity for these strategies. In the above-mentioned works, the authors have proposed the total size of the strategy as complexity, i.e., the overall complexity is the sum of all the symbols involved in the conditions. Is this the best way to define the complexity of a strategy? This is an open problem that needs to be investigated.

About information: perfect and imperfect is not enough. We believe there is significant work to be done in this area. For instance, it seems too reductive to consider only white (i.e. perfect information) and black (i.e. imperfect information) settings. We advocate the need to define a taxonomy for imperfect information. As discussed earlier in the paper, some approaches try to make some MAS with imperfect information decidable. However, the authors in [5, 3, 16] do not give a specific class of MAS with imperfect information that is decidable. Currently, only the class of hierarchical information has been proposed [25, 24] and analyzed in SL [7].

About logics: find the gap between ATL and SL. The two logics for the strategic reasoning discussed in this work suffer from two main problems on two different sides. On one hand, ATL has a good model checking complexity, but it cannot express several solution concepts such as the Nash Equilibrium. The strong limitation of ATL is that it treats strategies only implicitly in the semantics of its modalities. So, it is weak in expressiveness. On the other hand, SL is the more powerful logic for the strategic reasoning, but its model checking problem is not tractable. So, the full logic cannot have practical applications. The idea is to define a new logic for the strategic reasoning that can incorporate the positive features of ATL in terms of complexity and the good features of SL in terms of expressiveness. We understand that this is not a simple challenge, and finding a perfect trade-off between ATL and SL may be difficult. However, we see this need and want to go all the way on this point.

References

[1] Thomas Ågotnes & Dirk Walther (2009): *A Logic of Strategic Ability Under Bounded Memory*. *J. Log. Lang. Inf.* 18(1), pp. 55–77. Available at https://doi.org/10.1007/s10849-008-9075-4.

[2] Rajeev Alur, Thomas A. Henzinger & Orna Kupferman (2002): *Alternating-time temporal logic*. *J. ACM* 49(5), pp. 672–713. Available at https://doi.org/10.1145/585265.585270.

[3] Francesco Belardinelli, Angelo Ferrando & Vadim Malvone (2023): *An abstraction-refinement framework for verifying strategic properties in multi-agent systems with imperfect information*. *Artif. Intell.* 316, p. 103847. Available at https://doi.org/10.1016/j.artint.2022.103847.

[4] Francesco Belardinelli, Wojciech Jamroga, Damian Kurpiewski, Vadim Malvone & Aniello Murano (2019): *Strategy Logic with Simple Goals: Tractable Reasoning about Strategies*. In Sarit Kraus, editor: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, ijcai.org, pp. 88–94. Available at https://doi.org/10.24963/ijcai.2019/13.

[5] Francesco Belardinelli, Wojtek Jamroga, Vadim Malvone, Munyque Mittelmann, Aniello Murano & Laurent Perrussel (2022): *Reasoning about Human-Friendly Strategies in Repeated Keyword Auctions*. In Piotr Faliszewski, Viviana Mascardi, Catherine Pelachaud & Matthew E. Taylor, editors: *21st International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2022, Auckland, New Zealand, May 9-13, 2022*, International Foundation for Autonomous Agents and Multiagent Systems (IFAAMAS), pp. 62–71. Available at https://www.ifaamas.org/Proceedings/aamas2022/pdfs/p62.pdf.

[6] Francesco Belardinelli, Alessio Lomuscio, Vadim Malvone & Emily Yu (2022): *Approximating Perfect Recall when Model Checking Strategic Abilities: Theory and Applications*. *J. Artif. Intell. Res.* 73, pp. 897–932. Available at https://doi.org/10.1613/jair.1.12539.

[7] Raphaël Berthon, Bastien Maubert, Aniello Murano, Sasha Rubin & Moshe Y. Vardi (2021): *Strategy Logic with Imperfect Information*. *ACM Trans. Comput. Log.* 22(1), pp. 5:1–5:51. Available at https://doi.org/10.1145/3427955.

[8] Thomas Brihaye, Arnaud Da Costa Lopes, François Laroussinie & Nicolas Markey (2009): *ATL with Strategy Contexts and Bounded Memory*. In Sergei N. Artëmov & Anil Nerode, editors: *Logical Foundations of Computer Science, International Symposium, LFCS 2009, Deerfield Beach, FL, USA, January 3-6, 2009. Proceedings*, *Lecture Notes in Computer Science* 5407, Springer, pp. 92–106. Available at https://doi.org/10.1007/978-3-540-92687-0_7.

[9] Edmund M. Clarke & E. Allen Emerson (1981): *Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic*. In Dexter Kozen, editor: *Logics of Programs, Workshop, Yorktown Heights, New York, USA, May 1981*, *Lecture Notes in Computer Science* 131, Springer, pp. 52–71. Available at https://doi.org/10.1007/BFb0025774.

[10] E.M. Clarke, O. Grumberg, D. Kroening, D. Peled & H. Veith (2018): *Model Checking, second edition*. Cyber Physical Systems Series, MIT Press. Available at https://books.google.fr/books?id=0JV5DwAAQBAJ.

[11] E.M. Clarke, O. Grumberg, D. Peled & D.A. Peled (1999): *Model Checking*. The Cyber-Physical Systems Series, MIT Press. Available at https://books.google.fr/books?id=Nmc4wEaLXFEC.

[12] Catalin Dima & Ferucio Laurentiu Tiplea (2011): *Model-checking ATL under Imperfect Information and Perfect Recall Semantics is Undecidable*. *CoRR* abs/1102.4225. arXiv:1102.4225.

[13] R. Fagin, J.Y. Halpern, Y. Moses & M. Vardi (2004): *Reasoning About Knowledge*. A Bradford Book, MIT Press. Available at https://doi.org/10.7551/mitpress/5803.001.0001.

[14] Angelo Ferrando & Vadim Malvone (2022): *Towards the Combination of Model Checking and Runtime Verification on Multi-agent Systems*. In Frank Dignum, Philippe Mathieu, Juan Manuel Corchado & Fernando de la Prieta, editors: *Advances in Practical Applications of Agents, Multi-Agent Systems, and Complex Systems Simulation. The PAAMS Collection - 20th International Conference, PAAMS 2022, L'Aquila, Italy,*

July 13-15, 2022, Proceedings, *Lecture Notes in Computer Science* 13616, Springer, pp. 140–152. Available at https://doi.org/10.1007/978-3-031-18192-4_12.

[15] Angelo Ferrando & Vadim Malvone (2023): *How to Find Good Coalitions to Achieve Strategic Objectives*. In Ana Paula Rocha, Luc Steels & H. Jaap van den Herik, editors: *Proceedings of the 15th International Conference on Agents and Artificial Intelligence, ICAART 2023, Volume 1, Lisbon, Portugal, February 22-24, 2023*, SCITEPRESS, pp. 105–113. Available at https://doi.org/10.5220/0011778700003393.

[16] Angelo Ferrando & Vadim Malvone (2023): *Towards the Verification of Strategic Properties in Multi-Agent Systems with Imperfect Information*. In Noa Agmon, Bo An, Alessandro Ricci & William Yeoh, editors: *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2023, London, United Kingdom, 29 May 2023 - 2 June 2023*, ACM, pp. 793–801. Available at https://dl.acm.org/doi/10.5555/3545946.3598713.

[17] Wojciech Jamroga, Vadim Malvone & Aniello Murano (2017): *Reasoning about Natural Strategic Ability*. In Kate Larson, Michael Winikoff, Sanmay Das & Edmund H. Durfee, editors: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2017, São Paulo, Brazil, May 8-12, 2017*, ACM, pp. 714–722. Available at http://dl.acm.org/citation.cfm?id=3091227.

[18] Wojciech Jamroga, Vadim Malvone & Aniello Murano (2019): *Natural strategic ability*. *Artif. Intell.* 277. Available at https://doi.org/10.1016/j.artint.2019.103170.

[19] Wojciech Jamroga, Vadim Malvone & Aniello Murano (2019): *Natural Strategic Ability under Imperfect Information*. In Edith Elkind, Manuela Veloso, Noa Agmon & Matthew E. Taylor, editors: *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*, International Foundation for Autonomous Agents and Multiagent Systems, pp. 962–970. Available at http://dl.acm.org/citation.cfm?id=3331791.

[20] Bastien Maubert, Munyque Mittelmann, Aniello Murano & Laurent Perrussel (2021): *Strategic Reasoning in Automated Mechanism Design*. In Meghyn Bienvenu, Gerhard Lakemeyer & Esra Erdem, editors: *Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021, November 3-12, 2021*, pp. 487–496. Available at https://doi.org/10.24963/kr.2021/46.

[21] Fabio Mogavero, Aniello Murano, Giuseppe Perelli & Moshe Y. Vardi (2014): *Reasoning About Strategies: On the Model-Checking Problem*. *ACM Trans. Comput. Log.* 15(4), pp. 34:1–34:47. Available at https://doi.org/10.1145/2631917.

[22] R.B. Myerson (1991): *Game Theory: Analysis of Conflict*. Harvard University Press. Available at https://books.google.fr/books?id=E8WQFRCsNr0C.

[23] Amir Pnueli (1977): *The Temporal Logic of Programs*. In: *18th Annual Symposium on Foundations of Computer Science, Providence, Rhode Island, USA, 31 October - 1 November 1977*, IEEE Computer Society, pp. 46–57. Available at https://doi.org/10.1109/SFCS.1977.32.

[24] Amir Pnueli & Roni Rosner (1990): *Distributed Reactive Systems Are Hard to Synthesize*. In: *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, IEEE Computer Society, pp. 746–757. Available at https://doi.org/10.1109/FSCS.1990.89597.

[25] John H. Reif (1984): *The Complexity of Two-Player Games of Incomplete Information*. *J. Comput. Syst. Sci.* 29(2), pp. 274–301. Available at https://doi.org/10.1016/0022-0000(84)90034-5.

[26] Pierre-Yves Schobbens (2003): *Alternating-time logic with imperfect recall*. In Wiebe van der Hoek, Alessio Lomuscio, Erik P. de Vink & Michael J. Wooldridge, editors: *1st International Workshop on Logic and Communication in Multi-Agent Systems, LCMAS 2003, Eindhoven, The Netherlands, June 29, 2003*, *Electronic Notes in Theoretical Computer Science* 85, Elsevier, pp. 82–93. Available at https://doi.org/10.1016/S1571-0661(05)82604-0.

[27] Steen Vester (2013): *Alternating-time temporal logic with finite-memory strategies*. In Gabriele Puppis & Tiziano Villa, editors: *Proceedings Fourth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2013, Borca di Cadore, Dolomites, Italy, 29-31th August 2013*, *EPTCS* 119, pp. 194–207. Available at https://doi.org/10.4204/EPTCS.119.17.

Adaptive Application Behaviour for Robot Swarms using Mixed-Criticality

Sven Signer
Department of Computer Science
University of York
York, United Kingdom
sven.signer@york.ac.uk

Ian Gray
Department of Computer Science
University of York
York, United Kingdom
ian.gray@york.ac.uk

Communication is a vital component for all swarm robotics applications, and even simple swarm robotics behaviours often break down when this communication is unreliable. Since wireless communications are inherently subject to interference and signal degradation, real-world swarm robotics applications will need to be able handle such scenarios. This paper argues for tighter integration of application level and network layer behaviour, so that the application can alter its behaviour in response to a degraded network. This is systematised through the implementation of a mixed-criticality system model. We compare a static application behaviour with that of an application that is able to alter its behaviour in response to the current criticality level of a mixed-criticality wireless protocol. Using simulation results we show that while a static approach is sufficient if the network conditions are known a priori, a mixed-criticality system model is able to adapt application behaviour to better support unseen or unpredictable conditions.

1 Introduction

Swarm robotics platforms usually rely on wireless communications, which by their nature can be unreliable and exhibit unpredictable timing characteristics. The standard approach is to use best-effort protocols with sufficient redundancy so that all important traffic can be conveyed. Conventional protocols such as WiFi, ZigBee, and Bluetooth offer good throughput and robustness, but they rely on features like random backoff and retransmissions which can be disqualifying when attempting to build high-criticality systems that rely on timing correctness. There are similarly a range of swarm-oriented protocols such as Glossy [7], and the related Low-Power Wireless Bus [6] and Blink [17] which all remain topology-agnostic by flooding the network with packets and retransmissions in order to maximise connectivity at the expense of throughput and power-efficiency.

Accordingly there has been a more recent trend towards protocols which are timing-aware, such as WirelessHART [5] and AirTight [3]. These approaches require a priori information about the system, but compensate by allowing for greater timing confidence. They still, however, suffer from the inherent unreliability of wireless links.

This work combines ideas from these timing-aware protocols and from the real-time systems domain to argue for application-level adaptive behaviour to support reliability in the presence of errors and unreliable communications. If a swarm robotics platform employs a wireless protocol that can provide feedback as to whether communication reliability requirements are currently being met, the application can be designed to adapt its own behaviour in order to recover connectivity if packet delivery rates fall below a predefined threshold. This paper argues for the use of a mixed-criticality approach, in which the application switches between two behaviour modes depending on the network conditions.

A. Ferrando, R.C. Cardoso (Eds.): Third Workshop on
Agents and Robots for reliable Engineered Autonomy 2023 (AREA'23)
EPTCS 391, 2023, pp. 71–82, doi:10.4204/EPTCS.391.9

2 Application Adaptivity and Mixed Criticality Systems

A common claim is that robot swarms achieve fault tolerance and redundancy through the size of the swarm. However as is well-understood in the research, communication failures are particularly problematic and can actually be compounded by increasing swarm size [2]. Swarm applications must be designed with the reliability of communications as a priority. Robotic swarm programming models like Buzz [10] allow for developers to discuss higher level concepts such as moving groups of robots as one, but frameworks which tie reliability requirements from the application developer to the reliability state of the running system are less formalised.

This work argues that in order to address this issue, a swarm robotics system with unreliable communications should be viewed as a Mixed Criticality System (MCS), and that such a characterisation can be used to explicitly guide the system's behaviour at times of overload or error. The MCS model is a commonly-deployed system model in the domains of high-integrity or safety-critical systems, and is used to provide guarantees on the behaviour and timing of the system in the presence of unreliability. Informally, the developer provides a description of their system, along with an "importance" for each task, and the guarantees that are required to support these tasks at each criticality level. If the system enters a state when it can no longer guarantee the "important" parts, then a graceful degradation is codified into the model.

The original formulation of MCS [15] was aimed at CPU scheduling problems and defined the system to have two criticality levels: *LO* and *HI*. Each task/process/job is designated either *LO* criticality or *HI* criticality: *HI* tasks perform safety-critical functions and are required for safety assurance, whereas *LO* tasks are less important and may occasionally fail or miss their deadlines. The response time analysis of CPU scheduling requires tasks to be assigned an estimate of their worst-case execution time (WCET) a priori, but determining this value proves difficult in practice. If a task's WCET comes from estimation or measurement it may be optimistic (i.e. not a true worst-case time). This could result in important tasks missing their deadlines if the true WCET at runtime is larger than the assumed value. A task's WCET may also come from a safe, analytical approach which is guaranteed to cover the worst-case but in practice this might be very pessimistic. This could lead to the application designer falsely believing that the system is not schedulable without greater hardware resources, which may in turn be infeasible for other reasons (cost, energy use, etc).

The key insight of the MCS model is that the optimistic execution times will be correct most of the time, and the true worst-case is usually only seen in rare situations. Tasks are therefore assigned two estimates of their worst-case execution time, a *LO* WCET and a *HI* WCET. A system can proceed at the *LO* criticality level most of the time, assuming that tasks will exhibit their expected *LO* WCETs. However, when any task exceeds its *LO* WCET, the system enters the *HI* criticality state and stops executing any *LO* tasks. This means that to guarantee the safety of a system in the presence of unreliability, it is necessary to prove only that the *HI* tasks will not exceed their *HI* WCET times. The application adapts its behaviour based on the current state of the system by dropping less important activities and focusing on only the most important system guarantees.

Prior work has applied mixed-criticality scheduling to wireless networks by assuming that it is the level of interference (i.e. the assumed maximum number of retransmissions required within a time period) that varies by criticality level [3]. This paper takes this idea and applies it to both the transmission of packets in a wireless communications swarm, but also to the behaviour of that swarm. Instead of treating criticality as just a feature of the scheduling and communications layer, this work argues that the application itself can respond to such changes usefully, based on developer-provided criticality requirements, in order to preserve the performance of the overall swarm. Such a characterisation can be

exploited to allow for applications which assume communications will mostly operate well, and so behave accordingly, but can respond appropriately when, for whatever reason, communicating tasks start to fail to meet their deadlines. In Section 6 we describe an example of such an application, with Section 6.1 describing how the application modifies its own behaviour based on feedback from the network layer.

3 Prior Work

In existing research, unrealistic assumptions are often made about the reliability of wireless communications, such as assuming perfect transmission within a given radius. Prior research has shown that imperfect communication results in swarm behaviours breaking down [12], and so it is therefore not always appropriate to simply ignore the communications medium and rely on TCP to transmit data 'eventually'.

The MCS system model has been applied to industrial wireless communications [16] to show that such an approach can give more reliable timing bounds by adapting the network routing based on the system's criticality level. This work provides better timing bounds, but is restricted to considering the communications layer only.

Our previous work has argued for the use of an MCS wireless protocol in swarm robotics applications [13]. The system initially operates in *LO* criticality mode, and if the level of successful message transmissions degrades past some threshold this is detected and the network switches to *HI* criticality mode. The result of this change is typically to drop or deprioritise less important traffic. While this allows the application designer to preserve some behavioural elements in the presence of partial network degradation, it is inherently limited to preserving a subset of the full behaviour, and will eventually fail if network conditions continue to degrade.

The AirTight protocol [3] considered in that work, being purely a point-to-point protocol, is impractical for real swarm robotics applications. We therefore introduce a model (Section 4) and simulation implementation (Section 5) for a new protocol that can provide mixed criticality behaviour over broadcast transmissions.

The work in this paper extends this idea to consider the how the application's *overall* behaviour can adapt to criticality. Specifically, we enable an application to observe the network layer's criticality level, so that the application can modify its own behaviour in response to current network conditions (Section 6.1).

4 Communication Model

This paper argues for a closer integration of an application implementation with a mixed-criticality network MAC layer, specifically by allowing the application to be aware of the MAC layer's current criticality mode. This allows the application to detect when the network is no longer able to guarantee reliable message delivery due to communication faults, enabling it to adjust its behaviour to prevent further degradation. We compare the effect allowing a robot to use this criticality mode information to locally adjust its cohesion factor against simply using a static cohesion factor, as described in Section 6.1.

We consider a network of N homogeneous robot nodes in which all transmissions are assumed to be broadcasts that should reliably reach each other node. The network provides multiple buffers of configurable priority and criticality such that the application designer can determine the order of transmissions. In each transmission slot, the node chooses to transmit the first message from the highest priority non-empty buffer at, or above, the current criticality level.

The network layer observes the number of "failed" transmissions, defined as transmissions after which it cannot prove that all nodes have received its last message. If the number of failed transmissions within a busy-period exceed a predefined threshold, the network switches to *HI* criticality mode. Messages in *LO* criticality buffers are discarded if the node enters *HI* criticality mode. Time sensitive messages in *HI* criticality buffers can optionally be assigned a maximum time-to-live (TTL), such that they expire and are not retransmitted after a given time period has elapsed since they were first queued.

5 Communication Implementation

In order to implement the behaviour described by the model in Section 4, we introduce an implementation of AirTight [8] modified to use broadcast transmission rather than point-to-point links. AirTight is a slot-based real-time, mixed criticality protocol which can guarantee the time-sensitive transmission of a set of traffic flows using ahead of time analysis under a given fault rate assumption.

The network runs using time-division multiplexing over 10ms time slots. There is a periodic slot table assigned a priori, such that the slot table is of length N and each node has exactly one exclusive transmission slot. It is assumed that background clock synchronisation takes place such that nodes agree on the current slot.

Each node maintains a bit-field of length $N-1$, where each bit encodes whether the node successfully received a transmission in the last occurrence of the corresponding slot in the slot table (excluding the node's own transmission slot). This bit-field is included in the header of each transmission as a type of delayed acknowledgement, such that each receiving node can determine if its own last transmission was received by the sender.

For each transmission buffer, the node maintains a further bit-field of length $N-1$ in which each bit encodes whether confirmation of successful delivery has been received from a corresponding other node. Upon reception of a frame, a node can thereby set the transmitting node's bit in its last transmission buffer's bitfield if the received header indicates that the transmission was received. Once all bits have been set, the message at the head of the buffer has been delivered to all other nodes. The message can then be removed from the buffer and the bit-field is cleared.

The node keeps two counters: a counter of the length of the current busy-period, and a counter of the number of retransmissions. The busy-period counter is incremented on all transmissions, whilst the retransmission counter is incremented whenever the node rebroadcasts a frame that had previously been transmitted. If the number of retransmissions exceeds a given threshold, the node switches to *HI* criticality mode.

During its assigned transmission slot, a node broadcasts a single frame from the highest priority non-empty buffer at or above the current criticality level. If no such frame exists the counters are reset and, if the node was in *HI* criticality mode, it switches back to *LO*. The node then broadcasts a no-op frame such that the bit-field used for delayed acknowledgements is always transmitted. The real-time timing analysis of this implementation is future work but can be based on the structure of the original AirTight analysis.

The implementation of the simulation plugin, communications layer, and other artefacts associated with this paper can be found in our code repository [1].

[1] https://github.com/yorkrobotlab/argos3-airtight

6 Exploration Application

We consider an autonomous wireless swarm robotics platform in which a swarm of robots should collaborate to explore an unknown area. The environment is partitioned into a two-dimensional grid. Robots should visit each cell in the grid and detect if there is an object present in that cell, building a map of clear/occupied cells in the area. It is assumed that obstacles are stationary, such that it is unimportant when a robots visits the cell, or whether a cell is visited multiple times by one or more robots. Each robot has its own local copy of the map, which it should complete for all accessible cells[2]. Robots can communicate the sensed values for a given cell over the network, allowing other robots to insert this value into their map without needing to visit the cell.

For a concrete instantiation we consider a set of 10 Pi-Puck [9] robots exploring a 6x6m grid of 10x10cm square cells. Pi-Puck robots only have simple infrared range-finder sensors that determine the distance to the closest object within a short range, but are unable to determine the nature of any detected object. Therefore, to avoid falsely detecting other robots as an obstacle in the environment, robots must maintain a minimum separation. Each robot is assumed to be able to determine its own location, which it must communicate to the other robots to avoid such near-collisions. Robots cannot store their complete location history, so they cannot retrospectively determine that incorrect data may have been sensed where position messages have been lost or delayed. These position messages are therefore intuitively subject to soft real-time constraints, since robots must learn the positions of other robots before the minimum separation distance is violated.

The robot behaviour is loosely inspired by an existing algorithm [14], but adapted to the much simpler Pi-Puck robots and to much reduced communication ability. It is implemented by picking and driving towards a target cell, which is always chosen as one of the nine cells within a 3x3 grid centred on the robot's current position. A new target cell is chosen once the current target is reached, or the robot encounters an obstacle such that it deems the current target to be unreachable. The target cell is selected as the cell for which the sum of the following weights results in the smallest value.

- Diagonal movements are assigned a weight of +1.
- The cell the robot is currently in is assigned a linearly increasing weight the longer it remains in that cell, and the cell it was in immediately previously is assigned a weight of +1.
- An avoidance score of +1000 if the cell is known to contain an obstacle.
- An attraction score of -10 if it is an unexplored cell.
- A separation score of 400000, 200000, 100000, 4, 1, 0.25, or 0.1 respectively if the distance to the closest target cell of another robot, counted as a number of cells, between 0 and 6.
- An alignment score, given by the dot product of the robot's forwards vector and the vector from it's current position to the potential target.
- An attraction score equal to the distance to the closest reachable unexplored cell, counted as a number of cells.
- Optionally, a cohesion force of $8d^3$, where d is the distance to the computed centroid of all robots using the most recent position information the robot has received. This force is applied according to the rules defined in Section 6.1.

[2]The positioning of obstacles in the environment may render some cells inaccessible.

6.1 Robot Cohesion

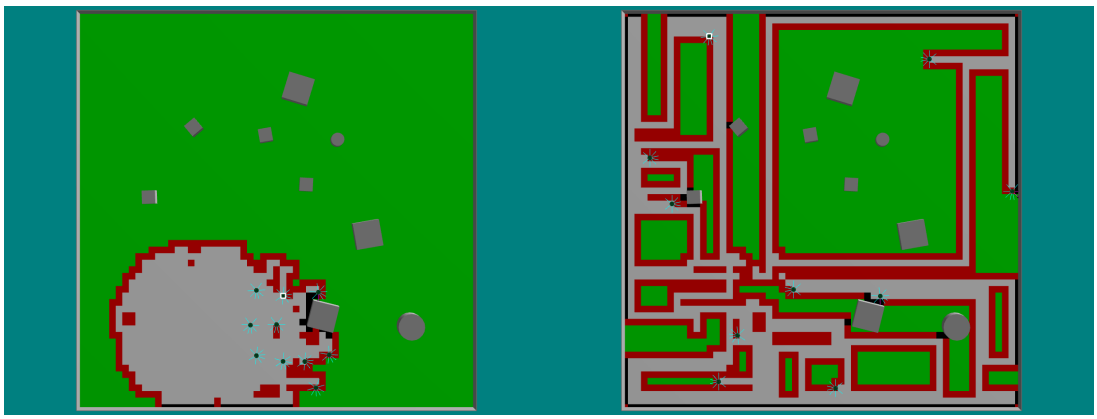


Figure 1: Visualisation of exploration state after 95s when applying constant cohesion (left) compared with no cohesion (right) under perfect network conditions. Explored cells are shown in grey if empty or black if containing/bordering an obstacle; unexplored cells are shown in red when bordering explored cells, else in green.

The effect of adding a cohesion element to the target cell selection depends on the environmental factors encountered by the robots. Under perfect communication, robot cohesion reduces the overall application performance. This can be intuitively understood by considering that robots that disperse maximally will not block each other and can greedily explore new cells. The closer the robots are pulled together, the more often robots may need to change direction (for example by revisiting already explored cells) in order to avoid violating the minimum separation constraint. As shown in Figure 1, this may result in robots towards the rear of a group being surrounded by already explored cells.

With a communication model that deteriorates with distance, the performance of a solution where robots disperse decreases, since robots are unable to receive the sensing information of other robots. In extreme cases, this could effectively result in each robot needing to explore the entire area. There is therefore a trade-off in which some amount cohesion is useful to preserve communication, but too much cohesion decreases performance by limiting the exploration ability. The optimal level of cohesion depends on the properties of the wireless medium, which in a real scenario may not be known a priori.

In this paper, we argue for application adaptation based on the state of the network, by applying cohesion weighting on target cell selection using a mixed-criticality approach. While the network layer is in *LO* criticality mode the robots can disperse to maximise their exploration potential, before being pulled back towards each other if/when the network changes to *HI* criticality mode. Once the network has recovered, the robots can then resume exploring. This results in an equilibrium that allows the robots to adapt their behaviour to the encountered conditions. We compare the effects of the following four ways of applying the cohesion weighting:

- No cohesion: The cohesion weight is completely disabled.
- Constant cohesion: A cohesion weight is always applied.
- Half cohesion: The cohesion weight is always applied but is computed using half of the true centroid distance.
- Mixed criticality: The application applies a cohesion weighting when the network protocol has been in *HI* criticality mode at any point within the last three seconds.

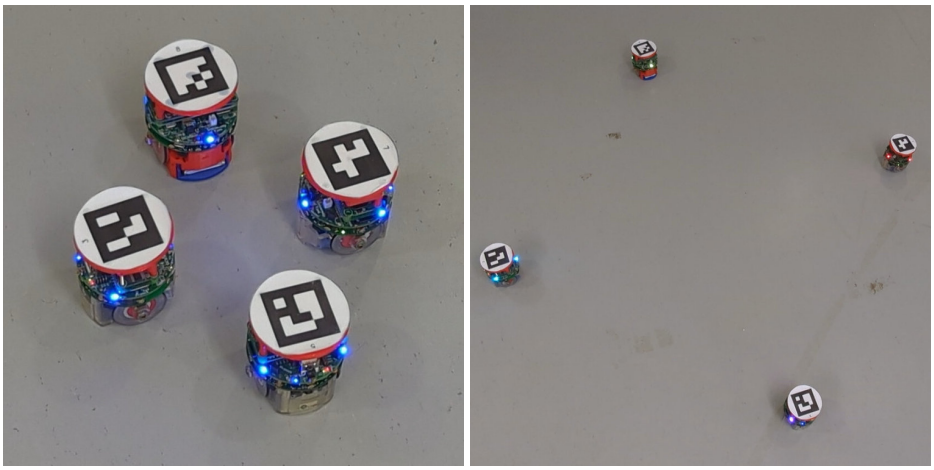


Figure 2: The running "physicalised simulation" of an earlier experiment on the Pi-Puck robots. The simulation shows the LED synchronisation breaking down as the robot separation increases.

7 Simulation Setup

7.1 Simulation Framework

Our simulation setup is based on the popular ARGoS robot simulator [11]. An extended version of a custom networking plugin [13] implements the network simulation capabilities. Each simulation step is assumed to correspond to a network level transmission slot in which a robot can attempt to send or receive a single message. A simulated transmission model controls which messages are successfully received.

In order to partially validate our simulation framework, we have implemented a "physicalised simulation" of our earlier experiments using real Pi-Puck robots (Figure 2). By this we mean an implementation on physical robots, but where some key components are still simulated. Specifically, we use Pi-Puck robots communicating over IEEE 802.15.4 provided by XBee modules, whilst simulating the packet loss by artifically discarding some messages according to the packet delivery rates provided by the transmission model. Since this still relies on a simulated transmission model, the results between the full simulation and physicalised simulation are very similar. Due to the logistical challenges of running larger scale experiments with physical robots, we have not yet implemented such a physicalised simulation for the current experiments.

7.2 Robot Configuration

Each robot is configured with two network buffers, for position messages and cell status message. A position message contains the robots target location, while a cell status message encodes a single cell to be either clear or containing an obstacle. Since the focus of these experiments is on the effect of application level behaviour changes based on the network criticality level, we configure both buffers as *HI* criticality to prevent criticality changes having an impact at the network level. Positional data is time sensitive to ensure the minimum robot separation is preserved, and so this buffer is configured with the higher priority. Positional messages are set to be retried for a maximum of 0.8s to prevent old position data filling up transmission buffers, whilst cell status messages are set not to expire since these are not time sensitive.

7.3 Arena Generation

We randomly generate 100 simulated arenas, comparing the exploration performance of each cohesion and transmission model configuration combination across these arenas. The robots are placed first, by distributing them within a 1.8m by 1.8m square centred around a random point in the area, whilst ensuring each robot placed at least 30cm away from the next closest robot. We then distribute up to 17 obstacles across the arena, again requiring a minimum separation from each other obstacle and each robot starting position.

7.4 Transmission Model

Packet delivery is determined by a simulated transmission model in which successful or unsuccessful delivery is determined independently for each transmission and each potential receiving node. The packet delivery rate is assumed to be fixed to a maximum of 95% for distance of less than 0.5m, after which the packet delivery rate is inversely proportional to the square distance between the nodes, subject to an additional constant scaling factor (k). By modifying this scaling factor, the effect of the packet delivery rate on the application behaviour can be observed.

$$\text{PDR} = \frac{0.95}{1 + (k \cdot X)^2}$$

Where:

$$X = \begin{cases} 0 & d \leq 0.5 \\ d - 0.5 & d > 0.5 \end{cases}$$

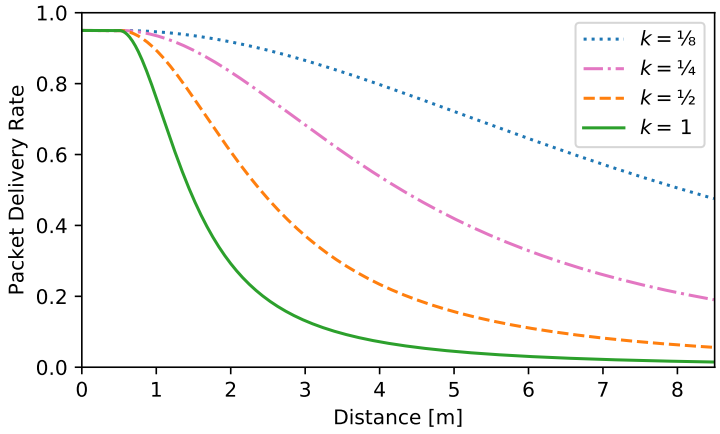


Figure 3: Transmission model defining packet reception probability in relation to distance, for scaling factors $k = \frac{1}{8}$, $k = \frac{1}{4}$, $k = \frac{1}{2}$, and $k = 1$.

This is a simple model that cannot capture the true complexity of real-world wireless communications. Prior research [1, 4] has shown that observed packet delivery rates do not correlate as strongly with distance in real-world experiments. Nonetheless, observed results broadly show that there exists a safe distance cutoff up to which the communication is generally reliable. Baccour et al. describes this as the "connected" region [1], which is followed by "transitional" and "disconnected" regions where packet delivery first becomes intermittent and then mostly unsuccessful. This fits well with the mixed criticality system model, which inherently assumes that the system will operate under its optimistic assumptions most of the time, before encountering some kind of state change that requires rectification.

The implementation here does not rely on specific details of the transmission model, and the criticality response merely assumes that moving closer to other nodes is likely to improve packet reception rates. We therefore believe the simulation results should be broadly applicable regardless of the selected transmission model.

8 Results

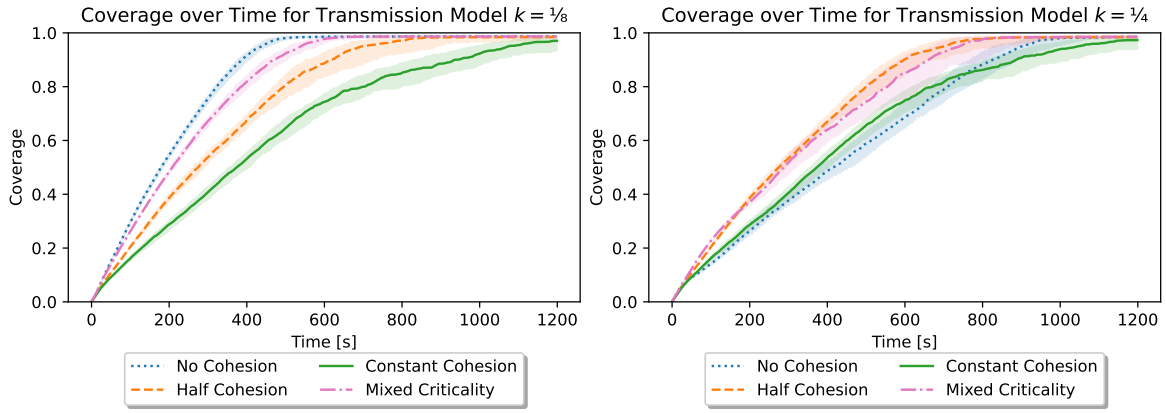


Figure 4: Coverage over time across the four cohesion modes for transmission model scaling factors $k = \frac{1}{8}$ and $k = \frac{1}{4}$. Line shows median value; shaded region shows interquartile range across 100 simulation runs.

With the transmission model scaling factor configured for the most gradual dropoff in packet delivery rates, $k = \frac{1}{8}$, the simulation results in Figure 4 shows that not applying a cohesion force results in the highest performance. At this low dropoff rate the robots can adequately communicate across the entire arena such that there is no advantage to moving as a cohesive group. The mixed criticality configuration can spend a significant proportion of its runtime in *LO* criticality mode (where no cohesion is applied) and thus displays better performance than the half cohesion or constant cohesion configurations.

When the transmission model scaling factor is increased to $k = \frac{1}{4}$ a maximal dispersion of the nodes begins to impede communication between the nodes, creating an advantage to applying some level of cohesion. The mixed criticality and half cohesion configurations perform similarly to each other. The no cohesion configuration starts to suffer from the aforementioned communication issues, while the performance of the constant cohesion configuration is still reduced from applying a stronger cohesive force than necessary.

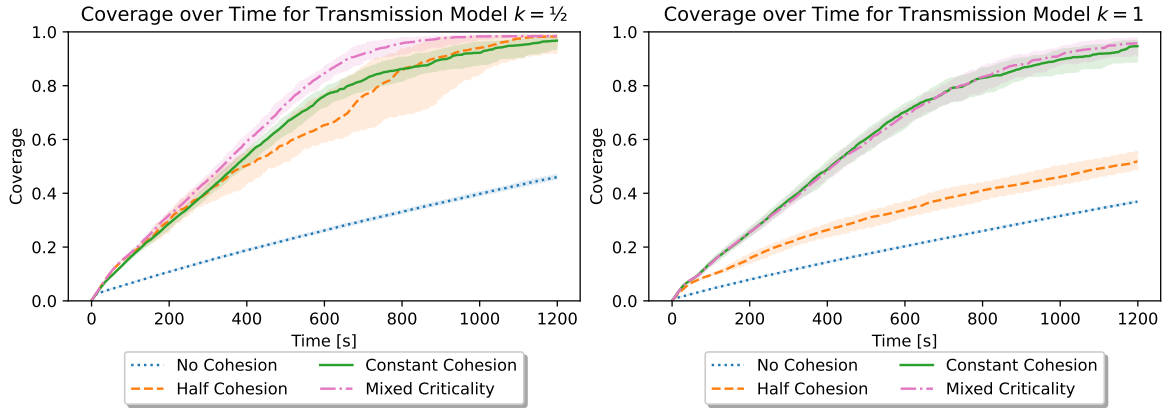


Figure 5: Coverage over time across the four cohesion modes for transmission model scaling factors $k = \frac{1}{2}$ and $k = 1$. Line shows median value; shaded region shows interquartile range across 100 simulation runs.

A further increase of the transmission model scaling factor to $k = \frac{1}{2}$, shown in Figure 5, causes the no cohesion configuration to break down. The performance of the half cohesion configuration is also decreased, showing a fairly wide spread depending on runtime circumstances. The mixed criticality configuration is able to adapt to the conditions, yielding the highest performance for this transmission model.

At the maximum tested dropoff rate, with the transmission model scaling factor set to $k = 1$, the mixed criticality mode and constant cohesion modes perform very similarly, with the low packet delivery rates causing the mixed criticality configuration to spend large proportions of its time in *HI* criticality mode where cohesion is applied. The half cohesion and no cohesion modes both show poor performance, with the half cohesion mode no longer resulting in a sufficiently tight formation to maintain communication between the nodes.

The constant cohesion configuration is mostly unaffected by the transmission model's scaling factor. While this results in comparatively good performance when communications are limited to short ranges, it cannot take advantage of long communication ranges under good radio conditions. The half cohesion configuration provides comparatively good performance for the two middling transmission models, but neither takes full advantage of long communication ranges nor does it cope with very short communication ranges.

The mixed criticality mode is better able to adjust to different communication dropoff rates, and can do so in a way that does not require the application layer to understand the specific issues that are happening at the networking layer. While a static cohesion factor can provide similar levels of performance for any given transmission model, this requires the transmission characteristics to be known beforehand and to remain static. For a real-world application this is unlikely to be the case. A static cohesion factor therefore requires the application designer to make a tradeoff between performance under good networking conditions and reliability under network degradation. A mixed criticality approach avoids this tradeoff by allowing the application to observe the true conditions at runtime.

9 Limitations and Future Work

The communication protocol presented in this paper is developed to study the effect of adjusting application behaviour based on a network layer criticality level. Compared to a complete protocol it is lacking in several aspects, most importantly the absence of formal timing analysis that can provide guarantees on the network performance. In future work we intend to develop a real-time mixed-criticality protocol that supports network layer and application integration while being suitable for swarm robotics and providing such timing guarantees.

We also intend to further study the two-way relationship between application behaviour and network MAC layer. In the implementation provided in this paper, the application is aware of the current network layer criticality level, and has been programmed that cohesion should be applied only when the network layer is in *HI* criticality mode. Beyond this simple rule, however, it could be imagined a more intelligent application could attempt to predict the effect of its future behaviour on the network. This could allow the application to either modify its planned behaviour to avoid network issues, or warn the network layer such that it could prepare for a drop in packet delivery rates.

10 Conclusion

In this paper we have presented a mixed-criticality approach to swarm robotics application behaviour. Mixed criticality is widely applied in real-time CPU scheduling and network protocols, but has so far not been widely applied to swarm robotics. The parameters controlling a robot's behaviour may have different optimal values depending on the conditions encountered by the robot. Communication conditions at runtime can vary from those expected by the application designer in ways that are often opaque to the application, placing stress on the network and leading to a loss of real-time performance. A mixed-criticality approach allows a systematic way for the application to both define what is considered of higher and lower importance, and then respond in a way that prioritises resources appropriately.

Our simulation results show that a swarm robotics application using a mixed-criticality approach to adjust its behaviour to match the encountered conditions can be made more robust than one that uses a static configuration. In future work we intend to develop mixed-criticality network protocols targeted at swarm robotics applications and further study the integration of network and application behaviour.

References

[1] Nouha Baccour, Anis Koubâa, Maissa Ben Jamâa, Denis do Rosário, Habib Youssef, Mário Alves & Leandro B. Becker (2011): *RadiaLE: A framework for designing and assessing link quality estimators in wireless sensor networks*. *Ad Hoc Networks* 9(7), pp. 1165–1185, doi:10.1016/j.adhoc.2011.01.006.

[2] Jan Dyre Bjerknes & Alan FT Winfield (2013): *On fault tolerance and scalability of swarm robotic systems*. In: *Distributed Autonomous Robotic Systems: The 10th International Symposium*, Springer, pp. 431–444, doi:10.1007/978-3-642-32723-0_31.

[3] Alan Burns, James Harbin, Leandro Indrusiak, Iain Bate, Rob Davis & David Griffin (2018): *AirTight: A Resilient Wireless Communication Protocol for Mixed-Criticality Systems*. In: *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*, pp. 65–75, doi:10.1109/RTCSA.2018.00017.

[4] Alberto Cerpa, Naim Busek & Deborah Estrin (2003): *SCALE: A tool for Simple Connectivity Assessment in Lossy Environments*. Technical Report. Available at `https://escholarship.org/uc/item/2g49z78g`.

[5] International Electrotechnical Commission et al. (2016): *Industrial Networks—Wireless Communication Network and Communication Profiles—WirelessHART (IEC 62591: 2016)*. *IEC: Geneva, Switzerland* 3, pp. 1–1043.

[6] Federico Ferrari, Marco Zimmerling, Luca Mottola & Lothar Thiele (2012): *Low-Power Wireless Bus*. In: *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, SenSys '12, Association for Computing Machinery, New York, NY, USA, p. 1–14, doi:10.1145/2426656.2426658.

[7] Federico Ferrari, Marco Zimmerling, Lothar Thiele & Olga Saukh (2011): *Efficient network flooding and time synchronization with Glossy*. In: *Proceedings of the 10th ACM/IEEE International Conference on Information Processing in Sensor Networks*, pp. 73–84.

[8] J. Harbin, A. Burns, R. I. Davis, L. S. Indrusiak, I. Bate & D. Griffin (2019): *The AirTight Protocol for Mixed Criticality Wireless CPS*. *ACM Trans. Cyber-Phys. Syst.* 4(2), doi:10.1145/3362987.

[9] Alan G. Millard, Russell Joyce, James A. Hilder, Cristian Fleşeriu, Leonard Newbrook, Wei Li, Liam J. McDaid & David M. Halliday (2017): *The Pi-puck extension board: A raspberry Pi interface for the e-puck robot platform*. In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 741–748, doi:10.1109/IROS.2017.8202233.

[10] Carlo Pinciroli & Giovanni Beltrame (2016): *Buzz: An extensible programming language for heterogeneous swarm robotics*. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, pp. 3794–3800, doi:10.1109/IROS.2016.7759558.

[11] Carlo Pinciroli, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick Ducatelle, Mauro Birattari, Luca Maria Gambardella & Marco Dorigo (2012): *ARGoS: a Modular, Parallel, Multi-Engine Simulator for Multi-Robot Systems*. *Swarm Intelligence* 6(4), pp. 271–295, doi:10.1007/s11721-012-0072-5.

[12] Mark Selden, Jason Zhou, Felipe Campos, Nathan Lambert, Daniel Drew & Kristofer S. J. Pister (2021): *BotNet: A Simulator for Studying the Effects of Accurate Communication Models on Multi-Agent and Swarm Control*. In: *2021 International Symposium on Multi-Robot and Multi-Agent Systems (MRS)*, pp. 101–109, doi:10.1109/MRS50823.2021.9620611.

[13] Sven Signer, Alan G. Millard & Ian Gray (2022): *Mixed-Criticality Wireless Communication for Robot Swarms*. In: *Proceedings of the 9th International Workshop on Mixed Criticality Systems*, pp. 20–25. Available at `https://wmc2022.github.io/assets/WMC_2022_Proceedings.pdf`.

[14] Vu Phi Tran, Matthew A. Garratt, Kathryn Kasmarik, Sreenatha G. Anavatti & Shadi Abpeikar (2022): *Frontier-led swarming: Robust multi-robot coverage of unknown environments*. *Swarm and Evolutionary Computation* 75, p. 101171, doi:10.1016/j.swevo.2022.101171.

[15] Steve Vestal (2007): *Preemptive Scheduling of Multi-criticality Systems with Varying Degrees of Execution Time Assurance*. In: *28th IEEE International Real-Time Systems Symposium (RTSS 2007)*, pp. 239–243, doi:10.1109/RTSS.2007.47.

[16] Changqing Xia, Xi Jin, Linghe Kong & Peng Zeng (2017): *Bounding the Demand of Mixed-Criticality Industrial Wireless Sensor Networks*. *IEEE Access* 5, pp. 7505–7516, doi:10.1109/ACCESS.2017.2654483.

[17] Marco Zimmerling, Luca Mottola, Pratyush Kumar, Federico Ferrari & Lothar Thiele (2017): *Adaptive Real-Time Communication for Wireless Cyber-Physical Systems*. *ACM Trans. Cyber-Phys. Syst.* 1(2), doi:10.1145/3012005.

Autonomous Systems' Safety Cases for use in UK Nuclear Environments*

Christopher R. Anderson
Department of Computer Science
University of Manchester
Manchester, UK
chris.anderson@manchester.ac.uk

Louise A. Dennis
Department of Computer Science
University of Manchester
Manchester, UK
louise.dennis@manchester.ac.uk

An overview of the process to develop a safety case for an autonomous robot deployment on a nuclear site in the UK is described and a safety case for a hypothetical robot incorporating AI is presented. This forms a first step towards a deployment, showing what is possible now and what may be possible with development of tools. It forms the basis for further discussion between nuclear site licensees, the Office for Nuclear Regulation (ONR), industry and academia.

1 Introduction

Autonomy lends itself to activities in the nuclear industry. Traditional, verifiable [8, 7] robotic systems lack the ability to perform many desirable tasks, however, this could be made possible by the use of Artificial Intelligence (AI) technologies, which include machine learning (ML) (sub-symbolic) and formally verifiable logical reasoning (symbolic).

The deployment of these systems in nuclear environments necessitates that a complete and coherent set of arguments is made which demonstrates that the activity to be undertaken (by the autonomous system) is adequately safe, by utilising a **claims**, **arguments** and **evidence** trail [10] (CAE).

However, in the view of the UK nuclear industry, AI is not a mature technology that can easily be shown to meet the well-established, conservative approaches to safety. It can appear difficult or indeed impossible to construct a safety case, in contrast to other domains where AI is in the process of being adopted (e.g. automotive). Whilst the nuclear consequence may be higher, the environment in which the activity is undertaken is generally well constrained. We have proposed a route that, with some thought and good engineering practices, can enable a safety case to be constructed. This builds on a white paper providing principles for the development and assurance of autonomous systems [9]. An outline architecture and safety case [1] has been developed to demonstrate this.

2 The Robot and Task

In the absence of a suitable and timely deployment, a hypothetical robot and scenario have been used based very loosely on the A2I2 Lilypad ASV [15] deployed to survey a nuclear waste storage pond to assist with remediation tasks. These ponds hold the spent fuel from a reactor in water which has a depth of more than 10 m. The robot is programming using a combination of symbolic and sub-symbolic AI.

*This work is supported by the EPSRC, through the Robotics and AI for Nuclear (RAIN) Hub (EP/R026084, EP/W001128/1, EP/R026084/1). Thanks are due to the Office for Nuclear Regulation (ONR) and Sellafield Ltd. for input into and comments upon the Safety Case this paper describes.

A. Ferrando, R.C. Cardoso (Eds.): Third Workshop on
Agents and Robots for reliable Engineered Autonomy 2023 (AREA'23)
EPTCS 391, 2023, pp. 83–88, doi:10.4204/EPTCS.391.10

A Robotics with Autonomy/AI Safety Case framework for our hypothetical robot can be found online [1][1]. In this paper we discuss salient features of this safety case that would enable similar safety cases to be created, for real deployments in UK Nuclear environments.

3 Safety Cases in the UK Nuclear Industry

In this section we discuss the key features of creating a safety case for use in a UK Nuclear environment, with particular focus on those aspects relevant to the deployment of autonomous robotic systems.

Identification of hazards: The building of a safety case starts with the holistic identification of hazards, analysed for the robot within its application task and environment, whether the robot is pre-existing or proposed. Analyses of robot behaviour that do not account for the specific task and location are not sufficient on their own for creating a safety case. Hazards should be identified for all of the robot's lifecycle phases and tasks. This can usually be achieved by applying the site licensee's high level principles. Normal and abnormal operations should also be analysed.

Developing risk mitigation strategies: Mitigation strategies are derived through one or more optioneering studies (the selection of the best option from a set of alternatives), where the objective is to reduce risks to As Low As Reasonably Practicable (ALARP). This process should identify the benefits and disadvantages of any proposed solution in comparison to other options. It may include an argument that deployment of a prototype technology that may have not be preferable for the specific task has long term benefits that out-weigh the immediate disadvantages. The aim is to either reduce the consequence (which is typically fixed) or reduce the likelihood of any unwanted outcome. Defence in depth principles are applied to each hazard. In the case of autonomous robots, most of the robotics design efforts lie in an 'occurrence barrier' regime – the protection and mitigation barriers are usually external to the robot.

This paper assumes that an optioneering study has determined that an autonomous robot is necessary.

Tolerability and ALARP: The principles of tolerability and ALARP are key to acceptance of the autonomous robot to perform a particular task at a particular location on a nuclear site. Tolerability is expressed in The Tolerability of Risks from Nuclear Power Stations (TOR), 1992 [2] which defines risks which are so high they are unacceptable unless there are exceptional circumstances. The requirement for risks to be ALARP (take all measures to reduce risk where doing so is reasonable [12]) is fundamental and applies to all activities within the scope of the Health and Safety at Work Act, 1974 (HSWA). ALARP can be achieved through applying established 'Relevant Good Practice' (RGP) and standards and only in cases where these are inappropriate is a cost/benefit analysis used. Tolerability and ALARP are understood through the diagrams in TAG 094 [11], Figure 3.

Safety functions A Safety Function (SF) is a mechanism for implementing mitigation strategies. An SF can be realised as either: a function which is diverse, independent and segregated from the control system, sensors, control and actuators of the robot (a guard); the control function (system) itself or a combination of guard and control system. The guard and/or control system must: lend itself to design, implementation, verification & validation to the degree required by the hazard analysis and the safety requirements (functional and non-functional) imposed on it; meet all non-probabilistic requirements; meet the probabilistic claim required by the hazard analysis and the safety requirements imposed on it.

Safety Functions (SF) are realised by Structures, Systems and Components (also known as Safety Instrumented Functions (SIF)), using appropriate standards and RGP. e.g. IEC 61508 [8]. Production Excellence (PE), one of two "legs" used to substantiate safety to the regulator, demonstrates good control

[1]Note that, this Safety Case was generated using an ASCE Academic Licence and therefore cannot be used for commercial purposes

of the robot's development and verification lifecycle. The second leg, Independent Confidence Building Measures (ICBM), requires that the final validated software (in its target hardware deployment) and the testing programme be independently checked and can include statistical testing and static code analysis. In general, it is always preferrable to use the simplest, most effective mitigation and this should be demonstrated in the optioneering study and ultimately in the safety case.

4 The Safety Case

It is widely recognised that the use of autonomy/AI in robotics in high integrity applications can be difficult to justify. In particular challenges in analysing the behaviour of such systems form a challenge to the creation of a robust safety case. The definition of the activity and formal identification and analysis of the hazards (the Preliminary Hazard Analysis (PHA)) forms the primary, and probably most crucial, task in such a robot's development lifecycle. The PHA ensures that the safety system is not over engineered or, indeed, unnecessary. In particular, we assert that it is not always necessary to introduce new mitigations simply because autonomy is involved.

Whilst there are potentially a number of hazards which the robot we consider here could either encounter or initiate, including propeller splash, being irretrievable due to complete robot system failure and explosion due to hydrogen evolution at the surface of the pond, the safety case presented here focuses on collision with either the pond structure or its contents and we assume that the PHA has resulted in the identification of this as a hazard that requires mitigation.

Our safety case has been generated as an example framework, based on a hypothetical robot operating in a nuclear material storage pond. Consequently, in several places the safety case contains placeholder nodes since this documentation is not available for the hypothetical robot. The safety case provides two examples of how a 'Collision' hazard may be mitigated (referred to below as Method 1 and Method 2). It is unlikely that both methods would be needed to show that the risk has been reduced to be ALARP and, therefore, one could be deleted. In the event that both are needed to show defence in depth and diversity, then Method 1 and 2 would need to be restructured into one set of CAE. We are reasonably certain that Method 1 can be used now, and it is feasible that Method 2 can be used in the medium term, providing the necessary verification tools and methodologies are developed.

The 'Top View' provides the best entry point for the safety case [1]. Here the primary claim (node C1) "Robot is adequately safe... to provide..." is elaborated by the main elements: Optioneering study and justification of the choices made; establishment and analysis of the hazards; the derivation of safety requirements; compliance of the design with safety requirements; the use within a safety management system; and compliance with standards and RGP. The hazards are separated into four groups: Nuclear (radiological), conventional, physical, and cyber security. The compliance of the design with safety requirements is further broken down and eventually reaches the claim that the safety risk is mitigated and managed. At this claim node the arguments split into Method 1 (Using an Engineered SSC in the form of a guard) and Method 2 (Using verifiable AI technologies).

Robot hazard and mitigations: The collision hazard for our hypothetical robot was determined to have a consequence of < 2 mSv (Sievert - the SI (International System of Units) unit which represents the stochastic risk to health of ionising radiation), based on experience of this environment. However, normally specific site licensee experience would be applied here. The principle of ALARP means that the same strategies can be applied in the range of 2 to 20 mSv (below the BSL for on-site workers). We deem this hazard to be mitigated by the use of an Occurrence Barrier which we describe below.

4.1 Occurrence barrier (SIF)

This safety case proposes two approaches to mitigation of the collision hazard, as follows:

Diverse guard ([1], Method 1: This SIF is comprised of cat's whiskers surrounding the robot which actuate microswitches and in turn, open safety relay contacts, removing the delivery of power to the propellers. This is relatively crude and does not allow for subtle control. e.g. an 'intelligent' robot may be able to avoid the collision by steering away from the obstacle or reversing away from it. From a safety perspective it has the advantage of the ability to make a highly deterministic and probabilistic claim because of its simplicity. A minimal claim is necessary to ensure that the SIF (guard) is not demanded too often thereby causing excessive deterioration of the components or encouraging the operator to ignore or disable the safety feature.

Rules based reasoning ([1], Method 2: This SIF is comprised of the intelligent (safety) control system itself and a collision sensor with a software component, as shown in Figure 1. At higher Safety Integrity Level (SIL) (lower Class in nuclear) this may be completely independent of any ML-based components involved in implementing the base functionality of the robot. We assume that navigation utilises an ML image classifier and that other ML components may have been involved in developing navigation planning systems. It is not feasible to place a claim on such systems at this time, therefore, a second independent (software based) sensor, such as an ultrasonic sensor or LIDAR, is used to detect collisions on which a safety claim is made. This and the differential control equations that control motion have been constructed from traditional software and hardware and developed using a safety lifecycle process such as that described in IEC 61508 [8] at the appropriate SIL for which PE and ICBM can be demonstrated.

The detection of the potential collision and actuation of the consequential avoidance action is then mediated by a 1-out-of-2 voting system [13], between the image classifier and the collision sensor, providing output to a separate intelligent control system. The intelligent control system here is constructed from a rules based reasoning (RBR) architecture (such as those supported by the MCAPL framework [5, 6]). This strategy allows for a more subtle control where the robot can take evasive action. RBR systems controlling autonomous and robotic systems work in a cyclic fashion first sensing the environment, then applying rules to pick an appropriate action and then acting before returning to sensing. They can generally be transformed into decision tree structures but the representation as a set of rules with specified applicability conditions is more compact and so has advantages in terms of code readability. The applicability conditions and output actions of the rules themselves define a state space which can be explored using techniques such as model-checking [4] to check for desirable properties. These properties can be expressed in a variety of temporal and probabilistic logics and tools exist to allow requirements to be captured and expressed in these logics. Typically model-checking operates on an abstract model of the system to be verified, but so called *program model-checkers* exist which apply the same process to the actual code. For instance the MCAPL Framework uses a version of the JPF model-checker [14] to verify the behaviour of RBR systems written in a number of agent programming languages. To achieve this it executes the actual code in order to determine the next state of the system. Some property languages allow timing constraints to be expressed, such as UppAal [3] which has a property specification language but it is not a program model-checker and we are not aware of any program model-checkers that allow timing constraints to be expressed. However, a timer and runtime monitors (and watchdog timers (WDT)) could be provided to ensure that the collision avoidance takes place in a timely fashion and triggering some default safe behaviour (e.g. the complete stop of method 1) if it did not. The full architecture is shown in Figure 1. To achieve this both a deterministic (complying with standards and RGP) and probabilistic (from, for example, a Failure Modes, Effects and Criticality Analysis) claim must

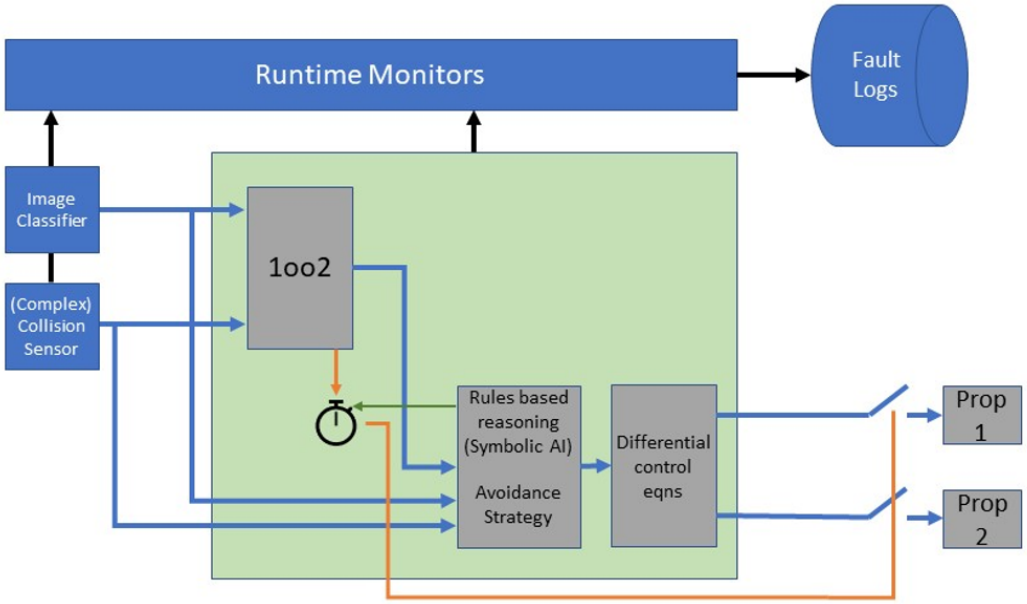


Figure 1: Rules based reasoning SIF (Method 2).

be placed on the intelligent control system. RBR enables the use of formal verification (i.e. the application of a formal mathematical proof) of the collision avoidance. Its implementation is on an end to end basis. i.e. from requirements through to hardware deployment.

At present the necessary technology at a high enough Technology Readiness Level (TRL) does not exist to realise Method 2. The MCAPL Framework which enables verifiable RBR for agent programs is academic software and depends upon the underlying use of the Java programming language. Obviously it would be preferable to use a safety critical realtime operating system and programming language. However, the barriers to this method are primarily the availability of appropriate languages and toolsets, not a lack of methodology.

5 Conclusion

The safety case described here and found in [1] provides a first attempt at an argument for deploying a robot with autonomy on a nuclear site in the UK. It introduces two approaches: one of which is possible now but does not allow all of the benefits of the autonomy to be realised and a second which identifies a potential route to incorporating these benefits. The safety case shows that autonomy in and of itself is not necessarily a barrier to the deployment of autonomous robots in UK nuclear environments and, in fact, that existing approaches to analysing hazards, devising mitigations and establishing safety claims are still applicable where autonomy is involved. However we have also shown that there remains a gap in situations where we might want the autonomy itself to form a part of the SF and claim. Method 2 outlines a potential approach to bridging this gap.

References

[1] Christopher R Anderson (2022): *RAIN Robot operation SC Strawman*, doi:10.48420/19512010. Available at https://autonomy-and-verification.github.io/events/strawman/RAIN%20Robot%20operation%20SC%20Strawman.htm. Accessed 31st August 2022.

[2] Geoff Ballard, Donald Broadbent, Roger Clarke, Mr H J Dunster, Bev Littlewood & David Slater (1992): *The tolerability of risk from nuclear power stations*. Technical Report, HSE/ONR. Available at https://www.onr.org.uk/documents/tolerability.pdf.

[3] Johan Bengtsson, Kim G. Larsen, Fredrik Larsson, Paul Pettersson & Wang Yi (1995): UPPAAL — *a Tool Suite for Automatic Verification of Real–Time Systems*. In: *Proc. of Workshop on Verification and Control of Hybrid Systems III*, *Lecture Notes in Computer Science* 1066, Springer, pp. 232–243, doi:10.1007/BFb0020949.

[4] E. M. Clarke, O. Grumberg & D. Peled (1999): *Model Checking*. MIT Press.

[5] Louise A. Dennis (2017): *Gwendolen Semantics: 2017*. Technical Report ULCS-17-001, University of Liverpool, Department of Computer Science.

[6] Louise A Dennis (2018): *The MCAPL Framework including the Agent Infrastructure Layer and Agent Java Pathfinder*. *The Journal of Open Source Software* 3(24), doi:10.21105/joss.00617.

[7] IEC (2006): *60880:2006, Nuclear power plants - Instrumentation and control systems important to safety - Software aspects for computer-based systems performing category A functions*.

[8] IEC (2010): *61508-1, Edition 2.0, Functional safety of electrical/electronic/programmable electronic safety-related systems, Part 1: General requirements*.

[9] Matt Luckcuck, Michael Fisher, Louise Dennis, Steve Frost, Andy White & Doug Styles (2021): *Principles for the Development and Assurance of Autonomous Systems for Safe Use in Hazardous Environments paper, White Paper*, doi:10.5281/zenodo.5012322.

[10] ONR (2014): *Safety Assessment Principles For Nuclear Facilities*. Technical Report 1, ONR. Available at https://www.onr.org.uk/saps/saps2014.pdf.

[11] ONR (2019): *TAG 094 Categorisation of safety functions and classification of structures, systems and components*. Technical Report, Office for Nuclear Regulation. Available at https://www.onr.org.uk/operational/tech_asst_guides/ns-tast-gd-094.pdf.

[12] ONR (2020): *TAG 005 Guidance on the Demonstration of ALARP (As Low As Reasonably Practicable)*. Technical Report, Office for Nuclear Regulation. Available at www.onr.org.uk/operational/tech_asst_guides/ns-tast-gd-005.pdf.

[13] N. Storey (1996): *Safety-Critical Computer Systems*. Harlow: Addison-Wesley.

[14] Willem Visser & Peter C. Mehlitz (2005): *Model Checking Programs with Java PathFinder*. In: *Proc. 12th International SPIN Workshop*, *Lecture Notes in Computer Science* 3639, Springer, p. 27, doi:10.1007/11537328_5.

[15] S Watson, G Cross, P Green, J Parsons, P Routledge, N Tudor & T Wood (2021): *A2I2: Autonomous Aquatic Inspection and Intervention in Nuclear Storage Ponds*. In: *2021 ANS Winter Meeting and Technology Expo*. Available at https://pure.manchester.ac.uk/ws/portalfiles/portal/206634504/A2I2_ANS_Paper_Final.pdf.

Rollout Heuristics for Online Stochastic Contingent Planning

Oded Blumenthal
Software and Information Systems Engineering, Ben Gurion University, Israel
odedblu@post.bgu.ac.il

Guy Shani
Software and Information Systems Engineering, Ben Gurion University, Israel
shanigu@bgu.ac.il

Partially observable Markov decision processes (POMDP) are a useful model for decision-making under partial observability and stochastic actions. Partially Observable Monte-Carlo Planning is an online algorithm for deciding on the next action to perform, using a Monte-Carlo tree search approach, based on the UCT (UCB applied to trees) algorithm for fully observable Markov-decision processes. POMCP develops an action-observation tree, and at the leaves, uses a rollout policy to provide a value estimate for the leaf. As such, POMCP is highly dependent on the rollout policy to compute good estimates, and hence identify good actions. Thus, many practitioners who use POMCP are required to create strong, domain-specific heuristics.

In this paper, we model POMDPs as stochastic contingent planning problems. This allows us to leverage domain-independent heuristics that were developed in the planning community. We suggest two heuristics, the first is based on the well-known h_{add} heuristic from classical planning, and the second is computed in belief space, taking the value of information into account.

1 Introduction

Many autonomous agents operate in environments where actions have stochastic effects, and important information that is required for obtaining the goal is hidden from the agent. Agents in such environments typically execute actions and sense some observations that result from these actions. Based on the accumulated observations the agents can better estimate their current state and decide on the next action to execute. Such environments are often modeled as partially observable Markov decision processes (POMDPs) [28].

POMPD models allow us to reason about the hidden state of the system, typically using a belief state – a distribution over the possible environment states. The belief state can be updated given the executed action and the received observation. One can compute a policy, a mapping from beliefs to actions, that dictates which action should be executed given the current belief. Many algorithms were suggested for computing such policies [25].

However, in larger environments, it often becomes difficult to maintain a belief state, let alone compute a policy for all possible belief states. In such cases, one can use an online re-planning approach, where after every action is executed, the agent computes which action to execute next. Such online approaches replace the lengthy single policy computation which is done offline, before the agent begins to act, with a sequence of shorter computations, which are executed online, during execution, after each action [21].

POMCP [27] is such an online replanning approach, extending the UCT algorithm for fully observable Markov decision processes (MDPs) to POMDPs. POMCP operates by constructing online a search tree, interleaving decision and observation nodes. The root of the tree is a decision node. Each decision

A. Ferrando, R.C. Cardoso (Eds.): Third Workshop on
Agents and Robots for reliable Engineered Autonomy 2023 (AREA'23)
EPTCS 391, 2023, pp. 89–101, doi:10.4204/EPTCS.391.11

node has an outgoing edge for every possible action, ending at an observation node. Then, the outgoing edges from an observation node denote the possible observations that result from the incoming action. The agent computes a value for each node in the tree, and then, the agent can choose, from the root node, the action associated with the edge leading to the highest value child node.

To evaluate the value of leaf nodes, POMCP executes a random walk in belief space, known as a rollout, where the agent selects actions from some rollout policy to construct a trajectory in belief space and obtain an estimation of the quality of the leaf node. Clearly, this evaluation is highly dependant on the ability of the rollout policy to reach the agent goals. In complex problems, obtaining the goal may require a lengthy sequence of actions [16], and until the goal is reached, no meaningful rewards are obtained. Indeed, practitioners that use POMCP often implement complex domain-specific heuristics for the rollout policy.

In this paper we focus on suggesting domain-independent heuristics for rollout policies. We leverage work in automated planning, using heuristics defined for classical and contingent planning problems [7, 8, 6]. We thus represent POMDP problems in a structured manner, using boolean facts to capture the state of the environments. This allows both for a compact representation of large problems, compared with standard flat representations that do not scale, as well as the ability to use classical planning heuristics.

We begin by suggesting using the well-known h_{add} heuristic for choosing rollout actions [2]. This heuristic searches forward in a delete relaxation setting, until the goal has been reached. Then, the value of an action is determined by the number of steps in the delete relaxation space following the action, required for obtaining the goal.

Next, we observe that any state-based rollout policy is inherently limited in its ability to evaluate the missing information required for reaching the goal, and hence, provide some estimate as to the value of information [12] of an action. We hence suggest a multi-state rollout policy, where actions are executed on a set of states jointly, and observations are used to eliminate states that are incompatible with the observed value. We show that this heuristic is much more informed in domains that require complex information-gathering strategies.

For an empirical evaluation, we extend domains from the contingent planning community with stochastic effects. We evaluate our heuristics, comparing them to random rollouts, showing that they allow us to provide significantly better behavior.

2 Background

We now provide the required background on POMDPs, contingent planning, domain-independent heuristics, and the POMCP algorithm.

2.1 POMDPs

A goal-oriented partially observable Markov decision process (POMDP) is a tuple $\langle S, A, tr, \Omega, O, G\rangle$ [3]. S is a set of states; A is a set of actions. $tr : S \times A \times S \to [0,1]$ is the transition function, i.e., $tr(s,a,s')$ is the probability that when executing action a at state s we would reach state s'. Ω is the set of observations the agent can obtain. $O : S \times A \times \Omega \to [0,1]$ is the observation function, such that $O(s,a,o)$ is the probability of observing o when a was performed and led to state s. G is a set of goal states.

Because the state of a POMDP is partially observable, the agent typically does not know what the true underlying state of the world is. Hence, it can maintain a *belief state* b, which is a distribution over S,

i.e., $b(s)$ is the likelihood that s is the current state. When a goal belief is reached, that is $sum_{s\in G}b(s) = 1$, then the agent is sure that it is at a goal state, and the execution terminates.

A solution to a POMDP, called a *policy*, is a function that assigns an action to every belief state. The optimal policy minimizes the expected cost, i.e., the expected number of steps before a goal belief has been reached.

2.2 POMCP

The Partially Observable Monte-Carlo Planning (POMCP) online re-planning approach uses a Monte Carlo tree search (MCTS) approach to select the next action to execute [27]. At each re-planning episode POMCP constructs a tree, where the root node is the current belief state. Then, POMCP runs forward simulations, where a state is sampled from the current belief, and actions are chosen using an exploration strategy that initially selects actions that were not executed a sufficient amount of times, but gradually moves to select the seemingly best action. Observations in the simulations are selected based on the current state-action observation distribution.

When reaching a leaf node, POMCP begins a so-called rollout. This rollout is designed to provide a value estimate for the leaf, based on some predefined rollout policy. The value of the leaf is updated using the outcome of the rollout, and then the values of the nodes along the branch that were visited during the simulation are updated given their descendants. Obviously, the values of all nodes in the tree are hence highly dependent on the values obtained by the rollout policy.

POMCP is an anytime algorithm, that is, it continues to run simulations until a timeout, and then returns the action that seems best at the root of the search tree.

POMCP was designed for large problems. Hence, POMCP does not maintain and update a belief state explicitly. Instead, POMCP uses a particle filter approach, where a set of states is sampled at the initial belief, and this set is progressed through the tree.

2.3 Contingent Planning under Partial Observability

A partially observable contingent planning problem is a tuple: $\pi = \langle P, A_{act}, A_{sense}, \varphi_I, G\rangle$ [10, 1, 4, 24]. P is a set of facts, A_{act} is a set of actuation actions, and A_{sense} is a set of sensing actions. φ_I is a formula describing the set of initially possible states. For ease of exposition, we will assume that φ_I is a conjunction of facts, disjunctions of facts, or *oneof* clauses over facts, specifying that exactly one fact in the clause holds. A state s assigns truth values to all $p \in P$. G is a formula over P defining goal conditions.

A *belief-state* is a set of possible states. The initial belief state, $b_I = \{s : s \models \varphi_I\}$ is the set of initially possible states.

An actuation action $a \in A_{act}$ is a pair, $\{pre(a), eff(a)\}$, where $pre(a)$ is a set of fact preconditions, and $eff(a)$ is a set of pairs (c,e) denoting conditional effects. We use $a(s)$ to denote the state that is obtained when a is executed in state s. A sensing action $a \in A_{sense}$ is a pair, $\{pre(a), obs(a)\}$, where $pre(a)$ is as above, and $obs(a)$ is a set of facts in P whose value is observed when a is executed. We denote by $obs(a,s)$ the values of the observed facts when a is executed at state s. This separation to actuation and sensing actions is only for ease of exposition, and our methods apply also to actions that both modify the state of the world and provide an observation.

Preconditions allow us to restrict our attention only to applicable actions. An action is applicable in a belief b is all possible states in b satisfy the preconditions of the action. Obviously, one can avoid specifying preconditions for actions, allowing for actions that can always be executed, as is typically the

case in flat POMDP representations. However, in domains with many actions, preconditions are a useful tool for drastically limiting the amount of actions that should be considered.

2.4 Regression-based Belief Maintenance

Updating a belief can be costly. Alternatively, one can avoid the computation of new formulas representing the updated belief, by maintaining only the initial belief formula, and the history — the sequence of executed actions and sensed observations [5]. When the agent needs to query whether the preconditions of an action or the goal hold at the current node, the formula is regressed [20] through the action-observation sequence back towards the initial belief. Then, one can apply SAT queries to check whether the query formula holds. We now briefly review the regression process for deterministic actions. This approach can be highly useful for larger POMDPs, complementing the particle filter approach used in POMCP.

First, let us consider the regression of an actuation action a that does not provide an observation. Let ϕ be a propositional formula and a a *deterministic* actuation action. Let $c_{a,l}$ denote the condition under which l is an effect of a, and that $a(s)$ satisfies l iff either $s \models c_{a,l}$ or $s \models l \wedge \neg c_{a,\neg l}$. Hence, we define the *regression* of ϕ with respect to a as:

$$rg_a(\phi) = pre(a) \wedge \phi_{r(a)} \tag{1}$$

$$\phi_{r(a)} = \text{replace each literal } l \text{ in } \phi \text{ by } c_{a,l} \vee (l \wedge \neg c_{a,\neg l}) \tag{2}$$

Now, consider a sensing action and an ensuing observation. Suppose we want to validate that ϕ holds following the execution of $a \in A_{sense}$ in some state s given that we observed $obs(a) = o$. Thus, we need to ensure that following a, if l holds then ϕ holds. That is:

$$rg_{a,o}(\phi) = rg_a(obs(a) = o \rightarrow \phi) \tag{3}$$

Regression maintains the equivalence of the formula [20, 5]. For any two formulas ϕ_1 and ϕ_2 we have:

1. $\phi_1 \equiv \phi_2 \Rightarrow rg_{a,o}(\phi_1) \equiv rg_{a,o}(\phi_2)$
2. $\phi_1 \equiv \phi_2 \Rightarrow rg_a(\phi_1) \equiv rg_a(\phi_2)$

Hence, we can produce a regression over formulas, and compare the regressed formulas, making conclusions about the original formulas.

The regression can be recursively applied to a sequence of actions and observations (history) h as follows:

$$rg_{h+(a,o)}(\phi) = rg_h(rg_{a,o}(\phi)); \quad rg_{\varepsilon,\varepsilon}(\phi) = \phi \tag{4}$$

where ε is the empty sequence. This allows us to perform a regression of a formula through an entire plan, and analyze the required conditions for a plan to apply.

In addition, the regression mechanism of [5] maintains a cached list of fluents $F(n)$ that are known to hold at node n, given the action effects or observations. Following an actuation action a, $F(a(n))$ contains all fluents in $F(n)$ that were not modified by a, as well as effects of a that are not conditioned on hidden fluents. For a sensing action revealing the value l, $F(a(n,l)) = F(n) \cup \{l\}$. During future regression queries, when a value at a particular node becomes known, e.g. when regressing a later observation, it is added to $F(n)$. All fluents p such that $p \notin F(n) \wedge \neg p \notin F(n)$ are said to be hidden at n. The cached list is useful for simplifying future regressed formulas.

3 Related Work

Augmenting MCTS methods with heuristics in the context of fully observable MDPs was previously suggested. [14] describe an MCTS tree based approach that uses planning based heuristics. The PROST planner [13] uses a heuristic for estimating the value of states. The DP-UCT approach [26] use a planning heuristic based on deep learning for the rollout phase. We are not aware of previous attempts to adapt these approaches to POMDPs.

There were several extensions suggested for POMCP [16]. For example [17] considers dynamic environments, and [11] consider non-linear dynamics. All these methods rely on rollouts, and can hence leverage the rollout strategies that we suggest here.

POMCPOW [30] extends POMCP to the challenge of solving POMDPs with continuous state, action, and observation spaces. It constructs the search tree incrementally to explore additional regions of the observation and action spaces. It also requires a rollout policy to evaluate the utility of leaf nodes. Our rollout strategies relay on classical planning approaches which are discrete, and it is hence unlikely that our methods can be extended to continuous domains.

DESPOT [29, 32, 18] is an online POMDP solver based on tree search, similar to POMCP. DESPOT uses a different strategy than the UCB rule for constructing the tree, designed to avoid the overly greedy nature of POMCP exploration strategy. DESPOT also requires a so-called default policy to evaluate the utility of a leaf in the tree, and the authors stress the importance of a strong default policy to improve the convergence. Thus, our methods can be directly applied to DESPOT as well. We chose here to focus on POMCP rather than DESPOT, because POMCP is a simpler method, which allows us to better focus on the importance of the heuristic function, independent of the effect of the various augmentations that DESPOT adds on top of POMCP.

[22] suggest a method called PSG to evaluate the proximity of states to the goal. They suggest using PSG in several places within POMCP including rollouts. PSG assumes that states are defined in a factored manner using state features, and computes a function from features to the goal using subgoals. In essence, their approach can be considered as a type of heuristic, which is highly related to the concept of landmarks in classical planning [19]. Representing the POMDP as a stochastic contingent planning problem, as we do, allows us to use any heuristic developed in the planning community, and can hence be considered to be an extension of PSG.

Similarly, [31] also find it difficult to provide good rollout strategies to compute the value of POMCP leaves. Focusing on a robotics motion planning domain, they suggest SVE, state value estimator, that attempts to evaluate the utility of a state directly.

[15] also focus on the need to use heuristics for guiding search in POMDPs. They focus on RTDP-BEL [4], an algorithm that runs forward trajectories in belief space to produce a policy. They show that using a heuristic can significantly improve RTDP-BEL. They use domain specific heuristics, and as such, our domain independent approach can also be applied to their methods.

4 POMCP for Stochastic Contingent Planning

We focus here on goal-oriented POMDP domains specified as stochastic contingent planning problems. We now define this concept formally.

We define a stochastic formula ψ to be a set of options. Each option o is a conjunction of facts, and is associated with a probability $pr(o) \in (0,1)$ such that $\sum_o pr(o) = 1$. One can sample a single option from the stochastic formula, given the distribution defined by $pr(o)$.

A stochastic contingent planning problem is a tuple $\pi = \langle P, A_{act}, A_{sense}, \varphi_I, pr_I, G \rangle$, where P, A_{sense}, φ_I, G are as in a deterministic contingent planning problem. pr_I is a stochastic formula defining probabilities over the initial values of some unknown facts. For each action $a \in A_{act}$, the formula defining the effects of a may contain stochastic formulas, capturing stochastic effects. We denote by $a(s)$ the distribution over next states given that a was executed at s.

This definition does not support noisy observations, however, this is not truly a limitation. One can compile a noisy observation into a deterministic observation over an artificial fact whose value changes stochastically. Consider, for example, a sensor that nosily detects whether there is a wall in front of a robot. Instead of noisily observing whether there is a wall, we can deterministically detect a green light that is lit when the sensor (stochastically) detects a wall. That is, we can observe the green light without noise, but the green light is only noisily correlated with the existence of a wall.

Algorithm 1 describes the POMCP implementation for stochastic contingent planning problems. When the agent needs to act, it calls Search. Search repeatedly samples a state (line 3-4) and simulates forward execution given this state is the true underlying system state.

We do not maintain or update a belief state. Instead, we use regression over the history of executed actions and sensed observations. The Search procedure hence samples a state s from the initial belief state, given the initial probability distribution pr_I (line 3). Then, the agent advances the sampled state through the history to obtain a current state s' (line 4).

The Simulate procedure is recursive. We first check whether the current tree node is a goal belief. This is done be regressing the negation of the goal formula $\neg G$ through the history of the current node. For goal beliefs, the value is 0, and we can stop.

Our implementation of POMCP also stops deepening the tree after a predefined threshold Max-Tree-Depth. If that threshold is reached (line 12), we run a Rollout to compute an estimation for the cost of reaching the goal from this node (line 13).

In line 15 we check whether this node has already been expanded, and if not, we compute its children. We do so only for applicable actions whose preconditions are satisfied in the current belief (line 17). Again, this is computed using regression over the history.

We now select an action a using the UCT exploration-exploitation criterion (line 21), and sample a next state and an observation (lines 22-27). We call Simulate recursively in line 28.

Lines 29-32 update the value of the current node. Lines 28,29 update the counters for the executed action and received observation. Line 31 computes the value for the action as a weighted average over all observations. Line 32 computes the value of the node as the minimal cost among all actions. Our value update, which we empirically found to be more useful, is different than the original POMCP, which uses incremental updates, and more similar to the value update in DESPOT [18].

The Rollout procedure receives as input the current simulated state s, as well as a set B of states (particles) in the node from which the rollout begins. B is used by some of our rollout heuristics, as we explain below. The rollout executes actions given the heuristic rollout policy until the goal has been reached, or a maximal number of steps has been reached.

5 Domain Independent Heuristics for POMCP

We now describe the main contribution of this paper — two domain independent rollout heuristics that leverage methods developed in the automated planning community, using the structure specified in the stochastic contingent planning problem.

Algorithm 1: POMCP for Stochastic Contingent Problems

Search(h)
  while *timeout not reached* **do**
    $s \sim \varphi_I, pr_I$
    $s' \leftarrow$ apply h to s
    $Simulate(s', root, o)$
Simulate($s, n, depth$)
  add s to $n.b$
  $count(n) \leftarrow count(n) + 1$
  if G *is satisfied in* $n.history$ **then**
    $V(n) \leftarrow 0$
    return
  if $depth > MaxTreeDepth$ **then**
    $V(n) += Rollout(s, n.b)$
    return
  if n *is a leaf node* **then**
    for $a \in A_{act} \cup A_{sense}$ **do**
      if $pre(a)$ *are satisfied at* $n.history$ **then**
        Add child $n.a$ to n
        for $o \in obs(a)$ **do**
          Add child $n.a.o$ to $n.a$
  $a \leftarrow argmin_a Q(n,a) - c\sqrt{\frac{log(count(n))}{count(n.a)}}$
  if $a \in A_{act}$ **then**
    $s' \sim a(s)$, $o \leftarrow null$
  else
    $s' \leftarrow s$, $o \leftarrow obs(a, s)$
  $Simulate(s', n.a.o, depth + 1)$
  $count(n.a) \leftarrow count(n.a) + 1$, $count(n.a.o) \leftarrow count(n.a.o) + 1$
  $V(n.a) \leftarrow \frac{\sum count(n.a.o) \cdot V(n.a.o)}{count(n.a)}$
  $V(n) \leftarrow \min_a V(n.a)$
Rollout(s, B)
  $depth \leftarrow 0$
  while $s \notin G \wedge depth < MaxRolloutDepth$ **do**
    $a \leftarrow \pi_{rollout,B}(s)$
    $s \sim a(s)$
    if a *is a sensing action* **then**
      Remove from B states that do not agree with s on the observation
    $depth \leftarrow depth + 1$
  return $depth$

Algorithm 2: Single State h_{add}

1 $\pi_{h_{add}}(s)$
2 $\quad fact_0 \leftarrow$ all facts in s
3 $\quad i \leftarrow 1$
4 $\quad$ **repeat**
5 $\quad\quad action_i \leftarrow \{a \in A_{act} : fact_{i-1} \models pre(a), a \notin \bigcup_{j=1..i-1} action_j\}$
6 $\quad\quad fact_i \leftarrow fact_{i-1} \cup \{f \in eff(a) : a \in action_i\}$
7 $\quad\quad i \leftarrow i+1$
8 $\quad$ **until** $fact_{i-1} = fact_{i-2}$;
9 $\quad$ **return** $\sum_{f \in G} i : f \in fact_i, f \notin fact_{i-1}$

Algorithm 3: Belief Space h_{add}

1 $\pi_{h_{add}}(s, B)$
2 $\quad \forall s' \in B, fact_0^{s'} \leftarrow$ all facts in s'
3 $\quad B_0 \leftarrow B$
4 $\quad i \leftarrow 1$
5 $\quad$ **repeat**
6 $\quad\quad B_i \leftarrow B_{i-1}$
7 $\quad\quad action_i \leftarrow \{a \in A_{act} : \forall s' \in B_i, fact_{i-1}^{s'} \models pre(a), a \notin \bigcup_{j=1..i-1} action_j\}$
8 $\quad\quad$ **for** $a \in A_{sense}$ **do**
9 $\quad\quad\quad$ **if** $\forall s' \in B_i, fact_{i-1}^{s'} \models pre(a)$ **then**
10 $\quad\quad\quad\quad F \leftarrow$ the values of $obs(a)$ in $fact_i^s$
11 $\quad\quad\quad\quad$ **for** $s' \in B_i, s' \neq s$ **do**
12 $\quad\quad\quad\quad\quad F' \leftarrow$ the values of $obs(a)$ in $fact_i^{s'}$
13 $\quad\quad\quad\quad\quad$ **if** $F' \neq F$ **then**
14 $\quad\quad\quad\quad\quad\quad B_i \leftarrow B_i \setminus \{s'\}$
15 $\quad\quad \forall s' \in B_i, fact_i^{s'} \leftarrow fact_{i-1}^{s'} \cup \{f \in eff(a) : a \in action_i\}$
16 $\quad\quad i \leftarrow i+1$
17 $\quad$ **until** $B_{i-1} = B_{i-2} \wedge \forall s' \in B_{i-1} : fact_{i-1}^{s'} = fact_{i-2}^{s'}$;
18 $\quad$ **return** $\sum_{f \in G} i : f \in fact_i^s, f \notin fact_{i-1}^s$

5.1 Delete Relaxation Heuristics

Delete relaxation heuristics are built upon the notion that if actions have only positive effects, then the number of actions that can be executed before the state becomes fixed is finite, and in many cases, small. Also, as actions cannot destroy the precondition of other actions, one can execute actions in parallel. Algorithm 2 portrays a delete relaxation heuristic.

Delete relaxation heuristics create a layered graph, interleaving action and fact layers. The first layer, which is a fact layer, contains all the facts that hold in the state for which the heuristic is computed (line 2). The second layer, which is an action layer, contains all the actions whose preconditions hold given the facts in the first layer (line 5). The next layer, which is again a fact layer, contains all the positive effects of the actions in the previous layer, as well as all facts from the previous layer (line 6), and so forth. We stop developing the graph once no new facts can be obtained (line 8).

After the graph is created, one can compute a number of heuristic estimates. The h_{max} returns the depth of the first fact layer that satisfies G. The h_{add} heuristic sums the fact depth of all goal facts (line 9) [2]. The h_{ff} heuristic computes a plan in the relaxed space by tracing back actions that achieved the goal predicates [9].

In this paper we experimented using the h_{add} heuristic.

5.2 Heuristics in Belief Space

A major disadvantage of the above heuristics is that they focus on a single state. When the agent is aware of the true state of the system, observations have no value. Hence, the above heuristics, as well as any heuristic that is based on a single state, do not provide an estimate for the value of information, which is a key advantage of POMDPs. We hence suggest now a heuristic that is computed over a set B of possible states (Algorithm 3).

We compute again the delete relaxation graph, with a few modifications. We compute for each state in B a separate fact layer. An action can be applied only if its preconditions are satisfied in the fact layers of all agents (line 7). This is equivalent to the requirement in contingent planning where an action is applicable only if it is applicable in all states in the current belief, where B is served as an approximation of the true belief state.

Second, our method leverages the deterministic observations, that allow us to filter out states that are inconsistent with the received observation (lines 8-14). When a sensing action can be applied, all states that do not agree with the value of s on the observation are discarded from B (lines 10-14). That is, we remove the fact layers corresponding to these states, and no longer consider them when computing which actions can be applied.

We stop when both no states were discarded, and no new facts were obtained (line 17). This process must take into account sensing actions to remove states that are incompatible with s, which would allow, at the next iteration, that action preconditions would be satisfied for less states, and hence additional actions can be executed.

6 Empirical Evaluation

We conduct an empirical study to evaluate our methods. Our methods are implemented in C#.

6.1 Benchmark Domains

We extended the following contingent planning benchmarks to stochastic settings:

Doors: In the door domain the agent must move in a grid to reach a target position. Odd levels in the grid are all open, while in even levels there are doors, and only one door is open. The agent can sense whether a door is open when it is at adjacent cells. The agent must identify the open doors and get to the target position. In the stochastic version the agent can open a closed door with some probability of success. The agent can hence either search for the already open door, or attempt to open a closed door.

Blocks World: In the contingent blocks world problem, the agent does not know the structure of the initial block configuration, but it can sense whether one block is on top of another one, and whether a block is clear. In the stochastic version moving a block from one block to another has a 0.3 probability of success, while moving blocks to and from the table succeeds deterministicly. Hence, it is often preferable to use the table as an intermediate position.

Unix: In this domain the agent must search for a file in a file system, and copy it to a destination folder. In the stochastic version there is a non-uniform distribution over the possible locations of the file.

	Sim.	Avg cost			Avg step time (secs)			Success		
Domain		Rnd	$\pi_{h_{add}}(s)$	$\pi_{h_{add}}(s,B)$	Rnd	$\pi_{h_{add}}(s)$	$\pi_{h_{add}}(s,B)$	Rnd	$\pi_{h_{add}}(s)$	$\pi_{h_{add}}(s,B)$
doors 5	1500	17.3	**13.7**	14.4	6.552	0.403	1.86	100%	100%	100%
blocks 4	1500	4.9	4.45	4.8	0.33	0.13	0.37	100%	100%	100%
localize 3	1500	14.75	**10.35**	**10.95**	1.325	0.752	1.16	80%	**100%**	**100%**
MedPks 10	1500	**6.3**	7.25	7.45	4.029	4.594	3.388	100%	100%	100%
Unix 1	1500	7.35	**5.65**	6.4	2.551	0.274	1.322	100%	100%	100%
Wumpus 5	1500	39.47	33.352	**24.33**	10.829	3.491	4.909	85%	85%	**90%**
Wumpus 5	500	56.117	33.722	**22.05**	2.442	0.823	1.12	85%	90%	**100%**

Table 1: Comparing rollout heuristics on various domains over 20 runs on each problem. Each number following the domain name is a specific instance of the domain, for example Wumpus5 means the grid is 5x5.

MedPks: The agent here needs to identify which illness a patient has and treat it. To do so, the agent tests for each illness independently, until the proper illness is found. The stochastic version here has non-uniform distribution over the possible illnesses as well.

Localize: In this domain the agent must reach a goal position in a grid. The agent does not know where it initially is, and can only sense adjacent walls. In the stochastic version there several places in the grid where the agent may slip and stay in place. This makes the localization in the grid more difficult.

Wumpus: In this challenging problem the agent must reach a target position in a grid infested by monsters called Wumpuses. Cells may be unsafe to travel as they may contain either a Wumpus or a pit. Wumpuses emit a stench, and pits emit a breeze, both of which can be sensed in adjacent cells. The agent must sense in multiple positions to identify the safe cells. The stochastic version here also has non-uniform distribution over the safe cells.

6.2 Results

For each domain above we run 20 online episodes, and compute the success rate, the average run time for computing the next action, and the average cost to the goal. We did not use a timeout, but runs longer than 100 steps were considered to be stuck in a loop, and terminated.

Table 1 presents the experiments results over the benchmarks, comparing the random (uniform) rollout policy (denoted Rnd), the h_{add} heuristic using a single state ($\pi_{h_{add}}(s)$), and the h_{add} heuristic over multiple states ($\pi_{h_{add}}(s,B)$).

We begin by looking at the quality of the policy — the average cost to the goal. As can be seen, the random rollout policy is best only in the MedPks domain, and close to best in unix. This is not too surprising, because in these two domains the best strategy is very simple, and random strategies easily stumble upon the goal. In blocks all methods achieved similar performance, because the optimal strategy is very short, and rollouts are less important. This domain has many possible actions, and hence a huge branching factor, making it difficult to scale up using POMCP.

On doors and localize, which require lengthier trajectories to reach the goal, but do not need long information-gathering efforts, the single state h_{add} strategy operates very well. However, on Wumpus, where long sequences of actions are needed for information gathering, the multiple-state heuristic works best. We expected the results to be that way, although we expected more significant difference between the "smart" heuristics and the random rollout.

With respect to the required time to run the simulations for a single decision, the results are mixed. While obviously the random strategy requires no time to compute the next action during a rollout, it often results in lengthy rollouts, which reduce this effect. The single state heuristic is almost always faster than the multi-state heuristic, but not by much.

7 Conclusion

In this paper we suggested to model goal POMDPs as stochastic contingent planning models, which allows us to use domain independent heuristics developed in the automated planning community to estimate the utility of belief states. We implemented our domain independent heuristics into the rollout mechanism of POMCP — a well known online POMDP planner that constructs a search tree to evaluate which action to take next. We provide an empirical evaluation showing how heuristics provide much leverage, especially in complex domains that require a long planning horizon, compared to the standard uniform rollout policy that is often used in POMCP.

For future research we intend to integrate our methods into other solvers, such as RTDP-BEL, or into point-based planners as a method to gather good belief points. We can also experiment with additional heuristics, other than the h_{add} heuristic used in this paper.

References

[1] Alexandre Albore, Héctor Palacios & Hector Geffner (2009): *A Translation-Based Approach to Contingent Planning*. In: *IJCAI*, 9, pp. 1623–1628, doi:10.5555/1661445.1661706. Available at `https://dl.acm.org/doi/10.5555/1661445.1661706`.

[2] Blai Bonet & Héctor Geffner (2001): *Planning as heuristic search*. *Artificial Intelligence* 129(1-2), pp. 5–33, doi:10.1016/S0004-3702(01)00108-4.

[3] Blai Bonet & Hector Geffner (2009): *Solving POMDPs: RTDP-Bel vs. Point-based Algorithms*. In Craig Boutilier, editor: *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pp. 1641–1646, doi:10.5555/1661445.1661709. Available at `https://dl.acm.org/doi/10.5555/1661445.1661709`.

[4] Blai Bonet & Hector Geffner (2011): *Planning under Partial Observability by Classical Replanning: Theory and Experiments*. In: *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pp. 1936–1941, doi:10.5591/978-1-57735-516-8/IJCAI11-324.

[5] Ronen I Brafman & Guy Shani (2016): *Online belief tracking using regression for contingent planning*. *Artificial Intelligence* 241, pp. 131–152, doi:10.1016/j.artint.2016.08.005.

[6] Hector Geffner & Blai Bonet (2022): *A concise introduction to models and methods for automated planning*. Springer Nature.

[7] Malte Helmert & Carmel Domshlak (2009): *Landmarks, Critical Paths and Abstractions: What's the Difference Anyway?* In Lubos Brim, Stefan Edelkamp, Eric A. Hansen & Peter Sanders, editors: *Graph Search Engineering, 29.11. - 04.12.2009, Dagstuhl Seminar Proceedings* 09491, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, Germany, doi:10.1609/icaps.v19i1.13370. Available at `http://drops.dagstuhl.de/opus/volltexte/2010/2432/`.

[8] Malte Helmert, Patrik Haslum, Jörg Hoffmann & Raz Nissim (2014): *Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces*. *Journal of the ACM (JACM)* 61(3), pp. 1–63, doi:10.1145/2559951.

[9] J. Hoffmann & B. Nebel (2001): *The FF Planning System: Fast Plan Generation Through Heuristic Search*. *JAIR* 14, pp. 253–302, doi:10.1613/jair.855.

[10] Jörg Hoffmann & Ronen I. Brafman (2005): *Contingent Planning via Heuristic Forward Search witn Implicit Belief States*. In Susanne Biundo, Karen L. Myers & Kanna Rajan, editors: *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS 2005), June 5-10 2005, Monterey, California, USA*, AAAI, pp. 71–80, doi:10.5555/3037062.3037072. Available at `http://www.aaai.org/Library/ICAPS/2005/icaps05-008.php`.

[11] Marcus Hörger, Hanna Kurniawati & Alberto Elfes (2019): *Multilevel Monte-Carlo for Solving POMDPs Online*. In Tamim Asfour, Eiichi Yoshida, Jaeheung Park, Henrik Christensen & Oussama Khatib, editors: *Robotics Research - The 19th International Symposium ISRR 2019, Hanoi, Vietnam, October 6-10, 2019, Springer Proceedings in Advanced Robotics* 20, Springer, pp. 174–190, doi:10.1007/978-3-030-95459-8_11.

[12] Ronald A Howard (1966): *Information value theory*. *IEEE Transactions on systems science and cybernetics* 2(1), pp. 22–26, doi:10.1109/TSSC.1966.300074.

[13] Thomas Keller & Patrick Eyerich (2012): *PROST: Probabilistic Planning Based on UCT*. In Lee McCluskey, Brian Charles Williams, José Reinaldo Silva & Blai Bonet, editors: *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil, June 25-19, 2012*, AAAI, doi:10.1609/icaps.v22i1.13518. Available at `http://www.aaai.org/ocs/index.php/ICAPS/ICAPS12/paper/view/4715`.

[14] Thomas Keller & Malte Helmert (2013): *Trial-based heuristic tree search for finite horizon MDPs*. In: *Proceedings of the International Conference on Automated Planning and Scheduling*, 23, pp. 135–143, doi:10.1609/icaps.v23i1.13557. Available at `http://www.aaai.org/ocs/index.php/ICAPS/ICAPS13/paper/view/6026`.

[15] Sung-Kyun Kim, Oren Salzman & Maxim Likhachev (2019): *POMHDP: Search-based belief space planning using multiple heuristics*. In: *Proceedings of the International Conference on Automated Planning and Scheduling*, 29, pp. 734–744, doi:10.1609/icaps.v29i1.3542. Available at `https://ojs.aaai.org/index.php/ICAPS/article/view/3542`.

[16] Hanna Kurniawati (2022): *Partially observable markov decision processes and robotics*. *Annual Review of Control, Robotics, and Autonomous Systems* 5, pp. 253–277, doi:10.1146/annurev-control-042920-092451.

[17] Hanna Kurniawati & Vinay Yadav (2016): *An online POMDP solver for uncertainty planning in dynamic environment*. In: *Robotics Research: The 16th International Symposium ISRR*, Springer, pp. 611–629, doi:10.1007/978-3-319-28872-7_35.

[18] Yuanfu Luo, Haoyu Bai, David Hsu & Wee Sun Lee (2019): *Importance sampling for online planning under uncertainty*. *The International Journal of Robotics Research* 38(2-3), pp. 162–181, doi:10.1177/0278364918780322.

[19] Silvia Richter, Malte Helmert & Matthias Westphal (2008): *Landmarks Revisited.* In: *AAAI*, 8, pp. 975–982, doi:10.2307/j.ctt1zxsjcs. Available at `http://www.aaai.org/Library/AAAI/2008/aaai08-155.php`.

[20] Jussi Rintanen (2008): *Regression for classical and nondeterministic planning*. In: *ECAI 2008*, IOS Press, pp. 568–572, doi:10.3233/978-1-58603-891-5-568.

[21] Stéphane Ross, Joelle Pineau, Sébastien Paquet & Brahim Chaib-Draa (2008): *Online planning algorithms for POMDPs*. *Journal of Artificial Intelligence Research* 32, pp. 663–704, doi:10.1613/jair.2567.

[22] Juan Carlos Saborío & Joachim Hertzberg (2019): *Planning Under Uncertainty Through Goal-Driven Action Selection*. In: *Agents and Artificial Intelligence: 10th International Conference, ICAART 2018, Funchal, Madeira, Portugal, January 16–18, 2018, Revised Selected Papers 10*, Springer, pp. 182–201, doi:10.1007/978-3-030-05453-3_9.

[23] Juan Carlos Saborío & Joachim Hertzberg (2020): *Efficient planning under uncertainty with incremental refinement*. In: *Uncertainty in Artificial Intelligence*, PMLR, pp. 303–312, doi:10.1109/TSMC.1987.4309045.

[24] Guy Shani & Ronen I Brafman (2011): *Replanning in domains with partial information and sensing actions*. In: *IJCAI*, 2011, Citeseer, pp. 2021–2026, doi:10.5591/978-1-57735-516-8/IJCAI11-337.

[25] Guy Shani, Joelle Pineau & Robert Kaplow (2013): *A survey of point-based POMDP solvers*. Autonomous Agents and Multi-Agent Systems 27, pp. 1–51, doi:10.1007/s10458-012-9200-2.

[26] William Shen, Felipe Trevizan, Sam Toyer, Sylvie Thiébaux & Lexing Xie (2019): *Guiding search with generalized policies for probabilistic planning*. In: Proceedings of the International Symposium on Combinatorial Search, 10, pp. 97–105, doi:10.1609/socs.v10i1.18507.

[27] David Silver & Joel Veness (2010): *Monte-Carlo planning in large POMDPs*. Advances in neural information processing systems 23, doi:10.5555/2997046.2997137.

[28] Richard D Smallwood & Edward J Sondik (1973): *The optimal control of partially observable Markov processes over a finite horizon*. Operations research 21(5), pp. 1071–1088, doi:10.1287/opre.21.5.1071.

[29] Adhiraj Somani, Nan Ye, David Hsu & Wee Sun Lee (2013): *DESPOT: Online POMDP planning with regularization*. Advances in neural information processing systems 26, doi:10.1613/jair.5328.

[30] Zachary Sunberg & Mykel Kochenderfer (2018): *Online algorithms for POMDPs with continuous state, action, and observation spaces*. In: Proceedings of the International Conference on Automated Planning and Scheduling, 28, pp. 259–263, doi:10.48550/arXiv.1709.06196.

[31] Yuchen Xiao, Sammie Katt, Andreas ten Pas, Shengjian Chen & Christopher Amato (2019): *Online planning for target object search in clutter under partial observability*. In: 2019 International Conference on Robotics and Automation (ICRA), IEEE, pp. 8241–8247, doi:10.1109/ICRA.2019.8793494.

[32] Nan Ye, Adhiraj Somani, David Hsu & Wee Sun Lee (2017): *DESPOT: Online POMDP planning with regularization*. Journal of Artificial Intelligence Research 58, pp. 231–266, doi:10.1613/jair.5328.

Evaluating Heuristic Search Algorithms in Pathfinding: A Comprehensive Study on Performance Metrics and Domain Parameters*

Aya Kherrour Marco Robol Marco Roveri Paolo Giorgini
Department of Information Engineering and Computer Science
University of Trento – Trento, Italy
name.surname@unitn.it

The paper presents a comprehensive performance evaluation of some heuristic search algorithms in the context of autonomous systems and robotics. The objective of the study is to evaluate and compare the performance of different search algorithms in different problem settings on the pathfinding domain. Experiments give us insight into the behavior of the evaluated heuristic search algorithms, over the variation of different parameters: domain size, obstacle density, and distance between the start and the goal states. Results are then used to design a selection algorithm that, on the basis of problem characteristics, suggests the best search algorithm to use.

1 Introduction

Autonomous agents and robotics have increasingly been used in various domains, such as, industrial applications [16, 18, 8], surveillance [17], and exploration [4]. These systems are designed to autonomously make decisions and execute actions based on their dynamic and unpredictable environments. Under such conditions, systems are required to be as reactive as possible to changes in the environment. Therefore, ensuring good performance is a significant challenge. In the case of planning, selecting the most effective search algorithm becomes crucial to enhance the overall performance of the system. Real-time heuristic search (RTS) (e.g.,[14, 9]) is a state-of-the-art approach for planning while executing, that helps minimize agent reaction time. These algorithms enable agents to make decisions by interleaving planning with execution while considering the evolving environment, which is an essential property in applications such as robotics, and video game agents. Despite the numerous methods proposed in this field [14, 13, 12, 3, 2], a comprehensive understanding of these algorithms remains elusive. Existing studies [1, 13, 10] have primarily focused on testing the performance of the algorithm based on a single parameter (e.g., look-ahead, sensor range). However, the influence of the problem domain characteristics on algorithm performance remains an understudied aspect.

Our objective is to investigate the characteristics of the problem that may impact algorithm performances. To achieve this, we first review existing state of art of search algorithms and their applications. Then we design our experiments to evaluate some of the algorithm performances, also defining relevant metrics to use in the evaluation. Later on, experiment results are analyzed to provide a comprehensive understanding of how problem characteristics influence the performance of the different search algorithms. Finally, from the insight gained from our study, we introduce a selection algorithm that helps us to select the appropriate search algorithm.

*M. Robol and M. Roveri are partially supported by the project MUR PRIN 2020 - RIPER - Resilient AI-Based Self-Programming and Strategic Reasoning - CUP E63C22000400001. M. Roveri and P. Giorgini are partially supported by the PNRR project FAIR - Future AI Research (PE00000013), under the NRRP MUR program funded by NextGenerationEU.

A. Ferrando, R.C. Cardoso (Eds.): Third Workshop on
Agents and Robots for reliable Engineered Autonomy 2023 (AREA'23)
EPTCS 391, 2023, pp. 102–112, doi:10.4204/EPTCS.391.12

This paper is structured as follows. We begin by reviewing state-of-the-art search algorithms and previous performance evaluation studies. Next, we describe the approach we adopted to evaluate the search algorithms that are subject to our study, define the problem domain and its characteristics, and the performance metrics. In section four, we present our experimental results. Section five introduces our proposed selection algorithm with an execution example. Finally, we derive our conclusion, the limitations of our study, and possible directions for future work.

2 State of the art and related work

Path planning is a relevant problem in autonomous agents, such as, self-driving cars, robots, unmanned aerial vehicles (UAVs), and unmanned ground vehicles (UGVs), in which the host agent deliberates its path by moving from one position to another while avoiding obstacles and respecting some constraints [19]. One of the path planning approaches that have been proposed to control the movement of these agent-based systems is search-based algorithms, with Dijkstra and its extension A* [5, 21], being the most popular and effective ones. Besides these, other search algorithms have been proposed in the literature, specifically to reduce reaction time, broadly classified as real-time or incremental search algorithms.

Real-time search algorithms must find a solution in a limited time, while incremental search instead uses the previously obtained searches to speed up the search. Amongst the first class, Real-time A* (RTA*) and Learning real-time A* (LRTA*) [14] were some of the first algorithms to apply real-time heuristic search in path planning problems for moving agents. Both algorithms use heuristics to guide the search toward the goal, with RTA* storing the second-best f-values of the previous state as the best alternative to choose when backtracking from the current state. However, this may mislead the agent. Thereby, LRTA* overcomes this by storing the first best value rather than the second and learning from comparing the heuristic values of the adjacent states, thus preventing the algorithm from misleading the agent. Another optimized version of LRTA* is real-time adaptive A* (RTAA*) [12]. It first determines its local search space and then speeds up the search by updating the heuristics values of states. It was developed for stationary target search problems and it follows trajectories of smaller costs. Anytime repairing A* (ARA*) [15] is a variation of A* that has been designed to find suboptimal solutions fast and then improve them over time, which makes it a good algorithm for problems where finding an optimal solution is not mandatory or too expensive [15]. However, finding suboptimal solutions does not make it find the optimal solution [15].

Moving to incremental search algorithms, D* (Dynamic A*) is an incremental search algorithm used in real-time planning in robotics [20]. It is designed to react quickly to changes in the environment, by updating the nodes affected in the search tree rather than recomputing a new plan from scratch. D* has a main drawback that it requires a lot of memory to perform the search. LPA* (Lifelong Planning A*), an incremental version of A*. This algorithm is used in path planning or robot navigation in unknown terrain. It behaves just like A* in the first run, and then for the rest of subsequent searches it reuses the previous search thus reducing the number of examined nodes, which makes it fast. LPA* differs from D* in its search direction where it finds a path from the initial state to the initial goal state, therefore it does not fit in applications where the starting point may change over time. Another variant of D*, D* Lite, was developed based on LPA*, and is used for goal-directed path planning in unknown environments using the same idea as D*, however, it is a simple version of D* and produces effective results as the one delivered by D* as proven in [11]. While these heuristic search algorithms are numerous and diverse, it is important to emphasize that there are even more algorithms in the literature with new ones being developed, or existing ones being improved such as [2, 13, 7]. This variety of algorithms in the field emphasizes the complexity of the problem. Consequently, this variety of proposed algorithms presents a challenge; the wide number of these algorithms results in disparate performances in tasks such as path

planning, making it difficult to select the optimal algorithms for a given task.

Given the multitude of algorithms, studies have been done to compare the performance of RTS algorithms in path planning tasks such as [1] where authors have compared the performances of path planning algorithms using agents equipped with a sensor in both stationary and moving target settings to select the most appropriate algorithm. The obtained results were compared using various sensor ranges and angles, and the authors concluded that sensor range is an important parameter in selecting the algorithm, unlike the sensor angle. Therefore the most appropriate algorithm can be selected based on the sensor range and its priorities. However, the authors did not consider varying the environmental characteristics, which may influence the performance of the algorithms.

Another study has been presented in [13], in which the authors also compared real-time and incremental heuristic search algorithms using an autonomous agent in a navigation task to know which class of heuristic search approach is better to be used depending on how informed the h-values are, and the search objective such as minimizing the sum of the search and action execution.

The comparative analysis done in [1] considers only a static environment, neglecting the possible variations in the algorithm performance under different environmental characteristics. This limitation hampers us from creating a clear understanding of different search algorithms that could perform under different environmental characteristics, precisely in the context of path planning. Similarly, the study conducted in [13] restricts its investigation to only two algorithms each one from a different class i.e., real-time and incremental heuristic search. Even though it provides an insightful comparison, the study does not fully capture the breadth of available algorithms in these classes.

Moreover, the study aimed to provide a recommendation on when to use each of the algorithm classes, but also does not investigate algorithm performances in diverse environmental characteristics.

Consequently, considering the challenge posed by the presence of a wide range of search algorithms and the lack of a comprehensive comparative analysis under different environment settings, we propose to evaluate and analyze the performance of some search algorithms under different environment characteristics in the context of path planning, But first, we define the characteristics of the environment used, and through our experiments, we aim to provide a comprehensive evaluation of these algorithms followed by proposing our selection algorithm.

3 Experimental environment and performance metrices

Heuristic search algorithms play an important role in fields such as robotic pathfinding, as they determine the optimal path given a starting position and a goal position. Grid-based environments are commonly used for representing real-world environment scenarios, where these algorithms can be implemented, such as in autonomous navigation and robotics [6]. In this study, we comprehensively analyze well-known heuristic search algorithms, namely D*, D* Lite, LPA*, LRTA*, RTAA*, and ARA* in different grid environments. We use the Euclidean distance heuristic to guide the search of the algorithms and assess the impact of a few grid characteristics, such as the obstacle density and the grid size, on the performance of the algorithms.

In our study, we use grid-based environments due to their simplicity, and control ease, in addition to being commonly used in path-planning tasks in the research community. The grids are composed of white cells, representing traversable states, whereas the black ones represent non-traversable obstacles. The agent in our simulation can move in eight directions, with a cost equal to 1 for horizontal and vertical moves, and $\sqrt{2}$ for diagonal movements. We used two types of grid environments: randomly generated grid environments and personalized grid environments.

Randomly generated grid-based environments: The grids are characterized by three parameters: grid size (NxN), obstacle density, and the distance between the start and the goal states. To investigate the impact of each parameter on the performance of the search algorithms, we varied each parameter independently while keeping the other parameters constant. The variations included varying the grid size, varying start to goal distance, and varying the obstacle densities. For each grid in a variation, we generated ten random instances of the same grid parameters (e.g., a grid with 0.25 of obstacle density, size 300x300, and 140 of start to goal distance, have ten instances, which were generated randomly).

Personalised grid environment: Designed for simulating more specific scenarios. These grids have fixed size (71x31 units) and a fixed position for both the start and goal states, and they were divided into two parts based on their obstacle configuration:

- **Horizontal wall configuration**: For these experiments, we add horizontal walls of half grid width every 10 units of grid length. We added the walls in two orientations: once from left to right, and once from right to left in the newly generated grid.
- **Horizontal wall length configuration**: Here, we add all possible walls that can be placed within the grid length, and each time we generate a new grid we increase all wall lengths by 2 units.

We adopted these two distinct environments to provide a thorough analysis, seeking to reveal nuanced insights into the performance of the search algorithm in the presence of two different hindering scenarios. In the first one, obstacles are scattered randomly within the grid, whereas in the second one, the wall-like structures, appear as a mass of connected obstacles.

To evaluate the performance of the different search algorithms used in the experiments, we have selected the following metrics:

- **Path cost:** The metric measures the path length or the number of executed actions from the start to the goal state. It indicates the quality of the solution.
- **Memory consumption:** It measures the required amount of memory for the algorithm to find a solution. It is relevant to check the scalability of the algorithm, and it is measured in (KB).
- **Solving Time:** Represent the total time an algorithm takes to find a solution in (ms), measured in milliseconds (ms).

We carried out our experiments on 3.30GHz 27 Intel i9 cores, equipped with 250Gb of RAM and running Ubuntu Linux 22.04. Using the following settings: All algorithms were using the Euclidean distance as the heuristic function. For LRTA* and RTAA*, we set the number of expended nodes to 250 for each iteration. The ARA* algorithm was run with a weight of 2.5 for the heuristic. We ran each algorithm 100 times on each grid to account for randomness and to ensure the reliability of the results.

4 Experimental results

Grid Size Variation: In the results obtained by varying the grid size (see Figure 1, 3, 2), ARA* displays relatively stable solving time. However, it has some fluctuation in its standard deviation, with a slight increase as the grid size increases. Its maximum value of (76ms) was obtained at size 200. At grid size 50, RTAA* was the fastest algorithm, recording a solving time of (21ms). Subsequently, its solving time and allocated memory increased notably with the grid size. which indicates that the grid size affects the performance of RTAA* as it increases; and the algorithm is forced to do more searches. LRTA* has the highest solving time across all sizes compared to ARA*, RTAA*, D* Lite, LPA*, and even D* at size 50. Its performance greatly varies as indicated by its high standard deviations. Moreover, LRTA* did not show a clear trend in increasing time relative to grid size, indicating that it may be unpredictable

and inefficient for this task. Both D* Lite and LPA* exhibited relatively stable performances. D* Lite has a lower solution time across all grid sizes compared to LPA*. However, both algorithm's standard deviations show some degree of fluctuation. D*, on the other hand, showcased an extreme increase in solution time when transitioning from size 50 to 100, revealing its challenges as the grid size increases.

Regarding path cost (see Figure 3), while LPA*, D*, and ARA* lines overlap, suggesting similar path costs across all grid sizes, D* Lite consistently generates the shortest paths across all grid sizes, with a relatively small standard deviation. In terms of memory allocation (see Figure 2), LPA* and D* Lite were consistent across all grid sizes, requiring the least memory among all algorithms. In contrast, RTAA*, LRTA*, and D* demanded more memory as the grid size increased.

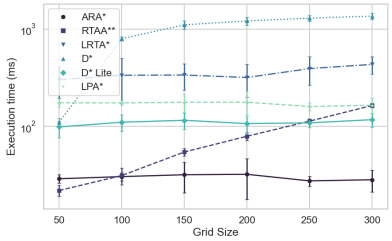


Figure 1: solving time vs. Grid Size

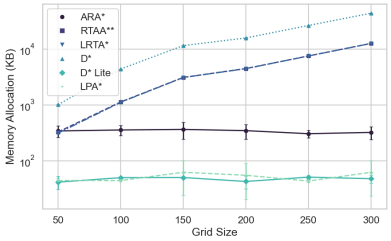


Figure 2: Memory Allocation vs. Grid Size

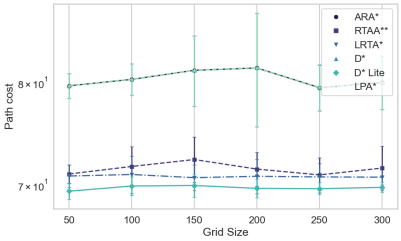


Figure 3: Path Cost vs. Grid Size

Start to goal distance variation: The obtained results from running the algorithms on grids with varying start-to-goal distances (see figures 4, 5, 6) revealed that both the mean of the path cost and the mean of solving time for all algorithms escalate as the distance increases (see Figure 4, 6). This outcome was expected, since longer distances naturally demand more computational efforts and more nodes to expand.

ARA* seems to have a strong correlation with the distance between the start and goal; its solving time is drastically influenced by this factor. Precisely, ARA* remains the fastest algorithm for distances smaller than approximately 140. Beyond this threshold, however, RTAA* takes the lead in terms of solving time.

Observing the path cost (see Figure 6), the lines for D* Lite, LRTA* RTAA* overlap, indicating similar performances. Correspondingly, LPA*, ARA*, and D* also exhibit overlapping lines, where D* Lite being the algorithm with the lowest path cost. In the context of allocated memory (see Figure 5), D* lite allocates the least memory for all distances followed by LPA* and then ARA*. The rest of the algorithms allocate almost a similar amount of memory with D* being the worst.

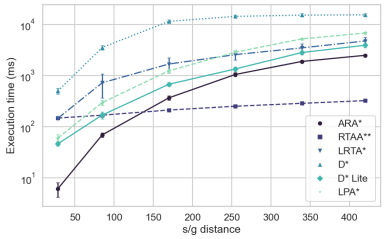


Figure 4: solving time vs. SG Distance

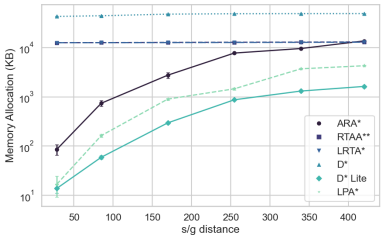


Figure 5: Memory Allocation vs. SG Distance

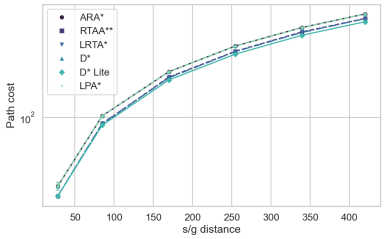


Figure 6: Path Cost vs. SG Distance

Obstacle density variation: RTAA* constantly excels by producing the shortest solving time among all algorithms and across all obstacle densities as depicted in Figure 7. This superior performance can be explained due to its rapid heuristic values update procedure within its local search space. The rest of the algorithm's solving time increases as the obstacle density rises, with D* being the one with a higher solving time, except for LPA*, which starts to decrease slightly after a density of 0.20.

Figure 9 shows that the path cost of all algorithms tends to increase as the density increases, which can be a predictable outcome since denser environments pose more complex navigation challenges. Amongst all algorithms, D* Lite maintains the lowest path across all densities, marking its efficiency in complex environments. D* Lite's performance is followed by LRTA* for obstacle densities below 0.25, and RTAA* outperforms the rest for densities higher than 0.25. In addition to maintaining the lowest path cost, D* Lite allocates the least amount of memory at all densities, as depicted in Figure 8, followed by LPA*. In contrast, both RTAA* and LRTA* consume a lot of memory, but not as much as D*, which allocates even more.

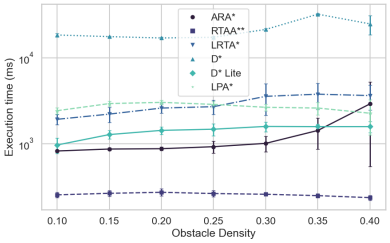


Figure 7: solving time vs. Obstacle Density

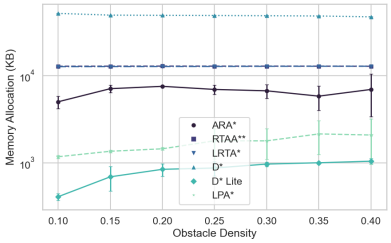


Figure 8: Memory Allocation vs. Obstacle Density

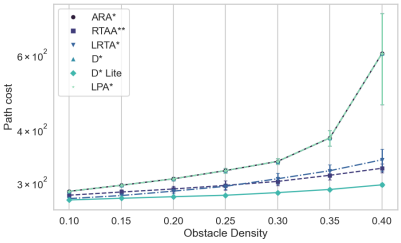


Figure 9: Path Cost vs. Obstacle Density

We hypothesize that changing other parameters that we have fixed while varying other ones could indeed provide us with further insight into how these parameters interact and affect the performance of the search algorithms i.e., using an obstacle density of 0.4 rather than 0.25 while changing the grid size. However, to maintain simplicity and manage the computational resources, we opted to keep a balanced grid size, obstacle density, and distance from the start to the goal that will help us fairly represent an environment for a pathfinding task. Also, the combination of varying one parameter independently from the other ones and then repeating the same experiment, in which we vary the fixed parameter using all other possible values would dramatically increase the number of possible experiments that must be done, in addition to the number of runs that they must be made for each grid, which will amplify the number of experiments that must be performed.

Horizontal wall configuration: In the horizontal wall configuration results, as depicted in Figures (10, 11, and 12). We observed that the path cost tends to increase for all algorithms as walls are added (see Figure 12), which is expected. LRTA* exhibited the highest solving time (see Figure 10), followed, in descending order by D* Lite, RTAA*, LPA*, ARA*, and LRTA*. Regarding the obtained results for the path cost, D* Lite produces the most optimal paths followed by ARA*, D*, and LPA*. In terms of memory allocation (see Figure 11), all algorithms tend to allocate the same amount of memory even when new walls are introduced, with only slight increases. Among all algorithms, D* allocates the highest amount of memory, while LPA* allocates the least memory, followed by D* Lite, LRTA*, RTAA*, and ARA*. However, it is worth noting that both ARA* and LPA* had a remarkable increase in memory consumption when adding the last wall.

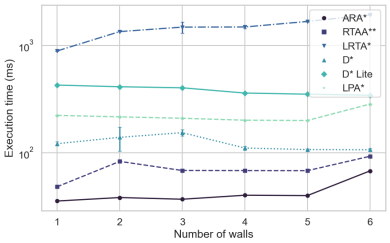


Figure 10: solving time vs. Number of walls

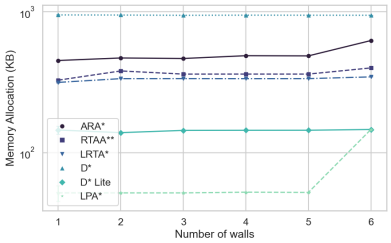


Figure 11: Memory Allocation vs. Number of walls

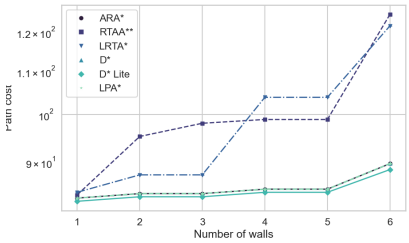


Figure 12: Path Cost vs. Number of walls

Horizontal wall length configuration: The results obtained from varying the wall sizes are depicted in the figures 13, 14, and 15. The numbers on the axis represent seven different lengths. The initial length corresponds to half the grid width, and with each subsequent step, it increases by two obstacles.

All algorithm's solving times varied across the different wall lengths showing a general trend of increasing as the wall lengths increase as depicted in Figure 13. LRTA* recorded the highest solving times, whereas ARA* recorded the lowest across all wall lengths. For the amount of memory used by the algorithms (see Figure 14), D* was again the one that consumed more memory. In contrast, both D* Lite and LPA* allocated almost similar and the latest amount of memory over all seven wall lengths. Turning to the path cost metrics (see Figure 15), all algorithm's path costs tend to increase as the wall lengths increase, with D* Lite generating the lowest path costs for all wall lengths.

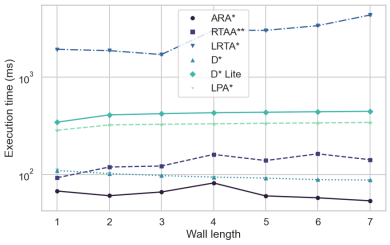


Figure 13: solving time vs. Walls length

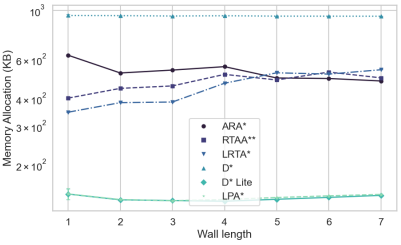


Figure 14: Memory Allocation vs. Walls length

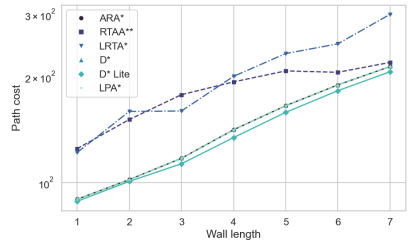


Figure 15: Path Cost vs. Walls length

Based on the results obtained in the grid size variation, the performance of most algorithms, particularly RTAA*, was affected by grid size. In the meanwhile, the high level of consistency of ARA* performance in terms of solving time regardless of the grid size indicates its suitability for various grid sizes higher than 100. D* proved to be the one generating the lowest path costs, and also allocating the least memory alongside LPA*. Obstacle density results showed that it is a factor that influences the performance of all algorithms. However, D* Lite kept generating the most efficient paths, which indicates its effectiveness in dense environments. Based on the results obtained by increasing the start-to-goal distance, ARA* appears to be the most affected one since its solving time continuously increases. In addition to allocating the least memory, D* Lite consistently exhibits the optimal paths, which indicates its utility where the shortest path is of importance. Adding horizontal walls each time in the same grid setting has increased the path length and solving time for all algorithms with D* being the worst. D* Lite keeps its optimal performance by generating the most optimal paths, which suggests its suitability in environments with many obstacles. However when increasing the length of the walls, ARA* displayed

Table 1: Selection algorithm evaluation results

Randomly generated NxN grid environments								
Algorithm	Number of walls	Wall length	Obstacle density	Grid size	s/g distance	Path cost	Memory Allocation (KB)	Solving time (ms)
RTAA*	-	-	0.35	100	43.174	81.012	1145.648	86.106
ARA*	-	-	0.35	100	43.174	86.041	437.012	63.376
D* Lite	-	-	0.35	100	43.174	73.698	109.248	252.110
RTAA*	-	-	0.35	250	56.080	78.769	7520.803	165.108
ARA*	-	-	0.35	250	56.080	83.798	173.603	33.001
D* Lite	-	-	0.35	250	56.080	73.698	94.861	196.605
RTAA*	-	-	0.25	100	38.118	77.012	1159.855	111.278
ARA*	-	-	0.25	100	38.118	75.455	382.359	55.283
D* Lite	-	-	0.25	100	38.118	72.870	111.344	285.450
RTAA*	-	-	0.35	100	36.069	70.497	1119.316	55.832
ARA*	-	-	0.35	100	36.069	97.426	825.953	136.500
D* Lite	-	-	0.35	100	36.069	69.083	91.369	218.514
Personalised grid environments								
Algorithm	Number of walls	Wall length	Obstacle density	Grid size	s/g distance	Path cost	Memory Allocation (KB)	Solving time (ms)
RTAA*	6	25	-	31x71	73.539	177.480	368.113	131.508
ARA*	6	25	-	31x71	73.539	117.195	531.484	88.499
D* Lite	6	25	-	31x71	73.539	113.095	243.723	708.3793
RTAA*	6	16	-	31x71	73.539	83.698	334.370	64.666
ARA*	6	16	-	31x71	73.539	83.113	449.302	46.540
D* Lite	6	16	-	31x71	73.539	82.527	244.100	686.459

the lowest solving times, suggesting its efficiency in environments with extensive barriers. Moreover, D* Lite keeps generating the lowest path costs.

In summary, while each algorithm has its strengths and weaknesses, D* Lite has continuously shown a good performance across most conditions, particularly in generating the optimal paths. Meanwhile, ARA* has proved its stability in producing the fastest paths regardless of the grid size. Moreover, RTAA* showed its ability to generate faster paths regardless of the obstacle density due to its faster procedure in updating the heuristic values.

5 Selection algorithm and example of execution

Based on the insights derived from our experimental results, we propose the selection algorithm represented in Algorithm 1. The algorithm is designed to select the appropriate search algorithm based on various priorities alongside the characterization of the environment used.

The selection algorithm emphasizes the desired priority first, which could manifest in various aspects of pathfinding tasks, including the path cost, memory usage, or the time taken to find a solution. The rational reason for emphasizing ”Priority” at the beginning of the selection algorithm is that, based on this user choice, different search algorithms may be considered to suit best the addressed problem. Thus, by tackling the ”Priority” upfront, The selected algorithm will eventually, cater to the main requirements of the task at hand.

The algorithm takes as input the Grid that is of size *NxN* with the start and goal positions, the distance threshold, and the priority criterion. If we aim to minimize memory usage, path cost, or both (Line 2), the algorithm suggests using D* Lite. However, If minimizing the solving time is our primary concern (line 4), we should first calculate the Euclidean distance between the start and the goal using the function in line 12. If the distance is higher or equal to the threshold (in our experiments it is equal to 140), the algorithms suggest using RTAA*. If the distance is less, then the selection algorithm suggests using ARA*.

Algorithm 1 selection algorithm

Input:
Grid: < array: $size[N][N]$, start($start_x, start_y$), goal($goal_x, goal_y$)>
Integer: D ▷ Distance threshold
String: P ▷ Priority_criterion
Output:
Selected algorithm based on specified criteria.

```
1: function SELECT_ALGORITHM( Grid<size, start, goal>, D, P)
2:     if P is 'Memory' or P is 'PathCost' then
3:         return DSTAR_LITE_ALGO
4:     else if P is 'SolvingTime' then
5:         if COMPUTEEUCLIDEANDISTANCE(start, goal) ≥ D then
6:             return RTAA_ALGO
7:         else
8:             return ARA_STAR
9:         end if
10:    end if
11: end function
12: function COMPUTEEUCLIDEANDISTANCE(start, goal)
13:    return √((start_x − goal_x)^2 + (start_y − goal_y)^2)
14: end function
```

5.1 Example of execution

To showcase the efficiency of our selection algorithm, we have designed an execution example that consists of generating a grid with random parameters i.e., choose a random grid size, obstacle density, and start to goal distance. After that, we generate an identical grid, however, we change only one parameter each time while keeping the other parameters as they were in the initial grid. We do the same for generating personalized grid environments, where we generate a grid with a random choice of wall number, and then we change the wall lengths.

We run RTAA*, ARA*, and D*Lite algorithms on the generated grids (all obtained results are shown in table 1) alongside our selection algorithm, so that we can compare the obtained results with what the selection algorithm is suggesting to use for the given grid.

It is essential to highlight that the randomly chosen values for the grid parameters are all within the range of the values used in the experiments, which ensures that the thresholds chosen are relevant. Also, the reason behind using only RTAA*, ARA*, and D*Lite in this execution example is because of their standout performance in our experiments.

For each grid, we run our selection algorithm each time with a different priority, and it suggests using an algorithm that will perform the best given the selected priority. The highlighted values in the table refer to the outcomes derived from the suggested search algorithm by our selection algorithm. These highlighted values are notably the best that we can obtain for each grid given a certain priority except for the red one; yellow represents the most efficient paths, green represents the best values that we can obtain if we want to reduce memory consumption, and the values highlighted in purple indicates the best solving time. However, our algorithm failed in selecting the right algorithm to use in the fourth grid when prioritizing the solving time, where the shortest solving time was found by RTAA*, instead our

algorithm suggested using ARA*.

6 Conclusion

This work aimed to evaluate the performance of several known heuristic search algorithms, such as D*, D* Lite, LPA*, LRTA*, RTAA*, and ARA* in terms of solving time, memory consumption, and path cost. The evaluation was made using different randomly generated grid environments with different characteristics alongside personalized grid environments with horizontal walls and different wall lengths as different hinders.

Our experimental evaluation revealed that all the algorithms exhibit different performances with strengths and weaknesses under different grid characterizations. D* Lite consistently generated the shortest paths even in obstacle-dense grids, indicating its efficiency in dense environments. ARA* consistently provides faster solutions as the grid size increases, particularly larger than 100, while RTTA* generates faster solutions in smaller grid sizes, with the advantage of not being affected by dense environments.

Our study provides valuable insights into selecting the appropriate heuristic search algorithm in the pathfinding domain. Using these insights, we propose a selection algorithm used to optimize the performance needed in a pathfinding domain, such as, solving time, path length, or memory consumption. However, our evaluation focused only on static environments, while dynamic environments may introduce additional challenges. Furthermore, we considered a limited set of experiments, not covering all possible combinations of the grid characteristics. This limitation means that the selection algorithm might not always recommend the most optimal solution, as seen in the example we introduced to evaluate our algorithm.

In future work, we aim to address these limitations as follows: Firstly, we plan to extend our evaluation of the search algorithms to include dynamic environments. Secondly, we intend to explore various combinations of both domain characterization and priorities. Furthermore, we want to include additional scenario configuration, extending the random obstacles and the walls scenarios. With such additional understanding, we aim to refine our selection algorithm to automatically take decisions among the best search algorithms, based on the type of obstacles in the local search space.

References

[1] Nafiz Arica, Aysegul Mut, Alper Yörükçü & Kadir Alpaslan Demir (2017): *An Empirical Comparison of Search Approaches for Moving Agents*. *Comput. Intell.* 33(3), pp. 368–400, doi:10.1111/coin.12092.

[2] Yngvi Björnsson, Vadim Bulitko & Nathan R. Sturtevant (2009): *TBA*: Time-Bounded A**. In Craig Boutilier, editor: *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*, pp. 431–436. Available at `http://ijcai.org/Proceedings/09/Papers/079.pdf`.

[3] David Bond, Niels A. Widger, Wheeler Ruml & Xiaoxun Sun (2010): *Real-Time Search in Dynamic Worlds*. In Ariel Felner & Nathan R. Sturtevant, editors: *Proceedings of the Third Annual Symposium on Combinatorial Search, SOCS 2010, Stone Mountain, Atlanta, Georgia, USA, July 8-10, 2010*, AAAI Press, doi:10.1609/socs.v1i1.18174. Available at `http://aaai.org/ocs/index.php/SOCS/SOCS10/paper/view/2103`.

[4] Guillaume Brat, Ewen Denney, Dimitra Giannakopoulou, Jeremy Frank & Ari Jónsson (2006): *Verification of autonomous systems for space applications*. In: *2006 IEEE Aerospace Conference*, IEEE, pp. 11–pp, doi:10.1109/AERO.2006.1656029.

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest & Clifford Stein (2009): *Introduction to Algorithms, 3rd Edition*. MIT Press. Available at `http://mitpress.mit.edu/books/introduction-algorithms`.

[6] A. Elfes (1989): *Using occupancy grids for mobile robot perception and navigation*. Computer 22(6), pp. 46–57, doi:10.1109/2.30720.

[7] Carlos Hernández & Jorge A. Baier (2011): *Real-Time Heuristic Search with Depression Avoidance*. In Toby Walsh, editor: *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, IJCAI/AAAI, pp. 578–583, doi:10.5591/978-1-57735-516-8/IJCAI11-104.

[8] Mohd Javaid, Abid Haleem, Ravi Pratap Singh & Rajiv Suman (2021): *Substantial capabilities of robotics in enhancing industry 4.0 implementation*. Cognitive Robotics 1, pp. 58–75, doi:10.1016/j.cogr.2021.06.001.

[9] Sven Koenig (2001): *Agent-Centered Search*. AI Mag. 22(4), pp. 109–132, doi:10.1609/aimag.v22i4.1596.

[10] Sven Koenig (2004): *A Comparison of Fast Search Methods for Real-Time Situated Agents*, pp. 864–871. doi:10.1109/AAMAS.2004.10122. Available at `https://doi.ieeecomputersociety.org/10.1109/AAMAS.2004.10122`.

[11] Sven Koenig & Maxim Likhachev (2002): *D*Lite*, pp. 476–483. Available at `http://www.aaai.org/Library/AAAI/2002/aaai02-072.php`.

[12] Sven Koenig & Maxim Likhachev (2006): *Real-time adaptive A**. In Hideyuki Nakashima, Michael P. Wellman, Gerhard Weiss & Peter Stone, editors: *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006), Hakodate, Japan, May 8-12, 2006*, ACM, pp. 281–288, doi:10.1145/1160633.1160682.

[13] Sven Koenig & Xiaoxun Sun (2009): *Comparing real-time and incremental heuristic search for real-time situated agents*. Auton. Agents Multi Agent Syst. 18(3), pp. 313–341, doi:10.1007/s10458-008-9061-x.

[14] Richard E Korf (1990): *Real-time heuristic search*. Artificial intelligence 42(2-3), pp. 189–211, doi:10.1016/0004-3702(90)90054-4.

[15] Maxim Likhachev, Geoffrey J. Gordon & Sebastian Thrun (2003): *ARA*: Anytime A* with Provable Bounds on Sub-Optimality*, pp. 767–774. Available at `https://proceedings.neurips.cc/paper/2003/hash/ee8fe9093fbbb687bef15a38facc44d2-Abstract.html`.

[16] Manuel Müller, Timo Müller, Behrang Ashtari Talkhestani, Philipp Marks, Nasser Jazdi & Michael Weyrich (2021): *Industrial autonomous systems: a survey on definitions, characteristics and abilities*. at-Automatisierungstechnik 69(1), pp. 3–13, doi:10.1515/auto-2020-0131.

[17] Michal Pechoucek, Simon G Thompson & Holger Voos (2008): *Defence Industry Applications of Autonomous Agents and Multi-Agent Systems*. Springer, doi:10.1007/978-3-7643-8571-2.

[18] Francisco J Perez-Grau, J Ramiro Martinez-de Dios, Julio L Paneque, J Joaquin Acevedo, Arturo Torres-González, Antidio Viguria, Juan R Astorga & Anibal Ollero (2021): *Introducing autonomous aerial robots in industrial manufacturing*. Journal of Manufacturing Systems 60, pp. 312–324, doi:10.1016/j.jmsy.2021.06.008.

[19] José Ricardo Sánchez-Ibáñez, Carlos Jesús Pérez-del-Pulgar & Alfonso García-Cerezo (2021): *Path Planning for Autonomous Mobile Robots: A Review*. Sensors 21(23), p. 7898, doi:10.3390/s21237898.

[20] Anthony Stentz (1995): *The Focussed D* Algorithm for Real-Time Replanning*. In: *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI 95, Montréal Québec, Canada, August 20-25 1995, 2 Volumes*, Morgan Kaufmann, pp. 1652–1659. Available at `http://ijcai.org/Proceedings/95-2/Papers/082.pdf`.

[21] Dmitry S. Yershov & Steven M. LaValle (2011): *Simplicial Dijkstra and A* algorithms for optimal feedback planning*. In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2011, San Francisco, CA, USA, September 25-30, 2011*, IEEE, pp. 3862–3867, doi:10.1109/IROS.2011.6095032.

Multi-Robot Task Planning to Secure Human Group Progress

Roland Godet[1,2] roland.godet@laas.fr
Charles Lesire[1] charles.lesire@onera.fr
Arthur Bit-Monnot[2] arthur.bit-monnot@laas.fr
[1] ONERA/DTIS, University of Toulouse, France
[2] LAAS-CNRS, University of Toulouse, INSA, Toulouse, France

Recent years have seen an increasing number of deployment of fleets of autonomous vehicles. As the problem scales up, in terms of autonomous vehicles number and complexity of their objectives, there is a growing need for decision-support tooling to help the operators in controlling the fleet.

In this paper, we present an automated planning system developed to assist the operators in the CoHoMa II challenge, where a fleet of robots, remotely controlled by a handful of operators, must explore and progress through a potential hostile area. In this context, we use planning to provide the operators with suggestions about the actions to consider and their allocation to the robots.

This paper especially focus on the modelling of the problem as a hierarchical planning problem for which we use a state-of-the-art automated solver.

1 Introduction

The "Battle-Lab Terre", a part of the French Army studying innovation, organized in 2022 the second version of the CoHoMa challenge [15] in order to study the collaboration between human operators and autonomous multi-robot systems.

The task was to navigate through a dangerous terrain in an Armoured Vanguard Vehicle (AVV) (Figure 1a). The land included 1m-wide red cube (Figure 1b) representing a trap said to be explosive and capable of damaging the AVV. Therefore, the human operators on board had to ensure that the AVV's environment was safe before moving it. To do this, they had to use various Unmanned Aerial Vehicles (UAVs) and Unmanned Ground Vehicles (UGVs), to perform reconnaissance missions, seek out traps, and avoid or disable them. A general system architecture of these vehicles has been studied in [7].

When the number of unmanned vehicles is too important for the number of human operators (6 UGVs and 3 UAVs for 4 human operators in our case), a decision-making aid is welcomed. This aid must decide which actions are to be performed, when, and by which vehicle. This problem of multi-robot task allocation is highly studied [10], especially when there are communications issues [1] which will be ignored in this study.

The model proposed in this paper is rooted in the CoHoMa challenge. At a high level it abstracts of emergency and rescue missions [6] such as floods controlling [14] or subterranean rescue [13], using mixed-initiative planning with automated vehicles [3].

The mission is for a group of humans to go through a hazardous zone with securable obstacles that they must avoid. Because the obstacles are unknown at the beginning of the mission, the operators have at their disposal UAVs and UGVs to explore the area, detect obstacles and secure them. The fleet of robots is typically heterogeneous: they have different capacities, in order to be complementary and be

A. Ferrando, R.C. Cardoso (Eds.): Third Workshop on
Agents and Robots for reliable Engineered Autonomy 2023 (AREA'23)
EPTCS 391, 2023, pp. 113–126, doi:10.4204/EPTCS.391.13

(a) The AVV and two UGVs

(b) A UAV detecting a trap

Figure 1: Illustrations of the CoHoMa challenge

able to secure the human movements. The obstacles will be discovered as the progression goes on, so there will be replanning steps for each event.

To simplify the interactions with the robots, their locations are discretized. Indeed, the operator does not need to have a precise representation of the robot's location for the planning process, the points of interest are sufficient. Therefore, a navigation graph as shown in the Figure 2 is used. This graph regroups the location of the vehicles, the location of the obstacles, and the objectives of the mission. Moreover, the edges of the graph are configured to forbid the access to some vehicles, *e.g.* a UAV can cross a cliff where the other vehicles cannot. This way, a unique graph can be used to store all the possible displacements.

The Figure 3a shows the Human-Machine Interface (HMI) used by a human operator to visualize the environment, the real location of the robots and the detected obstacles, *i.e.* the navigation graph with more details, on a satellite view of the terrain. The operator can interact with the map to specify events, *e.g.* an obstacle detection, and to change the mission's objectives. When the mission details have been updated, the operator can request a plan to achieve those objectives on the right side of the HMI. This plan is not sent to the robots directly. First, it is shown in the HMI (see Figure 3b) on the left side for potential modification, *e.g.* allocate an action to another robot, and for approbation. Thus, the plan needs to be as simple as possible in order to be easily understandable by the operator. Once the plan is validated, it is sent to each robot which are able to accomplish it. For example, considering the calculated plan

Move UAV from L_1 to L_5
Move UAV from L_5 to L_{10}
Move UAV from L_{10} to L_{12}

The operator does not need to know which path the vehicle will take since it is autonomous, so the tasks can be regrouped into a single task 'Move UAV from L_1 to L_{12}'. Eventually, the operator already knows

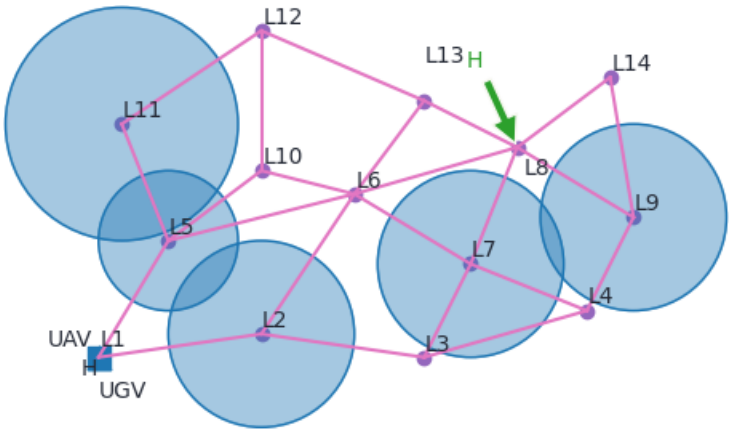
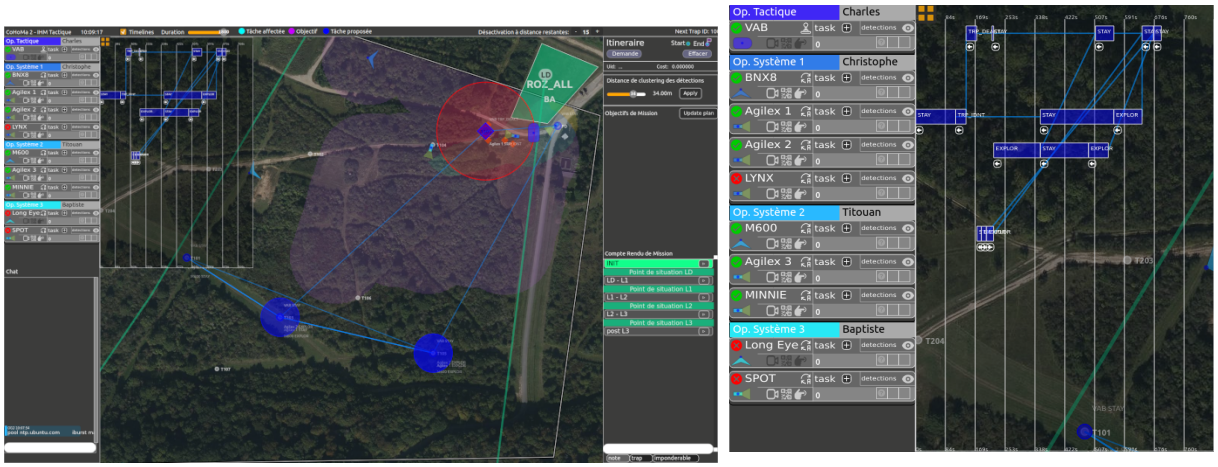


Figure 2: Example of a navigation graph where (i) the vehicles (UAV, UGV, H for Humans) are in L1 (ii) the objective is for H to go to L8 (iii) there are undetected obstacles in L2, L5, L6, L9 and L11

where the UAV is at the beginning (*i.e.* in location L_1), so the action can be transformed, and the plan can be simplified as 'Move UAV to L_{12}'. After validation, the task is sent to the concerned robot UAV, which knows how to go to the location L_{12}.



(a) Full view of the HMI

(b) Zoom on the plan visual: a line is the timeline of a robot

Figure 3: HMI used by the operator to interact with the robot fleet

Although the robots are capable of detecting obstacles, they do not modify the mission on their own because their detection cannot be perfectly accurate; they add several false obstacles next to the real one. To compensate for this, the detected obstacles are displayed in the HMI by grouping nearby obstacles together, with a customizable threshold, and operator approval is required to add the obstacle to the mission problem. This approach is generalized to all possible events. In this way, no uncertainty is taken into account in the planning process; it is taken upstream by the operator who has validated the event. Finally, as two events can occur at the same time for two different robots, replanning is not triggered automatically after each event but only when the operator requests it.

This paper will begin by presenting the necessary background for chronicle modelling. Next, a

model that is as simple as possible for a non-expert user, called the *natural* model in the following, will be proposed to show the limitations of simple models. Finally, some optimizations of this first model will be introduced, and the time needed for the planner to find a solution will be compared.

2 Background

To model the planning problem, we wish to exploit the hierarchical nature of the task where some high-level tasks to accomplished are specified by the operator that must then be refined into sets of primitives actions executable by the autonomous vehicles.

An Hierarchical Task Network (HTN)[2] can represent this kind of decomposition and is easily defined with the HDDL language [12]. This language however lacks the ability to express temporal properties of the problem such as the duration of action or deadlines. Instead, we rely on the formalism of chronicles [9] that support the specification of rich temporal planning problem. In particular, we exploit their extension for hierarchical task networks can represent combined temporal and hierarchical problems [11]. However, it does not have an input language that can represent both.

A **type** is a set of values that can be either domain constants (*e.g.* the type $Vehicle = \{ V_1, V_2 \}$ defines two vehicles objects V_1, V_2) or numeric values (*e.g.* **timepoints** are regularly spaced numerical values describing absolute times when events occur). The types can present a hierarchy, *e.g.* the type *Robot* is a subtype of $Vehicle$ meaning that a *Robot* is a $Vehicle$, but the reverse is not necessarily true. When there is a type hierarchy, an abstract root type named $Object$ is defined in order to have a decomposition tree.

A **state variable** describes the evolution of an environment characteristic over time. Generally, it is parametrized by one or multiple variables. Its value will depend on the value of the variables, *e.g.* $loc(v)$ denotes the evolution of the location of the vehicle v, its value will be $loc(V_1)$ or $loc(V_2)$ depending on the value taken by v of type $Vehicle$.

A **task** is a high-level operation to accomplish over time. Generally, it is parametrized by one or multiple variables. It is of the form $[s,e]task(x_1,\ldots,x_n)$ where s and e are timepoints denoting the start and end instants when the task occurs, $task(x_1,\ldots,x_n)$ is the task with each x_i a variable. For instance, $[2,4]Move(V_1,L_2)$ denotes the operation of moving the vehicle V_1 to the location L_2 during the temporal interval $[2,4]$. The set of available tasks of the planning problems is $\mathscr{T}$.

A **chronicle** defines the requirements of a process in the planning problem. A chronicle is a tuple $\mathscr{C} = (V,T,X,C,E,S)$ where:

- V is the set of *variables* of the chronicle. This set is split into a set of temporal variables V_T whose domains are timepoints and a set of non-temporal variables V_O.
- $T \in \mathscr{T}$ is the parametrized *task* achieved by the chronicle. The start and the end instants of the task correspond to the start and the end instants when the chronicle is active, it is its *active temporal interval*.
- X is a set of *constraints* over the variables of V. The chronicle cannot be *active* (defined bellow) if at leat one constraint is not respected over its active temporal interval.
- C is a set of *conditions* with each condition of the form $[s,e]var(x_1,\ldots,x_n) = v$ where $(s,e) \in V_T^2$ such that the temporal interval $[s,e]$ is contained in the active temporal interval of the chronicle, $var(x_1,\ldots,x_n)$ is a parametrized state variable with each $x_i \in V_O$, and $v \in V_O$. A condition is verified if the state variable $var(x_1,\ldots,x_n)$ has the value v over the temporal interval $[s,e]$. The chronicle cannot be active if at least one condition is not verified.

- E is a set of *effects* with each effect of the form $[s,e]var(x_1,\ldots,x_n) \leftarrow v$ where $(s,e) \in V_T^2$ such that the temporal interval $[s,e]$ is contained in the active temporal interval of the chronicle, $var(x_1,\ldots,x_n)$ is a parametrized state variable with each $x_i \in V_O$, and $v \in V_O$. An effect states that the state variable $var(x_1,\ldots,x_n)$ takes the value v at time e. The temporal interval $]s,e[$ is the moment when the state variable is transitioning from its previous value to its new value. During this transition, the value of the state variable is undetermined.
- S is a set of *subtasks* where each subtask is a task in $\mathscr{T}$ that must be achieved by another chronicle.

A chronicle can be **active** or not, defining whether the chronicle is present in the final solution. If the chronicle is not active, then the planner must find another chronicle achieving the same task to replace it.

We make the distinction between three types of chronicles: the **action chronicle** which has effects but no subtasks (*i.e.* $S = \emptyset$), the **method chronicle** which has subtasks but no effects (*i.e.* $E = \emptyset$), and the **initial chronicle** encoding the initial state as effect and the objectives of the problem as conditions and subtasks, it is the only one which does not have a task T to achieve (*i.e.* $T = \emptyset$).

As an alternative to specifying chronicles manually, the AIPlan4EU project[1] offers a Python API[2] for modelling different kinds of planning problems, notably temporal and hierarchical ones. The corresponding problems map almost immediately to the chronicles defined above. The python API for constructing planning problems is especially useful in our case where the new problems are defined online, as the situation evolves during the mission.

3 Initial Model

According to the mission specification, the humans need to be able to move while the autonomous vehicles need to move, explore to detect obstacles and secure them. This way, a list of high-level tasks appears:

- $[s,e]goto(v,l)$: The vehicle v (humans, UAV or UGV) goes to the location l
- $[s,e]explore(r,f,t)$: The robot r (UAV or UGV) explores the path from the location f to t
- $[s,e]secure(r,o)$: The robot r secures the obstacle o

From this list, one can easily extract the type hierarchy shown in the Figure 4. The *Obstacle* allows handling different types of obstacles for the *secure* task, *e.g.* in a fire rescue mission we could image to use different types of extinguishers (water, CO_2 or powder), each one for a different type of obstacle.

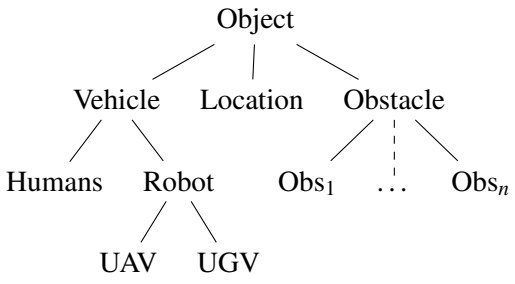


Figure 4: Type hierarchy

[1] https://www.aiplan4eu-project.eu/
[2] https://github.com/aiplan4eu/unified-planning

3.1 Goto task

A vehicle needs to be able to go from a location to another. However, a human, a UAV and a UGV does not move the same way. A human will *walk* while a UAV will *fly* and a UGV will *roll* on land. Therefore, we obtain the three following action chronicles:

$[s,e]walk(h,f,t)$

variables:	Humans h
	Locations f (from) and t (to)
task:	$[s,e]walk(h,f,t)$
constraints:	$f \neq t$
	$e-s = dur(h,f,t)$
conditions:	$[s,s]loc(h) = f$
	$[s,e]path(f,t) = \top$
	$[s,e]explored\ air(f,t) = \top$
	$[s,e]explored\ ground(f,t) = \top$
	$[s,e]obstacle(f,t) = \bot$
effects:	$[s,e]loc(h) \leftarrow t$

The humans h can move to the location t only if (i) the humans are in the location f at the beginning of the chronicle (ii) there is an edge from f to t in the navigation graph (iii) the path has been explored by a UAV and a UGV (iv) there is no obstacle affecting the path.
At the end of the chronicle, the humans h will be at the location t.
The state variable $dur(v,f,t)$ represents the duration taken by the vehicle v to go from the location f to the location t. It depends on the distance between f and t, and on the speed of the vehicle v.

$[s,e]fly(a,f,t)$

variables:	UAV a
	Locations f (from) and t (to)
task:	$[s,e]fly(a,f,t)$
constraints:	$f \neq t$
	$e-s = dur(a,f,t)$
conditions:	$[s,s]loc(a) = f$
	$[s,e]path(f,t) = \top$
effects:	$[s,e]loc(a) \leftarrow t$

The UAV a can move to the location t only if (i) the UAV is in the location f at the beginning of the chronicle (ii) there is an edge from f to t in the navigation graph
At the end of the chronicle, the UAV a will be at the location t.
As for the *walk* chronicle, the duration is specified with the constraint $e-s = dur(a,f,t)$.

The chronicle $[s,e]roll(g,f,t)$ is similar to the chronicle $[s,e]fly(a,f,t)$ by replacing the UAV a by the UGV g. However, the distinction is made because in a more detailed model it could be more conditions and effects making a difference between the air and ground movements.

The Figure 5 shows a possible decomposition of the $[s,e]goto(v,t)$ task made by a user. There are four possibilities for the vehicle v to go to the location t:

- It is already at the location, *i.e.* $loc(v) = t$, then there is no operation (*Noop*) to do. The associated chronicle is detailed in the Figure 6a.
- It is a UAV, then it flies to another location and retry to go to t from this new location. The recursion will end when $loc(v) = t$ with the *Noop* method. The associated chronicle is detailed in the Figure 6b.
- In the same way as UAVs, the UGVs and humans will move and try again. The associated chronicles are similar to the one of *UAV*.

3.2 Explore task

The robots need to be able to explore an edge the navigation graph in order to detect the obstacles and secure the path for the humans. To explore the edge going from the location f to the location t, the robot

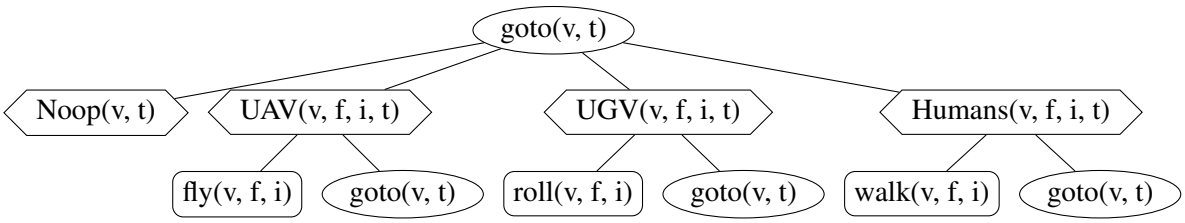


Figure 5: Natural decomposition of the goto task where (i) rectangles are action chronicles (ii) diamonds are method chronicles and (iii) ovals are tasks to achieve

$[s,e]Noop(v,t)$
`variables:` Vehicle v
Location t
`task:` $[s,e]goto(v,t)$
`conditions:` $[s,e]loc(v) = t$

(a) *Noop* method

$[s,e]\mathsf{UAV}(a,f,i,t)$
`variables:` UAV a
Locations f (from), i (intermediate), t (to)
`task:` $[s,e]goto(a,t)$
`conditions:` $[s,e]loc(a) \neq l$
`subtasks:` $[s_1,e_1]fly(a,f,i)$
$[s_2,e_2]goto(a,t)$
`constraints:` $e_1 < s_2$

(b) *UAV* method

Figure 6: Some method chronicles used to decompose the goto task

r needs to be either in location f or location t. Therefore, there are two methods to explore an edge (shown in Figure 7):

- going to the location f then explores from f to t: *forward* method
- going to the location t then explore from t to f: *backward* method

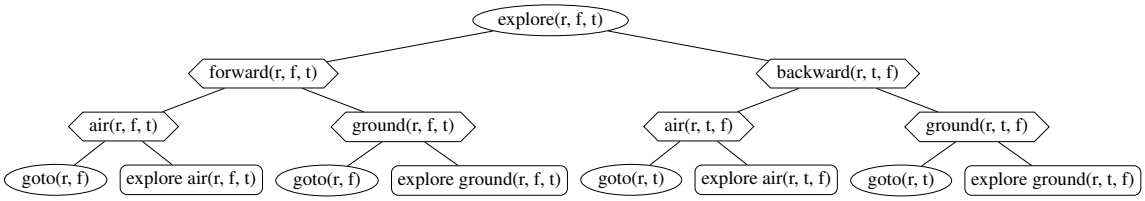


Figure 7: Natural decomposition of the explore task

These two methods can be accomplished by a UAV with the *air* method or by a UGV with the *ground* method. The distinction between the two associated actions is that one effect of the *explore air* action will be $explored\ air(f,t) \leftarrow \top$, and for the *explore ground* action it will be $explored\ ground(f,t) \leftarrow \top$. These two state variables are used in conditions of the *walk* action in order for the humans to move securely.

As for the movement actions, the duration of an exploration is based on the state variable $dur(v,f,t)$.

3.3 Secure task

Finally, the robots need to be able to secure detected obstacles so that they can be crossed by humans. Because there are several types of obstacles (see Figure 4), there will be several methods to secure them as shown in the Figure 8.

We made the assumption that the robot r needs to be close to the obstacle o to secure it for every method. In the case where it is not needed, *e.g.* in a military context as CoHoMa II where some obstacles representing enemy's troops could be secured in distance with artillery fire, the associated *goto* task should be removed.

For the following simulations, we consider only one way to secure an obstacle with the duration of 15 minutes.

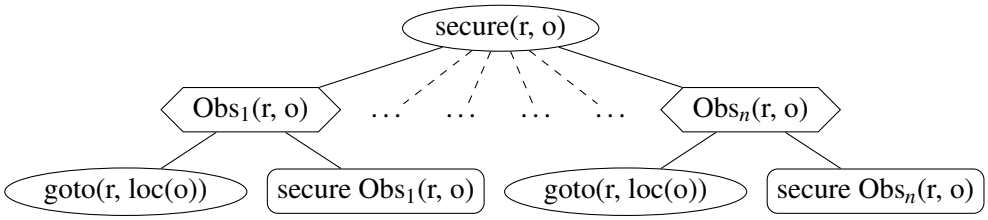


Figure 8: Natural decomposition of the `secure` task

3.4 Initial State and Objectives

Once the different high-level tasks have been defined, an initial chronicle needs to be specified to encode the initial state and the objectives.

$[s,e]initial$
`constraints:` $s = 0$
`effects:` $[s,s]loc(H) \leftarrow L1$
$[s,s]loc(UAV) \leftarrow L1$
$[s,s]loc(UGV) \leftarrow L1$
$[s,s]path(L1,L2) \leftarrow \top$
$\vdots$
$[s,s]path(L12,L13) = \top$
`subtasks:` $[s_1,e_1]goto(H,L8)$

The initial chronicle starts at the timepoint 0 and ends at the timepoint e. This timepoint can be used to specify objectives.
Initially, the vehicles are located to the location L_1 and the different paths are specified. All the unspecified state variable values are considered to be false.
The objective of the problem is for the humans to go to the location L_8.

With this initial chronicle, the planner will try to achieve the subtask $[s_1,e_1]goto(H,L8)$. Since there are no explored paths, this is impossible without the intervention of a robot, but they cannot explore because exploration tasks are not present in the initial chronicle's subtasks. However, the robots are not expected to explore all the paths, they are expected to be free to do whatever they want in order to help the humans.

3.5 Freedom Task

In order to achieve that, the $freedom(v)$ task (see Figure 9) is added. It allows the vehicle v to go to another location or to explore a path without any constraints. Once the robot will have nothing more to do, the $freedom\ noop(v)$ method will allow it to stop.

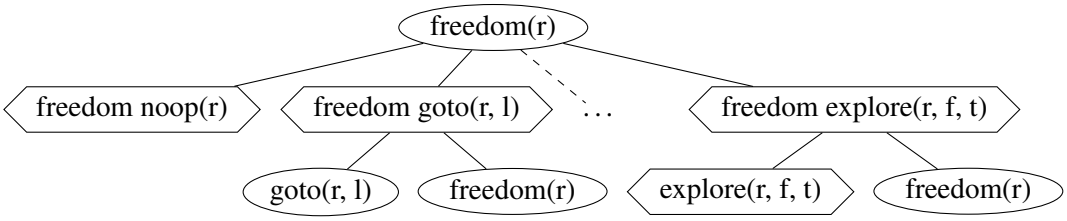


Figure 9: Natural decomposition of the freedom task

Next, the three subtasks $[s_2, e_2]freedom(UAV)$, $[s_3, e_3]freedom(UGV)$, and $[s_4, e_4]freedom(H)$ can be added to the initial chronicle's subtasks. Note that the $freedom(H)$ task only allows the humans to go wherever they want, they cannot do exploration even if it is present in the decomposition of the task. This is caused by the type hierarchy and the definition of the *explore* task that only take robots as parameters.

In general these freedom tasks allow the planner to insert some classes of actions in the plans regardless of the rest of the hierarchy. In this sense, it simulates in the HTN the notion of task insertion [8], where any action can be inserted along the hierarchy. It is in particular close to the *task-independent* action in FAPE [5], where only a subset of the actions are allowed to be inserted arbitrarily.

3.6 First Planning Results

Considering ground truth shown in the Figure 2, the vehicles are in the location L_1 and the humans need to go to the location L_8, but there are undetected obstacles on the path. Because the terrain is not fully known at the beginning of the mission, a replanning step is needed when the operator adds some details to the mission, *e.g.* when an obstacle is detected by a robot.

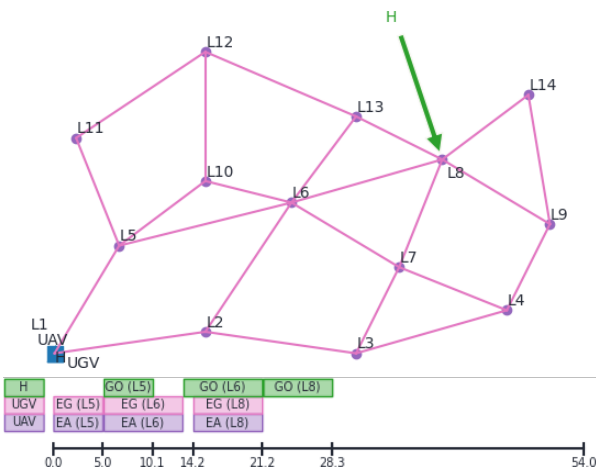
Initially, the knowledge of the terrain is empty. Therefore, the decision-making aid has the navigation graph shown in the Figure 10a and will propose the associated plan. This plan is to take the shortest route to the goal, with the robots ahead of the humans to secure the path. The planning operation has been done with the Aries planner [4], it took 333.59s to find the optimal solution.

During the execution of that plan, the robots detect an obstacle at the location L_5 (see Figure 10b). A new plan is proposed based on the new knowledge of the terrain. The planner believes it's quicker to go back and explore a new route than to secure the current obstacle. This plan has been found in 328.25s.

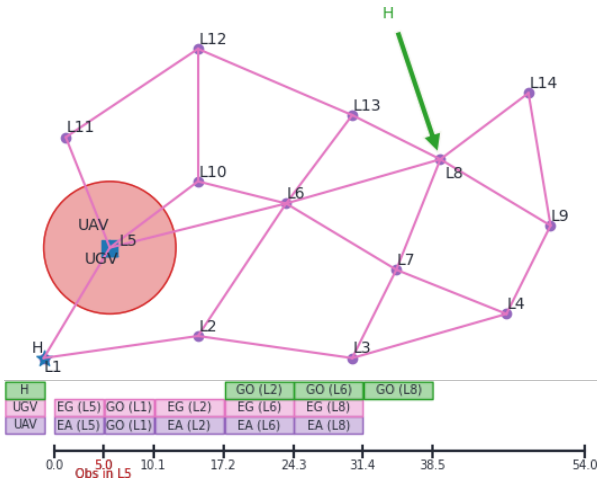
Finally, a new obstacle is discovered at the location L_2 (see Figure 10c). Again, a new plan is proposed based on the new knowledge of the terrain. This time, it is faster to secure the current obstacle and go to the target. The planner took 308.72s to find this plan.

4 Optimizations

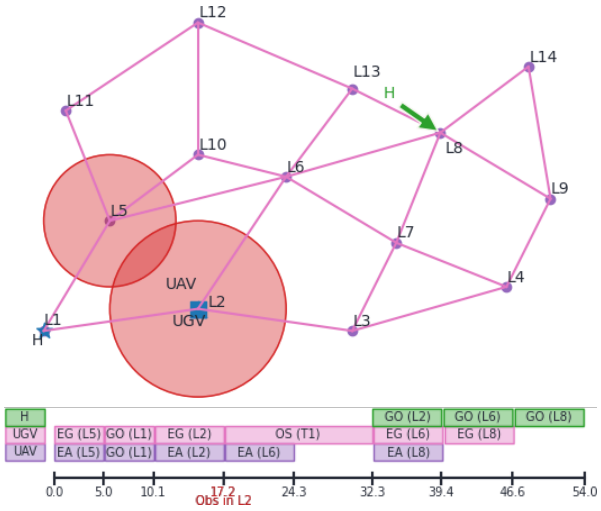
While reasonable, planning times of a handful of minutes are far from ideal in mixed-initiative planning context, especially when task durations are faster than the minute. To reduce this time, one could ask for the first solution found by the planner (instead of an optimal solution) with the risk of handing out bad quality solutions. Instead, in this section, we introduce some modifications that can be brought to the planning model in order to speed up the planning process.



(a) Step 1 - Beginning of the mission



(b) Step 2 - Obstacle detection at L_5



(c) Step 3 - Obstacle detection at L_2

Figure 10: Terrain knowledge with their associated plan to solve the problem
EA: Explore Air *GO*: Goto (with *walk*, *fly*, or *roll*)
EG: Explore Ground *OS*: Secure Obstacle

4.1 Recursive Tasks

To find a solution, the planner needs to scan the search tree and prune the branches that lead to no solution. Therefore, the model should use the least possible recursive tasks in order to reduce the size of the search tree.

Considering the *goto* task (see Figure 5) and $n > 0$ the decomposition depth, *i.e.* the number of times *goto* leaves are replaced by the decomposition. Note that if a leaf is not decomposed, the associated method is removed from the three since it will not be applicable. Then, the size of the tree, *i.e.* the number of nodes, is $2+3*4^n = \mathcal{O}(4^n)$ which is **exponential**.

However, one can notice that all the methods have the same pattern. There is an action followed by the recursive call to the *goto* task. Then, the actions can be grouped in a *goto once* task and the *goto* task

can be moved outside in order to be present only once as shown in the Figure 11.

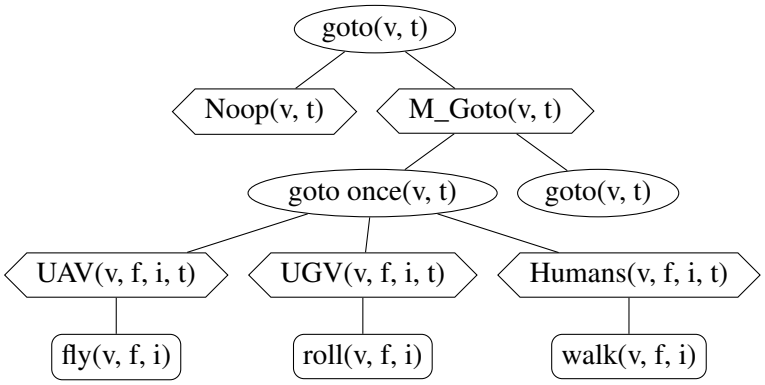


Figure 11: Optimized decomposition of the goto task

With this new decomposition and $n > 0$ the decomposition depth, the size of the tree is $2 + 10 * n = \mathcal{O}(n)$ which is **linear**.

This method can be applied to all the tasks. For *goto*, *explore*, and *secure*, it is the call to *goto* which will be extracted. For the *freedom* task, it is the recursive call to *freedom*.

4.2 Complete Navigation Graph

One of the mission's assumptions is that the calculated plan is not intended directly for the robots, but for a human operator to approve, so the plan must be as simple as possible, which translates in particular into the aggregation of movement actions.

The edges of the navigation graph can be set to prohibit the passage of certain vehicles. Therefore, one could make the graph complete, *i.e.* each node is connected to all the others, and make an edge allowed for a given vehicle if :

- the vehicle is a robot, humans need to know exactly where they are going
- there is a path in the initial graph corresponding to that new edge
- the vehicle is allowed to go through all the edges of this path
- the time taken by the vehicle to pass this new edge is the time taken to cover the associated path

This way, the action of going from L_1 to L_9 for a robot can be done with only one decomposition of the *goto* task rather than 4 decomposition without this navigation graph manipulation. As a result, the search tree will be smaller, and the solution will be found more quickly.

4.3 Objectives as Conditions

The initial chronicle defines the objective as a subtask. That means the given subtask needs to be accomplished. However, the subtasks also contain three *freedom* tasks in order to the vehicles to do whatever they want to complete the objective.

Looking closer, one can see that this allows many opportunities to achieve the objective $goto(H, L8)$, which are all the possible combinations of the two tasks $goto(H, L8)$ and $freedom(H)$, both allowing the humans to move.

To avoid that, the objective can be encoded in another way. The objective is not for the humans to go to the location L_8, but to be at the location L_8 at the end, *i.e.* $loc(H) = L_8$. Therefore, the initial chronicle can be updated as shown below. With this new encoding, the only way for the humans to be at the location L_8 is to use the $goto(H, L8)$ hidden in the $freedom(H)$ subtask. As a result, the search tree will be reduced.

$$
\begin{array}{ll}
[s,e]initial & \\
\texttt{constraints:} & s = 0 \\
\texttt{effects:} & [s,s]loc(H) \leftarrow L1 \\
 & [s,s]loc(UAV) \leftarrow L1 \\
 & [s,s]loc(UGV) \leftarrow L1 \\
 & \ldots \\
\textbf{conditions:} & [e,e]loc(H) = L8 \\
\texttt{subtasks:} & [s_1,e_1]freedom(H) \\
 & [s_2,e_2]freedom(UAV) \\
 & [s_3,e_3]freedom(UGV)
\end{array}
$$

4.4 Final Planning Results

Considering the same mission studied in the previous section, the planner found the same plans as shown in the Figure 10. This demonstrates that the proposed optimizations do not change the problem represented by the model, both are equivalent. However, as shown in the Table 1, the planner is 95% faster with these optimizations.

	Step 1	**Step 2**	**Step 3**
Natural model	333.59s	328.25s	308.72s
Optimized model	13.61s	14.17s	9.19s
Global reduction	95.9%	95.7%	97.0%

Table 1: Planning time with and without the proposed optimizations

As these optimizations are independent of the domain, the same results should be observed in other use cases.

5 Conclusion

In this paper, we presented a planning-based decision-making aid that exploits a hierarchical task planner for the control of a fleet of robots in an exploration scenario. A first natural model of the problem has been proposed. We then proposed some domain-agnostic optimization of this initial model, which resulting in the planner being at least 20 times faster to provide an optimal solution.

Some assumptions have been made in the current model, notably that the robot's battery level is infinite. It could be interesting to be able to represent these kinds of resources in order to accomplish more complex missions. Moreover, the planner is optimizing the makespan of the plan, *i.e.* it tries to make the plan as short as possible in time. It could be useful to associate a cost to each action to optimize the global cost of the plan, *i.e.* the sum of the present action's cost. This way, it could be possible to minimize, for example, the total distance travelled by the human group.

References

[1] Patrick Bechon (2016): *Planification multirobot pour des missions de surveillance avec contraintes de communication*. Theses, INSTITUT SUPERIEUR DE L'AERONAUTIQUE ET DE L'ESPACE (ISAE). Available at https://hal.science/tel-01434167.

[2] Pascal Bercher, Ron Alford & Daniel Höller (2019): *A Survey on Hierarchical Planning – One Abstract Idea, Many Concrete Realizations*. In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*, International Joint Conferences on Artificial Intelligence Organization, Macao, China, pp. 6267–6275, doi:10.24963/ijcai.2019/875. Available at https://www.ijcai.org/proceedings/2019/875.

[3] Giuseppe Bevacqua, Jonathan Cacace, Alberto Finzi & Vincenzo Lippiello (2015): *Mixed-Initiative Planning and Execution for Multiple Drones in Search and Rescue Missions*. *Proceedings of the International Conference on Automated Planning and Scheduling* 25, pp. 315–323, doi:10.1609/icaps.v25i1.13700. Available at https://ojs.aaai.org/index.php/ICAPS/article/view/13700.

[4] Arthur Bit-Monnot (2023): *Experimenting with Lifted Plan-Space Planning as Scheduling: Aries in the 2023 IPC*. In: *Proceedings of 2023 International Planning Competition*.

[5] Arthur Bit-Monnot, Malik Ghallab, Félix Ingrand & David E. Smith (2020): *FAPE: a Constraint-based Planner for Generative and Hierarchical Temporal Planning*. *CoRR* abs/2010.13121. Available at https://arxiv.org/abs/2010.13121.

[6] Jeffrey Delmerico, Stefano Mintchev, Alessandro Giusti, Boris Gromov, Kamilo Melo, Tomislav Horvat, Cesar Cadena, Marco Hutter, Auke Ijspeert, Dario Floreano, Luca M. Gambardella, Roland Siegwart & Davide Scaramuzza (2019): *The current state and future outlook of rescue robotics*. *Journal of Field Robotics* 36(7), pp. 1171–1191, doi:10.1002/rob.21887. Available at https://onlinelibrary.wiley.com/doi/10.1002/rob.21887.

[7] Emile Le Flecher, Alexandre La Grappe & Geert De Cubber (2022): *iMUGS - A ground multi-robot architecture for military Manned-Unmanned Teaming*.

[8] Thomas Geier & Pascal Bercher (2011): *On the Decidability of HTN Planning with Task Insertion*. In Toby Walsh, editor: *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, IJCAI/AAAI, pp. 1955–1961, doi:10.5591/978-1-57735-516-8/IJCAI11-327.

[9] Malik Ghallab, Dana S Nau & Paolo Traverso (2004): *Automated Planning: Theory and Practice*. Morgan Kaufmann Publishers Inc.

[10] Maria Gini (2017): *Multi-Robot Allocation of Tasks with Temporal and Ordering Constraints*. *Proceedings of the AAAI Conference on Artificial Intelligence* 31(1), doi:10.1609/aaai.v31i1.11145. Available at https://ojs.aaai.org/index.php/AAAI/article/view/11145.

[11] Roland Godet & Arthur Bit-Monnot (2022): *Chronicles for Representing Hierarchical Planning Problems with Time*. In: *ICAPS Hierarchical Planning Workshop (HPlan)*, Singapore, Singapore. Available at https://hal.archives-ouvertes.fr/hal-03690713.

[12] Daniel Höller, Gregor Behnke, Pascal Bercher, Susanne Biundo, Humbert Fiorino, Damien Pellier & Ron Alford (2020): *HDDL: An Extension to PDDL for Expressing Hierarchical Planning Problems*. *Proceedings of the AAAI Conference on Artificial Intelligence* 34(06), pp. 9883–9891, doi:10.1609/aaai.v34i06.6542. Available at https://aaai.org/ojs/index.php/AAAI/article/view/6542.

[13] Viktor Orekhov & Timothy Chung (2022): *The DARPA Subterranean Challenge: A Synopsis of the Circuits Stage*. *Field Robotics* 2(1), pp. 735–747, doi:10.55417/fr.2022024. Available at https://fieldrobotics.net/Field_Robotics/Volume_2_files/Vol2_24.pdf.

[14] Pan Tang (2013): *A temporal HTN planning paradigm and its application in flood controlling*. In: *IEEE Conference Anthology*, pp. 1–4, doi:10.1109/ANTHOLOGY.2013.6785016.

[15] Battle-Lab Terre (2022): *Challenge CoHoMa 2*. Available at https://www.defense.gouv.fr/sites/default/files/aid/Support%20CoHoMa%202023.pdf.

ORTAC+ : A User Friendly Domain Specific Language for Multi-Agent Mission Planning

Caroline Bonhomme
Safran Electronics and Defense
Massy, France
ONERA
Toulouse, France
caroline.bonhomme@safrangroup.com

Jean-Louis Dufour
Safran Electronics and Defense
Massy, France
jean-louis.dufour@safrangroup.com

A tactical military unit is a complex system composed of many agents such as infantry, robots, or drones. Given a mission, an automated planner can find an optimal plan. Therefore, the mission itself must be modeled. The problem is that languages like PDDL are too low-level to be usable by the end-user: an officer in the field. We present ORTAC+, a language and a planning tool designed for this end-user. Its main objective is to allow a natural modeling of the mission, to minimize the risk of bad modeling, and thus obtain reliable plans. The language offers high-level constructs specifically designed to describe tactical missions, but at the same time has clear semantics allowing a translation to PDDL, to take advantage of state-of-the-art planners.

1 Introduction

A tactical military ground unit is a complex system composed of many agents, such as infantry sections, armored units, ground robots, or flying drones. A typical mission lasts one day and is a mix of displacements on a road network of 20 km between given start and finish points and actions. We deal only with the reconnaissance action of a route or an area when an enemy presence is suspected. For us, the reconnaissance action is equivalent to a speed reduction of the progression on a route and exploration of an area. The progression and exploration should be done more safely, thus it is slower. It could be seen as the first visit to a location. This brings us to a variant of the classic Multi-Agent Path Finding (MAPF) problem [1]. Of course, if we consider moves as actions, this is also a multi-agent planning problem [2], but classification as a MAPF problem is more accurate.

Given a mission, if it is described in a programming language, an automated planner can find an optimal plan. Therefore the mission itself must be modeled. The automated planning community has developed the PDDL language in order to specify high level planning problems [3]. However, the specification of the mission should be done by an officer in the field who is not an expert in automated planning. Languages like PDDL are too low-level to be usable by the defined end-user who is not an expert in planning. This is why we present ORTAC+. Its main objective is to allow a natural modeling of the mission, so as to minimize the risk of errors on the modeling of the desired mission, and thus obtain reliable plans. The language is composed by high-level constructs specifically designed to facilitate the specification of tactical missions, with a clear semantic allowing a translation to PDDL, to take advantage of state-of-the-art planners. Planning is done offline, and the plans are deterministic, with no contingency. The default assumption is that the enemy, when suspected, is in fact not present. Otherwise, if there is contact with the enemy, the execution of the plan is de facto interrupted in the field, and the officer must replan with modified objectives.

We proceed by presenting related work of the paper in section 2. The domain-specific language ORTAC+ is presented in section 3, the operational context in section 4, and the conclusion in section 5.

A. Ferrando, R.C. Cardoso (Eds.): Third Workshop on
Agents and Robots for reliable Engineered Autonomy 2023 (AREA'23)
EPTCS 391, 2023, pp. 127–133, doi:10.4204/EPTCS.391.14

2 Related Work

ORTAC stands for "Optimal Resource and Technical Action Control". It is a multi-agent planning language and tool combining constraint-solving techniques with advanced search strategy to deconflict single-agent plans [4]. At the language level, ORTAC is innovative in the constraints of coordination between units introducing the notion of "support" explained later [4]. The proper term for that is "cross-schedule dependencies" [5], typically in the case of "coalition formation."

PDDL is the standard planning language used commonly by the automated planning community. One of the purposes of PDDL was to benchmark planning algorithms. Since its creation in 1998 for the International Planning Competition [6], the language evolved to model more complex planning problems as temporal or numerical planning [3]. The Competition of Distributed and Multiagent Planners [7] benchmarks some centralized and decentralized algorithms with an extension of PDDL for multiagent planning as a reference language. ORTAC and PDDL are powerful tools for modeling high-level missions and finding a corresponding plan, nevertheless, they cannot be used by an end-user.

In recent years, an effort on designing domain specific languages (DSL) for non-expert in robotics raised in the research community. Indeed, some of these languages lack clear semantics [8]. Additionnaly, users are required to define the behavior of each robot. Consequently, the autonomy of the system is hindered. Theses systems lack planning capabilities, making it challenging to initiate the integration of planning tools. The research community uses PDDL as the standard task planing language. However, its integration into robotics remains limited. To remedy this issue, efforts were done to integrate planning in robotics [9].

3 ORTAC+

ORTAC+ is a tool that assists a military officer in planning a mission. It is a language that aims to be expressive with well-defined syntax and semantics. To do that, the language should allow the description of a given mission and be compatible with automated planners. A military mission is limited in space and time. The first version of ORTAC+ emphasizes the spatial aspects of the mission. The temporal aspects will be added in future works.

3.1 Problem Definition

The missions could be seen as a multi-agent planning problem where each agent collaborates to reach a global goal. However, we only consider agents movements. Thus, the model is a variant of a MAPF problem. A classical MAPF problem is represented with a set of agents with an initial position and a target position, in an undirected graph. The solution of a classical MAPF problem is a set of single-agent paths that does not collide [1]. ORTAC+ allows the description of this kind of problem but aims to be more general with the following variants:

- Targets are no longer assigned to each agent but to teams of agents, leading to the possibility of having less target positions than agents. Several agents could be assigned to the same target position but only one of them should reach it.
- Spatial constraints are added to the problem. The constraint can be forbidden or enforced vertex for specific agent.
- Agents can be heterogeneous. While classical MAPF problems harnesses agents with similar characteristics, our problem differs in this aspect.
- The position of an agent is not only on vertices but can be on edges too. The path of an agent is then a succession of vertices and edges.

3.2 Modelisation

Our DSL describes the mission within a three axis representation:

- A geographical representation, corresponding to an undirected graph $G = (V, E)$, with V the space of vertices which models geographic points and E the space of edges that links one vertice to another. If there is an edge between two vertices, it means that there is a path between these two points, a road for example.
- A resource representation, including the agents involved in the mission and their characteristics. Each element of the description is an object with attributes.
- An operational representation where the constraints and the goal will be defined.

This approach allows a structured and organized representation of critical information, enabling a comprehensive analysis of a complex mission. While the geographic and resource representations correspond to static meta-data available at the start of the mission, the operational representation corresponds to flexible operational and tactical elements from the mission, allowing the definition of the goal and mandatory waypoints. Furthermore, some components of the description are objects with attributes. The type of mission modeled by ORTAC+ deals with three types of objects: the nodes, the edges, and the agents. By leveraging the three axes, we enable an in-depth portrayal of mission-critical information with a more expressive and accessible representation. It is worth noting that the agents involved in the missions are usually humans, but the tool aims to be compatible with robots.

During the conception phase, the graph is pre-defined or already given. The mission specification includes the available resources and the operational elements within the system with high-level predicates. Each predicate has a logical meaning and is a constraint for a planning algorithm.

3.2.1 Resources

The resources encompass the agents, their initial states, and their characteristics. The *agent_define(init, characteristics)* predicate instantiates the agent, creating the object agent with its initial position *init* and its *characteristics*. It is still possible to add attributes to the agent after its instantiation.

3.2.2 Operational Predicates

This representation is the modular part of the modeling. Indeed, modeling a complex mission can lead to the absence of solutions. In this part the user can change some elements of the problem and relax some constraints to find a plan. The planning tool allocates a path for each agent to reach the goal, and the tactical interpretation of the mission remains to the user.

Table 1 presents the high-level predicates introducing the constraints. Each predicate has a description and a logical meaning. For example, the predicate *node_goal()* can allocate a node goal to an agent. The predicates aim to give a declarative specification.

Besides, the predicates can be generalised so that they operate on lists of objects and no longer individual objects. To facilitate modeling, when a "generalised" predicate is applied to an individual object, this is automatically replaced by the application on the singleton containing that object. So the preceding predicate is in fact syntactic sugar for: *node_goal([node],[agent])*. The general case *node_goal(node_list,agent_list)* has the following meaning:

$$\forall n \in node\_list, \exists a \in agent\_list, a \text{ is in } n \text{ at } t_{final} \tag{1}$$

The $\forall$ *n* is in fact a general pattern to which all 'list-generalized' predicates conform, so as to facilitate their understanding: **the first list argument can always be "and-expanded"**, which means:

Predicates	Description	Logic formula
node_goal(node, agent)	The location of agent should be node at the final state	$agent.loc(t_{final}) = node$
node_visit(node, agent)	Agent should visit node during execution	$\exists t \in [\![t_{initial}; t_{final}]\!]$, $agent.loc(t) = node$
edge_visit(edge, agent)	Agent should visit edge during execution	$\exists t \in [\![t_{initial}; t_{final}]\!]$, $agent.loc(t) = edge$
node_avoid(node, agent)	Agent should never visit node	$\forall t \in [\![t_{initial}; t_{final}]\!]$, $agent.loc(t) \neq node$
edge_avoid(edge, agent)	Agent should never visit edge	$\forall t \in [\![t_{initial}; t_{final}]\!]$, $agent.loc(t) \neq edge$
node_supported_from($node_1$, $node_2$)	An agent can visit $node_1$ only if another agent supports him in $node_2$	$\forall t \in [\![t_{initial}; t_{final}]\!], \forall agent$, $agent.loc(t) = node_1 \Rightarrow$ $\exists agent_2 \neq agent$, $agent_2.loc(t) = node_2$

Table 1: Existing predicates in ORTAC+ DSL. Other predicates can be constructed

predicate([o1,o2], list2, ...) ⇔ predicate(o1, list2, ...) & predicate(o2, list2, ...)

Because of the multi-agent nature of the model, coordination constraints could be required. We are interested in the coordination presented in ORTAC [4]. The main coordination predicate uses quantitative time, but if we restrict ourselves to causality, it boils down to: *support(unit1,node1,unit2,node2)* which means: "when unit1 goes through node1 (if it goes there), unit2 must be in node2". For this to make sense, node1 and node2 must be close enough, and from a tactical point of view, it is the standard way to progress in dangerous areas. However, we are no longer specifying the mission, but have begun designing the plan. We will call ORTAC's description style "imperative", and ORTAC+ will add a new style that we will call "declarative": *node_supported_from(node1,node2)* which means: "when an agent goes through node1, another agent must be in node2". This first addition to ORTAC will be called "anonymization", it is not a generic evolution because it is only suitable for certain predicates like *support* or *goal*. And concerning goals, their "anonymization" corresponds to an interesting variant of MAPF initially called "permutation-invariant" [10] and today called "anonymous" [11].

3.2.3 Ontology

Considering the specificity of our language for its domain, the proposal entails adding a knowledge base to augment its capability. This knowledge base is an ontology designed to help the mission specification. The characteristics of the agents is defined with the declarative predicate presented in the section 3.2.1.

It is possible to specify a constraint involving several agents. For example, considering a mission with two types of agents, an agent can be a "UAV" for unmanned aerial vehicle or a "UGV" for unmanned ground vehicle. This characteristics will be define as an attribute of the agents. Instead of writing: *node_goal(14, [agent1, agent2])*, it is possible to write *node_goal(14, "UGV")* if agent1 and agent2 have the attribute UGV, meaning at the end of the mission a "UGV" should be at the node 14.

Another example of the combined efficiency of objects and ontologies is: french infantry uses impressive vehicles called "VBCI" which need some place to maneuver. If the officer wants to ban the progression of these vehicles on narrow roads, he has to stipulate: *edge_avoid("width < 10", "VBCI")* in spite of *edge_avoid(list_of_edges_with_width_less_than_10, list_of_units_with_VBCI)*

Currently, the ontology is hierarchical. An attribute can be a leaf in knowledge tree. For instance, if

agent1 is a UGV on wheels and agent2 is a UGV on caterpillars. There are the relations: a UGV with wheels is a UGV, and the UGV on caterpillars is a UGV. They have the same parent. The ontology allows to specify *node_goal(14, "UGV")* to involve agent1 and agent2 in the constraint.

4 Operational Context

ORTAC+ is built to specify military missions in order to assist an officer planning the mission. In the following part, an example of mission is described, then its representation in the proposed language.

4.1 Description of the Mission

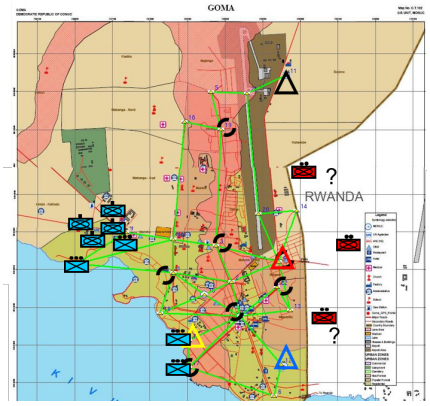


Figure 1: Cartography of the mission in Goma, Congo.

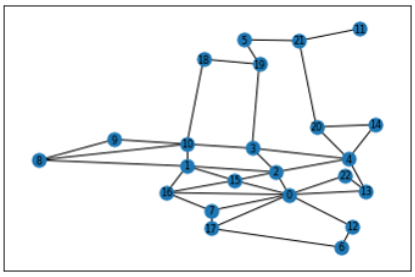


Figure 2: Geographical mission modelisation using graphs

Congolese rebels, supported by elements from Rwanda, are located in the eastern region of Goma. The rebels raise a significant threat to the peace and security of the city. The mission focused on two specific objectives: securing the United Nation (UN) point at the border, between Congo and Rwanda, and preparing for the potential evacuation of civilians at the airport. With forces comprising 4 distinct military sections and 4 motorized companies positioned along the west cost of the city, this mission implies the recognition of crucial points and axes leading to the destination goals.

Operational constraint. Units follow specific rules of engagements during the execution of their tasks:

- Simultaneous coexistence of two distinct units at the same spatial coordinates is not allowed.
- Any unit advancing on a main street or an intersection must be supported by another unit recently deployed to an adjacent position.

Tactical information. The mission encompasses critical data as a foundation for modeling and planning:

- The airport: the airport is a crucial point and must be controlled during the mission, with a designated convoy of one company and one unit equipped with night vision glasses will be tasked to evacuate people from the airport.
- Securing the UN point: The UN point is identified at the border, requiring strict security measures, with a priority to maintain control over
- Support team: A support team will be positioned at the southern region of the operation area

```
#geography
vertices_info(places)
edges_info(links)
# ontologie
categorie("mortar", ["mortar-60", "mortar-81", "mortar-120"])
# ressources
unit1, unit2, unit3, unit4 = agent_define([9,9,9,9], "company", "VBCI")
unit5, unit6, unit7, unit8 = agent_define([7,17,8,8], "section")
unit5.attribute("grenade-launcher-F1", "mortar-60")
# operation
node_goal(11, "company")
node_goal(4, company or (section and mortar))
node_visit(9, unit1)
edge_avoid([(10,18),(3,19),(4,20),(4,14),(3,4),(16,7)], unit3)
node_supported_from(3, 18) # node 3 should have a support at node 18
node_supported_from([15, 16], 1) # node 15 and node 16 should have a support at node 1
```

Figure 3: Specification of the "Secure the UN in Goma" mission in ORTAC+

4.2 Mission specification in ORTAC+

An example of the mission specification in ORTAC+ is given in Figure 3. The mission goal is the securing of the United Nations. The cartography of the mission is given in Figure 1. Friendly forces are blue rectangles at the west of Goma. Target positions are triangles at the east near to enemies in red rectangles. The black triangle correspond to the airport and the red one the UN. The 3 axis of representation (geography, resources and operations) appear clearly combined with the construction of a hierarchical ontology. Geography elements of the mission are supposed to be already given. It is represented in Figure 2 The Goma mission involves two types of agents: compagnies and sections. The predicates "*agent_define()*" is used to instantiate the units. At the initial state, the 4 mechanized compagnies are at the same node (9). This violates the constraint that only one unit can be at a node at the same time. That's why the notion of capacity is introduced for nodes and edges. By default the capacity of a node or an edge will be 1 but it can be changed for some exceptions. The method "*attribute()*" allows the user to add some characteristics to the units. Then, the high level predicates described earlier are used to add the constraints of the mission. The final state is defined with the predicate "*node_goal()*" with the help of the knowledge base. A compagny should be at the node 11, corresponding to the airport, at the final state. Other constraints are defined, for instance, the unit1 should visit the node 9 but avoid the edge (9,8). ORTAC+ introduces coordination constraints. If an enemy is suspected on a node or an edge, a unit cannot visit it without support. Therefore, the predicate "*node_supported_from()*" model this coordination. In the presented example, it is specified in order to visit the node 3, there must be a support at the node 18.

5 Conclusion

We have presented ORTAC+, a DSL that hopefully permits a military officer to specify a mission. Compared to its predecessor, ORTAC, the evolution can be summarized as follows:

1. predicates are anonymized when suitable, in particular for the declaration of goals or supports and operate on lists of homogeneous objects (agents, nodes, or edges),
2. these lists can be described intentionally, with a propositional sub-language constraining the attributes of objects,
3. these propositions, when they involve the "or" operator, can be shortened thanks to the declaration of the adequate ontology.

As a result, ORTAC+ allows more compact and declarative models, which will be more easily written and checked by end-users. At the same time, we take care to keep semantics to allow a translation to PDDL, making it possible to rely on the existing planners. Current and future work explores 3 orthogonal directions: new coordination constraints, quantitative time and resources, and compatibility with flying drones.

References

[1] Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Kumar, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the International Symposium on Combinatorial Search*, volume 10, pages 151–158, 2019. doi:10.48550/arXiv.1906.08291.

[2] Alejandro Torreño, Eva Onaindia, Antonín Komenda, and Michal Štolba. Cooperative multi-agent planning: A survey. *ACM Comput. Surv.*, 50(6), 2017. doi:10.1145/3128584.

[3] Patrik Haslum, Nir Lipovetzky, Daniele Magazzeni, Christian Muise, Ronald Brachman, Francesca Rossi, and Peter Stone. *An introduction to the planning domain definition language*, volume 13. Springer, 2019. doi:10.1007/978-3-031-01584-7.

[4] Christophe Guettier, Willy Lamal, Israel Mayk, and Jacques Yelloz. Design and experiment of a collaborative planning service for netcentric international brigade command. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, pages 3967–3974, 2015. doi:10.1609/aaai.v29i2.19055.

[5] G Ayorkor Korsah, Anthony Stentz, and M Bernardine Dias. A comprehensive taxonomy for multi-robot task allocation. *The International Journal of Robotics Research*, 32(12):1495–1512, 2013. doi:10.1177/0278364913496484.

[6] Derek Long, Henry Kautz, Bart Selman, Blai Bonet, Hector Geffner, Jana Koehler, Michael Brenner, Joerg Hoffmann, Frank Rittinger, Corin R. Anderson, Daniel S. Weld, David E. Smith, Maria Fox, and Derek Long. The aips-98 planning competition. *AI Magazine*, 21(2):13, 2000. doi:10.1609/aimag.v21i2.1505.

[7] Michal Stolba, Antonin Komenda, and Daniel Kovacs. Competition of distributed and multiagent planners (codmap). In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 30, 2016. doi:10.1609/aaai.v30i1.9870.

[8] Swaib Dragule, Thorsten Berger, Claudio Menghi, and Patrizio Pelliccione. A survey on the design space of end-user-oriented languages for specifying robotic missions. *Software and Systems Modeling*, 20(4):1123–1158, February 2021. doi:10.1007/s10270-020-00854-x.

[9] Erez Karpas and Daniele Magazzeni. Automated planning for robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 3:417–439, 2020. doi:10.1146/annurev-control-082619-100135.

[10] S. Kloder and S. Hutchinson. Path planning for permutation-invariant multirobot formations. *IEEE Transactions on Robotics*, 22(4):650–665, August 2006. doi:10.1109/tro.2006.878952.

[11] Hang Ma and Sven Koenig. Optimal target assignment and path finding for teams of agents. *arXiv preprint arXiv:1612.05693*, 2016. doi:10.48550/arXiv.1612.05693.

[12] Jingjin Yu and Steven M. LaValle. Multi-agent path planning and network flow. In *Springer Tracts in Advanced Robotics*, pages 157–173. Springer Berlin Heidelberg, 2013. doi:10.1007/978-3-642-36279-8_10.

[13] Bernhard Nebel. On the computational complexity of multi-agent pathfinding on directed graphs. *Proceedings of the International Conference on Automated Planning and Scheduling*, 30:212–216, June 2020. doi:10.1609/icaps.v30i1.6663.

[14] Alexander Kott, Ray Budd, Larry Ground, Lakshmi Rebbapragada, and John Langston. Building a tool for battle planning: Challenges, tradeoffs, and experimental findings. *Applied Intelligence*, 23(3):165–189, December 2005. doi:10.1007/s10489-005-4606-z.

[15] Hang Ma and Sven Koenig. Optimal target assignment and path finding for teams of agents, 2016. doi:10.48550/ARXIV.1612.05693.