Probabilistic Mission Planning and Analysis for Multi-agent Systems

Rong Gu, Eduard Enoiu, Cristina Seceleanu, and Kristina Lundqvist

Mälardalen University, Västerås, Sweden
(first.last)@mdh.se

Abstract. Mission planning is one of the crucial problems in the design of autonomous Multi-Agent Systems (MAS), requiring the agents to calculate collision-free paths and efficiently schedule their tasks. The complexity of this problem greatly increases when the number of agents grows, as well as timing requirements and stochastic behavior of agents are considered. In this paper, we propose a novel method that integrates statistical model checking and reinforcement learning for mission planning within such context. Additionally, in order to synthesise mission plans that are statistically optimal, we employ hybrid automata to model the continuous movement of agents and moving obstacles, and estimate the possible delay of the agents' travelling time when facing unpredictable obstacles. We show the result of synthesising mission plans, analyze bottlenecks of the mission plans, and re-plan when pedestrians suddenly appear, by modelling and verifying a real industrial use case in UPPAAL SMC.

Keywords: MAS, mission planning, Q-learning, statistical model checking

1 Introduction

Multi-Agent Systems (MAS) draw a wide interest in academia and industry, mostly due to their autonomous functions that ease people's daily lives and improve industrial productivity. Mission planning for MAS involves path planning and task scheduling, and is one of the most critical problems when designing such systems [4]. There are path-planning algorithms that have already proved useful for autonomous systems, e.g., RRT [15] and Theta* [5]. These algorithms are able to calculate collision-free paths towards a destination, yet they do not consider complex requirements and uncertainties in the environment. For instance, if agents need to prioritize or repetitively execute some tasks, path planning is not enough. In addition, when the task execution time is uncertain, or some moving objects such as humans and other machines appear irregularly in the environment, autonomous agents need to consider these factors when synthesising mission plans so that the resulting plans are comprehensive. Task scheduling algorithms are designed to solve the above problems. However, since task scheduling is an NP-hard problem, when the number of agents becomes large, traditional

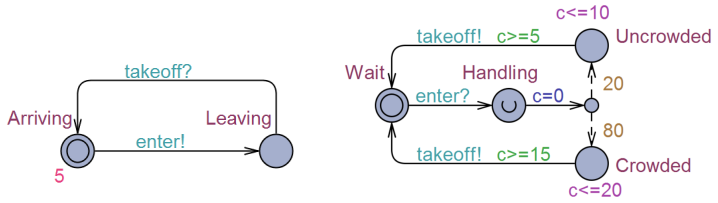
methods cannot manage to produce a result even for a simple instance with very restrictive constraints [1].

In our previous work, we have formally defined and modeled the movement and task execution of MAS [9], and proposed a combined model-checking and reinforcement learning method [10], to synthesise mission plans that are proved to satisfy complex requirements obtained from industry. However, when the agents perform some uncertain actions, e.g., unstable time of moving and operating, or the environment contains some stochastic phenomena, e.g., humans crossing the roads unpredictably, the proposed method does not provide quantitative verification and analysis, which is best suited in these cases.

In this paper, we propose an adjusted version of our method called MCRL (Model Checking + Reinforcement Learning) [10] to provide a means of synthesizing and analyzing mission plans for MAS with uncertainties of the type mentioned above. The method is based on Stochastic Timed Automata (STA) and statistical model checking (by employing UPPAAL SMC), and combines the latter with reinforcement learning. Instead of exhaustively exploring the state space of the model and looking for the execution traces that satisfy certain requirements, MCRL uses the simulation function of UPPAAL SMC to execute the model. Then, it adopts a reinforcement learning algorithm, namely Q-learning [21], to accumulate the rewards of the state-action pairs gathered in the simulation, and populate a Q-table that is used to guide the agents to move safely and finish tasks within a prescribed time limit. As the STA describe the stochastic behavior of the agents and uncertain events in the environment by probability distributions, based on which the simulation is executed, the collected state-action pairs reflect the possible scenarios that the agents would probably meet in the environment. Therefore, as long as the simulation generates enough data, the synthesised mission plans are comprehensive and optimal.

To estimate the possible delays of executing mission plans when the agents encounter unexpected situations, e.g., pedestrians, we adopt a hybrid-automata (HA) model of the agents that are equipped with a state-of-the-art collision-avoidance algorithm based on dipole flow fields [19]. By simulating and statistically verifying the HA model, we can get the estimated travelling time of the agents [11], respectively, which is then used to construct the STA model that is used for synthesising mission plans. Next, statistical verification and simulation of the STA are conducted in UPPAAL SMC in order to analyze the synthesised mission plans in an environment model containing uncertainties, which is not feasible by purely using reinforcement learning algorithms. To summarize, the contributions of this paper are:

- An innovative approach based on MCRL for synthesizing and analyzing mission plans for MAS that exhibit stochastic behavior.
- An effective combination of the STA and HA models of MAS, which enables the estimation of travelling time considering unexpected situations, and thus produces comprehensive mission plans.
- An evaluation of the method showing the ability of analyzing the bottleneck of mission plans and re-planning when facing unpredictable moving obstacles.



(a) A STA modeling passengers (b) A STA modeling an airport

Fig. 1. STA modeling a scenario of passengers arriving at an airport and taking off

The remainder of the paper is organized as follows. In Section 2, we introduce the preliminaries of this paper. Section 3 presents the problem and challenges. In Section 4, we introduce the adjusted version of MCRL and its combination with the HA model. Section 5 presents the bottleneck analysis as well as the ability of re-planning. In Section 6, we compare to related work, before concluding and outlining possible future work in Section 7.

2 Preliminaries

In this section, we introduce Stochastic Timed Automata and UPPAAL SMC, reinforcement learning, and a two-layer framework that we have proposed previously for formal modeling and verification of autonomous agents.

2.1 Stochastic Timed Automata and UPPAAL SMC

UPPAAL SMC [6] is an extension of the tool UPPAAL [14], which supports Statistical Model Checking (SMC) of Stochastic Timed Automata (STA). STA is a widely used paradigm for modeling the probabilistic behavior of real-time systems. The basic elements of STA are locations and edges connecting them. Time can elapse at locations, which is reflected by the increased values of clock variables in delayed transitions of STA, whereas transitions between locations are non-delayed. The delays at locations follow probabilistic distributions, which are either uniform distributions for time-bounded delays, or exponential distributions (with user-defined rates) for unbounded delays. The choices between multiple enabled non-delayed transitions are also probabilistic.

Fig. 1 depicts a network of STA modeling the scenario of passengers arriving at an airport and taking off. Fig. 1(a) shows the model of passengers, who randomly arrive at the airport. The arriving time follows the exponential distribution as it is modeled by an unbounded delay at location *Arriving*. The constant "*5*" is the exponential rate that can be replaced by any rational number. The channels (e.g., *enter* and *takeoff*) model the handshaking interaction between STA. Note that UPPAAL SMC only supports broadcast channels for a clean semantics of purely non-blocking automata. When a passenger enters an airport, the corresponding STA moves to location *Leaving* simultaneously with the airport STA (Fig. 1(b)) moving from location *Wait* to *Handling*, synchronized via the channel *enter*. Next, the airport STA goes to a branch point leading

to two locations, namely *Crowded* and *Uncrowded*, respectively. The constants, "*20*" and "*80*", are the probability weights of the edges marked by the dashed lines in Fig. 1(b), meaning that the probability of entering a crowded airport is *80*%, and *20*% for an uncrowded one. Delays at locations such as location *Crowded* are time-bounded, as the locations are constrained by invariants (e.g., $c <= 10$), so the delay time at these locations should not surpass the upper boundary specified by the invariants, respectively. If the outgoing edges of such locations are guarded by conditions, e.g., $c >= 5$ in our case, the STA cannot leave the locations until the lower boundaries of the guards are exceeded. A uniform distribution is set for the time-bounded delays by default in UPPAAL SMC, which is also adopted in this paper. Variables can be updated by assignments (e.g., $c = 0$) or C-code functions on the edges.

2.2 Reinforcement Learning

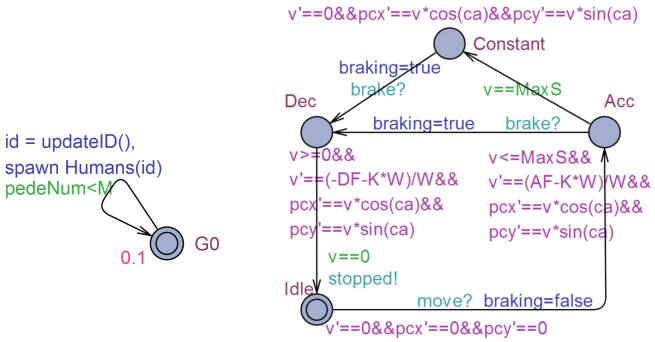
Reinforcement learning is a branch of machine learning that enables agents to learn how to take actions by themselves, in an environment. In this paper, we employ *Q-learning* [21] as the reinforcement learning algorithm to generate policies of movement and task execution for agents. A policy is associated with a state action value function called *Q function*, where "Q" stands for "quality". The optimal Q function satisfies the Bellman optimality equation:

$$q^*(s,a) = \mathbb{E}[R(s,a) + \gamma \max_{a'} q^*(s',a')], \tag{1}$$

where $q^*(s,a)$ represents the expected reward of executing action a at state s, $\mathbb{E}$ denotes the expected value function, $R(s,a)$ is the reward obtained by taking the action a at state s, γ is a constant of discounting, s' is the new state coming from state s by taking action a, $\max_{a'} q^*(s',a')$ represents the maximum reward that can be achieved by any possible next state-action pair (s',a'). The equation means that the expected reward of the state-action pair (s,a) is the sum of the current reward and the discounted maximum future reward. The Bellman equation accumulates the Q-values of state-action pairs and guarantees the values to converge to the maximum Q-value during the learning process [13]. In this paper, we use the simulation function in UPPAAL SMC to gather the information of state-action pairs in files, and invoke a Java program to parse the data and run the Q-learning algorithm, so that a Q-table is populated.

2.3 A Two-Layer Framework for Formal Modelling and Verification of Autonomous Agents

To provide a separation of concerns for the formal modeling and verification of autonomous agents, we have proposed a two-layer framework [11]. In this framework, a static layer is responsible for mission planning and only concerns static obstacles and milestones where the tasks are carried out. The dynamic layer uses hybrid automata (HA) [12] to model the continuous movement and



(a) An example of HA generating pedestrians

(b) An example of HA modeling the linear movement of agents

Fig. 2. Examples of HA model in the dynamic layer of the framework

operations of the agents in UPPAAL SMC. In addition, UPPAAL SMC provides a "spawning" function to dynamically generate instances of HA models during the verification, which enables one to mimic the sudden appearance of obstacles (e.g., pedestrians), which are considered unpredictable before the agents get close to them.

Fig. 2(a) shows the HA that generates pedestrians. As long as the number of pedestrians does not exceed a maximum number (i.e., "pedeNum<M"), the self-loop edge of location *G0* is enabled, which invokes the spawning function to generate an instance of the pedestrian model. The constant "*0.1*" denotes the rate of the exponential probability distribution of the pedestrians' appearance. Fig. 2(b) depicts the HA that models the continuous linear movement of agents. The model contains four locations, representing the four moving statuses of agents: idle, acceleration, constantly moving, and deceleration. At the each of the locations, the derivatives of speed and positions are regulated by Newtonian laws of motion in the form of ordinary differential equations (ODE). In a nutshell, the HA model describes the continuous movement of agents, and thus the simulation of the model reflects the agents' moving trajectories when circumventing obstacles. For brevity, we refer readers to the literature [11] for details. In this paper, we use this HA model to generate the moving trajectories of pedestrians and agents, and UPPAAL SMC to estimate the prolonged traveling time of the agents caused by collision avoidance, which is used for re-planning.

3 Problem Description

In this section, we introduce the research problem that originates from an industrial use case of an autonomous quarry, containing various autonomous vehicles, e.g., trucks, wheel loaders, etc. For example, as shown in Fig. 3, in an autonomous quarry, a wheel loader digs stones at stone piles and loads them into trucks, which carry the stones to a primary crusher, where stones are crushed into fractions, and proceed to carry the crushed stones to the secondary crushers, which is the destination. To accomplish their tasks and guarantee a certain level of pro-

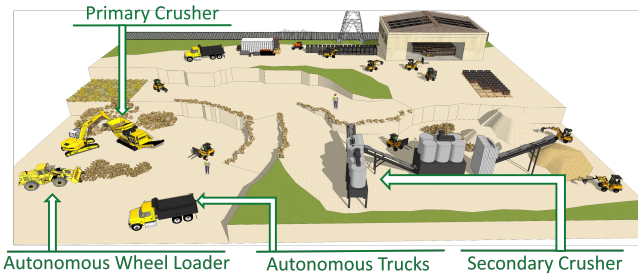


Fig. 3. An example of an autonomous quarry

ductivity, these autonomous vehicles need to calculate collision-free paths and schedule their tasks (e.g., digging stones) to finish their jobs within a time frame. In this paper, henceforth, we name path planning and task scheduling as mission planning in general. As our solution is generic and suits all kinds of autonomous systems that need to synthesise mission plans, the autonomous vehicles in this paper are referred to as autonomous agents [8].

In this paper, path planning is accomplished by the Theta* algorithm [5] as the environment in the problem is a 2D map and the algorithm is especially good at generating smooth paths with any-angle turning points in 2D maps. Task scheduling acquires satisfaction of various requirements, e.g., task assignment, execution order, and timing requirements. We extract the requirements of the autonomous quarry from our industrial partner, and generically categorize them as follows:

- *Task Assignment.* The task must be assigned to the right milestone containing the corresponding device.
- *Execution Order.* The task execution order must be correct, e.g., unloading into the primary crusher can start only after digging stones finishes.
- *Milestone Exclusion.* Some milestones containing a device that only allows one agent to operate at a time are exclusive when they are occupied.
- *Timing.* Tasks must be completed within a prescribed time frame.

The complexity of path planning of multiple agents increases linearly as the number of agents grows, because the path-planning algorithm runs on each individual agent and it does not consider the paths of other agents, as the collision avoidance is dealt with when the agents are actually moving. In other words, the time to calculate paths for multiple agents is the sum of the computation time of each agent. However, the task-scheduling problem is NP-hard and involves uncertainties that traditional methods do not consider [1].

- *Uncertain execution time of tasks.* The execution time of tasks is not a fixed value, but it is a time interval between the best-case execution time (BCET) and worst-case execution time (WCET), which are usually different.
- *Uncertain movement time.* Since some milestones are exclusive, when an agent approaches an occupied milestone, it most probably should wait until it is released. The waiting time is uncertain.
- *Uncertain environment.* Human workers sometimes appear in the sites but do not always stay there. This requires the agents to avoid those workers

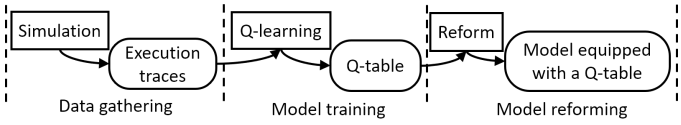


Fig. 4. The process of the MCRL method

at all cost, and adjust their mission plans accordingly, in order to maintain productivity.

These features make our problem even more difficult than the classic scheduling problem. For example, if human workers appear irregularly, it is hard to estimate their influence on the traveling time of agents. We formulate the target problems of this paper as follows.

Overall Challenge. Given a confined environment containing multiple autonomous agents, several predefined milestones and static obstacles, some unpredictable moving objects or humans, a set of tasks for the agents to finish in order to satisfy some requirements, the goal is to synthesize mission plans for these agents, such that:

- The mission plans satisfy the requirements that are categorized previously;
- The mission plans consider the uncertainties in the environment and handle them effectively so that the agents could finish tasks under various conditions;
- The solution provides a means of statistical analysis of the synthesised mission plans to investigate the bottleneck of the plans, and an ability of re-planning when facing disturbance, e.g., pedestrians.

4 Mission Planning Based on Reinforcement Learning and Stochastic Timed Automata

In this section, we introduce the modelling of MAS using STA, which is based on a method called MCRL [10]. MCRL combines model checking and reinforcement learning, which enables the method to cope with large numbers of agents and verify the synthesised mission plans. The use of stochastic timed automata in this paper extends MCRL with the ability of modelling stochastic behaviors. We also present some queries that are used in this method for statistical analysis of the mission plans.

4.1 MCRL: Combining Model Checking and Reinforcement Learning for Mission Planning

Previously, we have presented the formal definitions of agent movement and task execution and the model-generation algorithms to generate Timed Automata (TA) for mission-plan synthesis [9]. This initial work provides a theoretical foundation and a tool called TAMAA, based on which a novel approach is designed to synthesise mission plans, namely MCRL.

Overall Description of MCRL. As Fig. 4 depicts, MCRL consists of three phases. First, it simulates the TA that models the movement and task execution of autonomous agents by running the Monte Carlo simulation query in UPPAAL

SMC. The introduction of the TA model is in the literature [10]. The multi-round simulation produces the execution traces of the model. Some of them satisfy our requirements, e.g., finishing tasks in time, correct execution order of tasks; some traces fail, e.g., exceeding the time limit. The successful traces are assigned with positive values, which are calculated by $(ST - FT)^2$, where ST is the simulation time, FT is time of reaching the desired state, e.g., finishing all tasks; whereas a fixed negative value is assigned to all the failed traces.

Next, the traces and their values are input into the model-training phase, where a reinforcement learning algorithm, namely Q-learning, is performed to generate a Q-table. The Q-table contains the state-action pairs and their values that are accumulated by running Equation (1) using the data of the input traces. This equation guarantees that the values of state-action pairs converge, as long as the simulation has produced enough data of execution traces. Eventually, the Q-table is injected back to the TA model of agents, where a new TA named conductor is created so that the behavior of the agent model is controlled by it. The conductor TA looks up the Q-table and chooses the action that owns the highest value among the available actions at the current state for the agents to perform. Each agent model has its own conductor TA so that the agents can make decisions distributedly. However, as the Q-table contains the state-action pairs of all agents, when their actions conflict, e.g., moving to the same exclusive milestone simultaneously, the agents can compare their rewards of actions with others, and let the one having the highest reward to perform. In this way, the Q-table serves as the mission plan we intend to synthesise. In addition, since the method utilizes random simulation and reinforcement learning instead of pure exhaustive model checking, the solution is scalable for systems with large numbers of agents. For a detailed introduction of the method, we refer readers to the literature [10].

Although Q-learning strengthens MCRL's ability of handling large numbers of agents, the method provides no means of handling unpredictable events, which is important as the environment is uncertain. This limitation stems from the use of timed automata. This modelling language cannot depict the stochastic events in the environment. For example, when human workers sporadically appear in the environment, MCRL cannot estimate the possible delay that is caused by the detour taken by the agents to avoid humans. In addition, industries always focus on productivity. The waiting time of agents at exclusive milestones is an unnecessary consumption of time, but it is hard to capture as the waiting time depends on multiple factors. Original MCRL is not able to provide this kind of analysis, as it does not use any statistical analysing techniques.

4.2 Stochastic Timed Automata for MCRL

To overcome these shortcomings, we improve MCRL by adopting stochastic timed automata (STA) as the modelling language and statistical model checking for verification and analysis. In this section, we present the STA model in detail such that readers understand how the movement and task execution is modelled as STA, and how the stochastic behavior is handled by this model.

Fig. 5. The STA modeling an agent executing task *T1*

STA of Task Execution. Tasks in this paper are operations of the agents that need to be carried out in a right order and at the specific milestones. For instance, in the scenario of an autonomous quarry in Fig. 3, tasks for autonomous trucks can be unloading stones into the primary crushers, charging, etc. Collaborative tasks are the ones that need more than one agent to perform, e.g., loading stones at stones piles needs a wheel loader and a truck to accomplish. For mission planning, a task can be abstracted as time duration between the BCET and WCET, which is only permitted to start when a set of conditions is satisfied, e.g., precedent tasks are finished, and staying at the right milestone. The formal definition of tasks is presented in literature [9].

Fig. 5 depicts an example of the STA modelling an agent executing one of its tasks, namely *T1*. For brevity, the execution of other tasks for the same agent, which should be modelled in the same STA, is not shown in this figure. Note that the variable *id* in this figure is the index of the agent. The STA starts from the location named *Idle* that represents the status of running no tasks. Agents are only allowed to move at this status, hence, this location has a self-loop edge labelled by a synchronization channel *go[id]* that is used to inform the movement STA to start moving. Since the milestone that the agent is approaching to might be occupied and exclusive, the agent probably has to wait. The invariant on the location *Idle* (e.g., *te[id]*<=MT) and the guard on its self-loop edge (e.g., *te[id]*>=MT) is for triggering the "moving" command every MT time units, so that the agent would not wait forever and periodically detects whether the target milestone is available. The detection is done by the STA of agent movement, which is introduced in the next section.

If the agent decides to execute task *T1*, its task execution STA transfers to location *T1*. This edge is guarded by a Boolean expression that is composed of four parts (see Fig. 5). The first Boolean expression *cp[id]*==B checks if the agent is at milestone B currently, where the task is permitted. The following function *isReady(TK1)* returns a Boolean value indicating whether task *T1* is not finished yet. If *T1* is a collaborative task, this function also decides if the collaborating agents are ready for this task by checking if they are staying at the same right position, which is milestone B in this case. The Boolean array named *tasks* stores the execution status of tasks, namely finished or not, so *tasks[TK2]* here checks if the precedent task of *T1* is finished. The Boolean expression *!event[id][0]* indicates that the event monitored by this agent is not active, where the number "*0*" is the index of the event that can be replaced. An event can be a battery-level-low warning, or a critical-damage alert, etc., which needs to be prioritized than regular tasks, and responded within a time

frame. The task execution time is between the BCET and WCET. Therefore, the invariant on location *T1* regulates that the clock variable should not exceed the WCET of *T1*, whereas the guard on the outgoing edge of this location decides the earliest time to leave this location to be later than the BCET. In UPPAAL SMC, the default probability distribution of time-bounded delays is uniform distribution. Hence, the execution time of task *T1* here is between the BCET and WCET with equal possibilities.

When the guards hold, agents can take the transition with the execution of function *start(TK1)* to start *T1*. This function changes the variable of the current task of the agent, and stores the current state of the agent, as well as the corresponding action taken at this moment into an array. The array, which represents the execution trace, will be printed by UPPAAL SMC in the end of the data gathering phase (see Fig. 4). The function *finish(TK1)* simply changes the variable of the current task to *Idle*, and checks if all the tasks have been finished when the agent should leave the environment and stop.

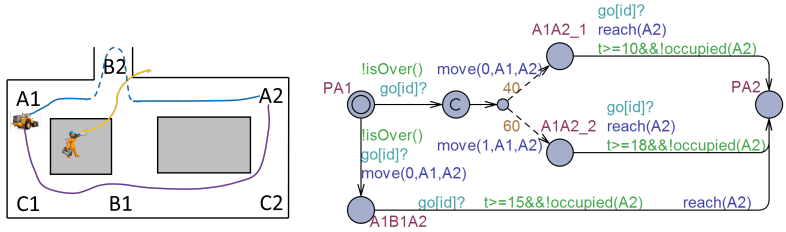
STA of Agent Movement Fig. 6(a) depicts a scenario containing an intersection where pedestrians keep crossing the road every once a while. An autonomous vehicle starting from position *A1* intends to go to *A2*. Though going straightly to *A2* is the shortest path, potential collision avoidance might increase the travelling time, as shown by the blue trajectory. Therefore, the vehicle can alternatively choose to detour via position *B1*, as shown by the violet trajectory. As the HA described in Section 2.3 model the probable appearance of human workers and the continuous movement of agents equipped with a collision-avoidance algorithm based on dipole flow fields [19], we can verify the HA model against queries in the following forms in order to obtain the prolonged travelling time and its probabilities.

```
Pr[<=T](<> arrived)                                    (2)
```

```
Pr[<=T]([] arrived imply t <= TL),                     (3)
```

where `T` is the simulation time, `arrived` is a Boolean variable indicating if the agent arrives at the destination or not, `t` is a clock variable, and `TL` is an integer indicating the time limit. Query (2) calculates the probability of the agent reaching the destination, and Query (3) further calculates the probability of always arriving at the destination within `TL` time units. The results are probability intervals and we use the average value to estimate the probability of travelling time, which is used in the STA of movement.

Fig. 6(b) shows a part of the movement STA modelling the movement from *A1* to *A2*. As there are two alternative paths, the STA starts with a non-deterministic choice between two transitions to location *A1B1A2* or a branch point. The function *isOver()* returns a Boolean value of whether the agent has finished all tasks and should stop. The update function *move(0,A1,A2)* changes the current position of the agent, and stores the current state-action pair into the array, which is similar to the function *start()* in Fig. 5. When the agent chooses to go via position *B1*, which does not have any pedestrians, the STA transfers to the location *A1B1A2* representing the duration of travelling. When the least travelling time has passed, e.g., *15* time units travelling via *B1*, the STA can



(a) A scenario of an intersection containing pedestrians (b) The STA modeling the possible movement of agents

Fig. 6. A scenario of intersection and the STA modeling the movement of agents

transfer to location *PA2*, as long as the milestone *A2* is not occupied. If the travelling time is uncertain by the influence of pedestrians, the STA transfers to a branch point that leads to different locations representing different probable travelling duration, e.g., location `A1A2_1`. After verifying the HA of agents (see Figure 2 for an example) against queries similar to Queries (2) and (3), and replacing *TL* with different numbers, we can obtain that going to position *A2* straightly can cost *10* or *18* time units, and their probabilities are 40% and 60%, respectively, which are depicted in Fig. 6(b). In the STA of movement, a synchronization channel named *go[id]* is used to get commands from the task execution STA (Fig. 5). So the verification of agents is for an integrated model composing the STA of agent movement and task execution.

In UPPAAL SMC, a simulation query composed as following randomly executes the model for `R` rounds and `T` time units in each round,

```
simulate[<=T;R] {ds[0].cs,ds[0].act,ds[0].value,...}:tasks[TK1],    (4)
```

where `ds` is the array variable whose type is a structure, `cs` and `act` are the elements of the structure representing the current state and action, respectively, `value` is the reward or penalty assigned to the pair. The definitions of the states and actions are in the literature [10]. The predicate in the end of the query regulates that the data in the curly brackets are printed only when the predicate is true. In this query, when the agent finishes task *T1*, the elements in `ds` are printed. The simulation needs to run multiple runs for obtaining enough state-action pairs that simulate various situations that the agents would encounter. Hence, the Q-learning algorithm, which uses the state-action pairs as input, would cover various cases comprehensively so that the final mission plans can satisfy various properties in an environment model containing uncertainties.

MCRL Revisited Now that the TA of task execution and movement are adjusted to STA, the simulation query in UPPAAL SMC would explore the state space of the model based on the probability distributions defined in the STA. The model-training phase that uses the state-action pairs representing the stochastic behavior of agents would generate mission plans that are statistically optimal.

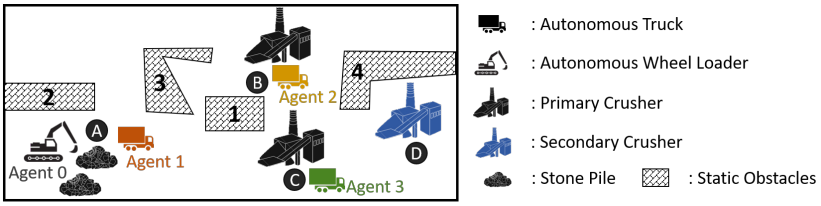


Fig. 7. An experimental scenario containing 4 autonomous agents

Table 1. Tasks for the autonomous agents in the experiment

	Task	BCET	WCET	Precedent task	Milestone
Wheel loader	Dig	2	2	none	Stone pile (A)
	Unload	1	4	Dig	Stone pile (A)
Truck	Load I	1	4	Dig	Stone pile (A)
	Unload I	4	4	Load I	Primary crusher (B or C)
	Load II	2	3	Unload I	Primary crusher (B or C)
	Unload II	3	5	Load II	Secondary crusher (D)

5 Statistical Verification and Analysis of the Use Case: an Autonomous Quarry

In this section, we evaluate our method by demonstrating a statistical verification and analysis on our use case: an autonomous quarry (as shown in Fig. 3). The experiments are conducted in UPPAAL 4.1.24. Most of the statistical parameters are set to the default values in UPPAAL SMC, except the probability of false negatives (α), which is 0.001, and probability uncertainty (ε), which is 0.001. The experimental scenario is depicted in Fig. 7. Tasks for those agents are shown in Table 1. Milestones A to D are exclusive, thus only one truck is allowed at one time. As there are two primary crushers, the trucks need to choose one of them to perform tasks, which take uncertain execution time. The agents must carry all the stones to the secondary crusher, and the job need to be accomplished within a time frame.

5.1 Mission Plan Synthesis

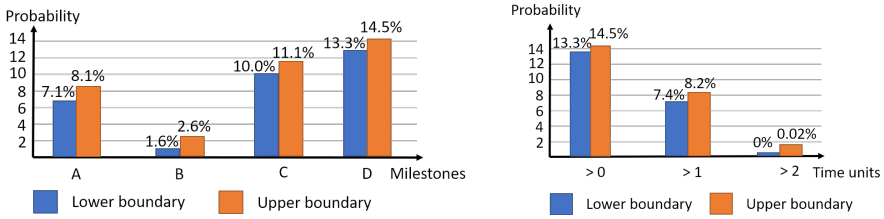
After building the STA and running MCRL by using UPPAAL SMC and our Java program of the Q-learning algorithm, we successfully synthesize mission plans for agents. By verifying queries as following, we demonstrate the synthesized mission plans satisfy different kinds of requirements that are described in Section 3.

– *Task Assignment.* Query (5) checks the probability of agent n performing task T_i at milestone P_i. The results for all tasks in Table 1 are above 99.8%.

$$\texttt{Pr[<=T]([] te}_n\texttt{.T}_i\texttt{ imply m}_n\texttt{.P}_i\texttt{)} \tag{5}$$

– *Execution Order.* Query (6) checks the probability that when agent n is performing task T_i, its precedent task T_j has finished. UPPAAL SMC returns that the results for tasks that have precedent tasks are above 99.8%.

$$\texttt{Pr[<=T]([] te}_n\texttt{.T}_i\texttt{ imply te}_n\texttt{.tasks[j])} \tag{6}$$



(a) Probabilities of waiting at milestones (b) Waiting time at milestone D

Fig. 8. Bottleneck analysis of the scenario in Figure 7

– *Milestone Exclusion.* Query (7) checks the probability that when agent n is at an exclusive milestone named P_i, other agents are not there. The results for milestones A to D are above 99.8%.

$$\texttt{Pr[<=T]([] m}_n\texttt{.P}_i\texttt{ imply !(m}_0\texttt{.P}_i\texttt{ \&\& ... \&\& m}_{n-1}\texttt{.P}_i\texttt{ \&\& m}_{n+1}\texttt{.P}_i\texttt{ ...))} \quad (7)$$

– *Timing.* Query (8) checks the probability of agent n travelling through all milestones and finishing all tasks within TL time units. If we set TL to be *10* and *25* for wheel loaders and trucks, the results are above 99.8%.

$$\texttt{Pr[<=T]([] (te}_n\texttt{.tasks[0] \&\& ... \&\& te}_n\texttt{.tasks[M-1]) imply x < TL)} \quad (8)$$

In these queries, $\texttt{te}_n$ and $\texttt{m}_n$ are the task execution STA and movement STA of agent n, respectively, $\texttt{te}_n\texttt{.tasts}$ is a Boolean array for storing the task execution status of agent n, namely *true* for finished tasks, and *false* for unfinished ones, `M` is the number of tasks, and `x` is a global clock variable that is only reset when all tasks finish.

5.2 Bottleneck Analysis

To perform this analysis, we verify the reformed model equipped with Q-tables against queries in the following form of Query (9) to get the waiting time at different milestones during the process of transferring stones.

$$\texttt{Pr[<=T](<> m0.wt[i] + m1.wt[i] + ... + mn.wt[i] > TL)}, \quad (9)$$

where `T` is the simulation time, `m0` to `mn` are the movement STA of agents *0* to n, `wt[i]` refers to the waiting time at milestone i, and `TL` is an integer estimating the waiting time. By setting `TL` to zero and replacing the index `i` with the indices of milestones A to D, one can investigate the probability of waiting at each milestone (see Fig. 8(a)). By replacing the integer `TL` with different values and fixing the index `i` to some certain milestone, one can estimate the waiting time at the milestone and the corresponding probability (see Fig. 8(b)). In UPPAAL SMC, the result of a probability estimation property (e.g., Query (9)) is given as a probability interval with a confidence level. Hence, the probabilities in Fig. 8 are presented as ranges from the lower boundaries to the upper boundaries. As shown in Fig. 8(a), the probabilities of waiting at milestones A to D are always larger than zero, and the average probability of waiting at milestone D is the highest. We specifically estimate the waiting time at milestone D. As shown in Fig. 8(b), the waiting time is most likely less than 2 time units.

5.3 Travelling Timed Estimation and Re-Planning

When the autonomous agents encounter pedestrians, they must run collision-avoidance algorithms to compute a new path to bypass the pedestrians, and that would possibly affect the travelling time significantly such that it is even quicker to take another path. We call the ability of agents choosing another path when encountering moving obstacles re-planning. In the scenario depicted in Fig. 7, if the number of autonomous trucks is decreased to one, the truck is free to choose between primary crushers at milestones B and C, as no other trucks are competing with it. Since the primary crusher at milestone C is closer to the secondary crusher, the Q-learning algorithm enables the autonomous truck to choose milestone C rather than milestone B as the precedent position of milestone D. We can verify this phenomenon by checking Query (10):

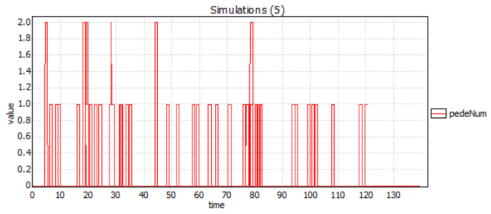
```
Pr[<=T] ([] m0.D imply (viaC && !viaB)),                    (10)
```

where `viaC` and `viaB` are Boolean variables, which are turned to *true* when the agent sets off from the starting point, i.e., milestone A, and reaches milestones C and B, respectively, and are turned back to *false* when the agent leaves milestone D. Hence, Query (10) checks the probability of an agent going to location `D` via location `C` but not location `B`.

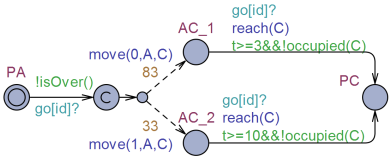
However, if pedestrians keep walking near milestone C and thus block the path (see Fig. 7), it could take longer time if the agent sticks to the original path plan (i.e., travelling via milestone C). By using the HA model depicted in Fig. 2(a), we can generate instances of the pedestrian model dynamically during verification. Then we verify the HA model that describes the continuous movement of agents (see Fig. 2(b) for an example of linear movement) together with the pedestrian model against queries in the form of Queries (2) and (3), in order to estimate the prolonged travelling time between milestones A and C, and the corresponding probabilities. Next, we encode the new travelling time and its probabilities into the movement STA and synthesize mission plans.

Fig. 9 shows two situations of the scenario, where pedestrians are few and crossing the road quickly (Fig. 9(a)), as well as pedestrians are many and walking slowly (Fig. 9(c)), which causes congestion on the road. The situation with fewer pedestrians results in the movement STA that is partly shown in Fig. 9(b), where the probability of going to milestone C quickly is 83% (i.e., $t >= 3$), whereas 33% is the probability of moving slowly (i.e., $t >= 10$). Similarly, the situation containing many pedestrians results in the movement STA partly depicted in Fig. 9(d), where the chance of agents moving slowly is much larger than the chance of moving quickly.

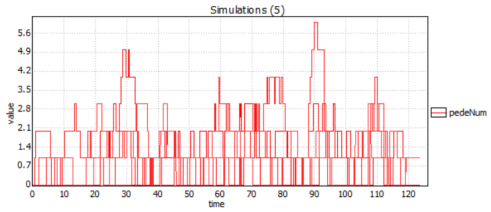
Verifying Query (10) against the model that is partly shown in Fig. 9(d) produces a result of a range of low probabilities, where as if query is changed to check the probability of agents going via milestone B, the result is much higher. This shows that MCRL enables the agents to re-plan a better path when the irregular appearance of pedestrians influences the path plans.



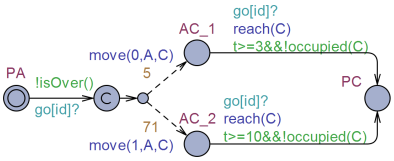
(a) The number of pedestrians. Exponential rate of the generator: 0.1. Existing time: 1



(b) The resulting movement STA in the situation with a few pedestrians



(c) The number of pedestrians. Exponential rate of the generator: 0.2. Existing time: 5



(d) The resulting movement STA in the situation with many pedestrians

Fig. 9. The Number and frequency of pedestrians and the movement STA

6 Related Work

Motion-plan synthesis has arisen a wide interest of research in recent years. Nikou et al. [16] present a method of automatic controller synthesis for multi-agent systems under the presence of uncertainties. Sadraddini et al. [17] propose an approach of synthesising control strategies for positive and monotone systems, which satisfy requirements formalized by Signal Temporal Logic, and demonstrate their method on a traffic management case study. Wang et al. [20] propose a novel formulation based on Partially Observable Markov Decision Processes to synthesis policies over a vast space of probability distributions. Although having promising results, these methods are not applied in industrial systems, which requires solutions to be practically usable and scalable.

To model the uncertain behavior of the autonomous agents and environment, Markov Decision Process and Probabilistic Computation Tree Logic (PCTL) have been adopted by many studies. A solution of behavior verification of autonomous vehicles (AV) proposed by Sekizawa et al. [18] considers the disturbance that causes the AV to swerve from the planned path. Their solution uses the probabilistic model checker PRISM to conduct the verification against PCTL properties. Al-Nuaimi et al. [2] also employs PRISM in their design of a stochastically verifiable decision making framework for AV. The authors demonstrate the applicability of their framework in a scenario of parking bay containing one AV, a pedestrian, and another vehicle. Ayala et al. [3] present a solution to find control strategies for mobile robotic systems moving in environments containing entities that are not completely observable. Compared with these studies, our approach systematically estimates the disturbance caused by unpredictable moving obstacles, and enables re-planning for the autonomous agents. UPPAAL STRATEGO is designed to synthesize strategies for stochastic priced timed games [7],

and it also implements the Q-learning algorithm as one of its algorithms for synthesis. The main difference between MCRL and UPPAAL STRATEGO is that the former supports a larger numbers of agents, and we refer the interested readers to previous work [10] for a detailed comparison between the methods.

7 Conclusion and Future Work

We present a method for automatic synthesis of mission plans for multi-agent systems. The method is based on MCRL, which combines model checking with reinforcement learning, and extends MCRL with the ability of handling uncertainties in the environment by employing Stochastic Timed Automata and Statistical Model Checking. We demonstrate the applicability of the method in an industrial use case: an autonomous quarry, provided by VOLVO CE. The demonstration shows that the method is capable of synthesising mission plans for MAS that satisfy various requirements, and further analyse the bottleneck of the mission plans. When encountering disturbance of unpredictable moving obstacles, e.g., pedestrians, the method is able to estimate the delays of traveling time of the agents, and conduct a re-planning when it is necessary. Future work includes integrating the new MCRL with our tool called TAMAA [9], so that a complete solution of mission-plan synthesis for MAS together with a user-friendly GUI is accomplished. Automating the transformation of requirements into temporal logic queries is another possible direction.

Acknowledgement The research leading to the presented results has been undertaken within the research profile DPAC - Dependable Platform for Autonomous Systems and Control project, funded by the Swedish Knowledge Foundation, grant number: 20150022.

References

1. Abdeddaı, Y., Asarin, E., Maler, O., et al.: Scheduling with timed automata. Theoretical Computer Science **354**(2) (2006), Elsevier
2. Al-Nuaimi, M., Qu, H., Veres, S.M.: A stochastically verifiable decision making framework for autonomous ground vehicles. In: 2018 IEEE International Conference on Intelligence and Safety for Robotics (ISR). pp. 26–33. IEEE (2018)
3. Ayala, A.M., Andersson, S.B., Belta, C.: Temporal logic control in dynamic environments with probabilistic satisfaction guarantees. In: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems. pp. 3108–3113. IEEE (2011)
4. Chandler, P., Pachter, M.: Research issues in autonomous control of tactical uavs. In: Proceedings of the 1998 American Control Conference. ACC (IEEE Cat. No. 98CH36207). IEEE (1998)
5. Daniel, K., Nash, A., Koenig, S., Felner, A.: Theta*: Any-angle path planning on grids. Journal of Artificial Intelligence Research **39**, 533–579 (2010)
6. David, A., Du, D., Larsen, K.G., Legay, A., Mikučionis, M., Poulsen, D.B., Sedwards, S.: Statistical model checking for stochastic hybrid systems. arXiv preprint arXiv:1208.3856 (2012)

7. David, A., Jensen, P.G., Larsen, K.G., Mikučionis, M., Taankvist, J.H.: Uppaal stratego. In: TACAS. Springer (2015)
8. Franklin, S., Graesser, A.: Is it an agent, or just a program?: A taxonomy for autonomous agents. In: International Workshop on Agent Theories, Architectures, and Languages. pp. 21–35. Springer (1996)
9. Gu, R., Enoiu, E.P., Seceleanu, C.: Tamaa: Uppaal-based mission planning for autonomous agents. In: 35th ACM/SIGAPP Symposium On Applied Computing SAC2020. ACM (2019)
10. Gu, R., Enoiu, E.P., Seceleanu, C., Lundqvist, K.: Verifiable and scalable mission-plan synthesis for multiple autonomous agents. In: 25th International Conference on Formal Methods for Industrial Critical Systems. Springer (2020)
11. Gu, R., Marinescu, R., Seceleanu, C., Lundqvist, K.: Towards a two-layer framework for verifying autonomous vehicles. In: NASA Formal Methods Symposium. pp. 186–203. Springer (2019)
12. Henzinger, T.A.: The theory of hybrid automata. In: Verification of digital and hybrid systems, pp. 265–292. Springer (2000)
13. Kochenderfer, M.J.: Decision making under uncertainty: theory and application. MIT press (2015)
14. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. International Journal on Software Tools for Technology Transfer **1**(1-2), 134–152 (1997), Springer
15. LaValle, S.M.: Rapidly-exploring random trees: A new tool for path planning. Tech. rep., Computer Science Dept., Iowa State University (10 1998)
16. Nikou, A., Tumova, J., Dimarogonas, D.V.: Probabilistic plan synthesis for coupled multi-agent systems. IFAC-PapersOnLine **50**(1), 10766–10771 (2017), Elsevier
17. Sadraddini, S., Belta, C.: Formal synthesis of control strategies for positive monotone systems. IEEE Transactions on Automatic Control **64**(2), 480–495 (2018), IEEE
18. Sekizawa, T., Otsuki, F., Ito, K., Okano, K.: Behavior verification of autonomous robot vehicle in consideration of errors and disturbances. In: 2015 IEEE 39th Annual Computer Software and Applications Conference. vol. 3, pp. 550–555. IEEE (2015)
19. Trinh, L.A., Ekström, M., Cürüklü, B.: Toward shared working space of human and robotic agents through dipole flow field for dependable path planning. Frontiers in neurorobotics **12** (2018), Frontiers Media SA
20. Wang, Y., Chaudhuri, S., Kavraki, L.E.: Bounded policy synthesis for pomdps with safe-reachability objectives. In: International Conference on Autonomous Agents and Multi Agent Systems. IFAAMS, ACM (2018)
21. Watkins, C.J.C.H.: Learning from delayed rewards (1989), King's College, Cambridge