GRAVITAS: A model checking based planning and goal reasoning framework for autonomous systems

GRAVITAS: A Model Checking Based Planning and Goal Reasoning Framework for Autonomous Systems

Hadrien Bride[a,*], Jin Song Dong[a,b], Ryan Green[c,d], Zhé Hóu[a], Brendan Mahony[c], Martin Oxenham[c]

[a]*Griffith University, 170 Kessels Rd, Nathan QLD 4111*
[b]*National University of Singapore, 21 Lower Kent Ridge Rd, Singapour 119077*
[c]*Defence Science and Technology Group, Australia*
[d]*University of South Australia, 101 Currie St, Adelaide SA 5001*

Abstract

This work follow the *verification as planning* paradigm and propose to use model-checking techniques to solve planning and goal reasoning problems for autonomous systems with high-degree of assurance. It presents a novel modelling framework – Goal Task Network (GTN) that encompass both goal reasoning and planning under a unified formal description that enables the use of assurance tools. The paper provides a systematic method that highlights how an industrial model checker (PAT) can be used to solve goal reasoning and planning problems modelled by GTNs. Further, this paper also introduces the design of an automated system framework for Goal Reasoning And Verification for Independent Trusted Autonomous Systems (GRAVITAS). The proposed framework is demonstrated in an experiment that simulates a survey mission performed by the REMUS-100 autonomous underwater vehicle.

Keywords: Goal reasoning, Planning, Model-checking, Verification,

*Corresponding author
Email address: h.bride@griffith.edu.au (Hadrien Bride)

1. Introduction

Autonomous systems are complex systems that involve sensors' integration, cognitive ability and physical actuators. This paper considers the high-level cognitive aspect of autonomous systems. More precisely, it focuses on goal reasoning and planning.

Planning is a central and hard Artificial Intelligence problem that is essential in the development of autonomous systems. Many existing solutions require a controlled environment to function correctly and reliably. However, there are situations where adaptive autonomous systems are needed to run for an extended period and cope with uncertain events during the deployment. This work is motivated by the requirements of next-generation autonomous underwater vehicles (AUV) in law enforcement and defence industries. Notably, the authors are currently developing a decision-making system which is suitable for an AUV designed to stay underwater for a long time and to have minimal communication with the outside world. The AUV is expected to carry out survey missions on its own and report details of its surveillance at semi-regular intervals. During the mission, the AUV may encounter underwater currents, deep ocean terrain, fishing boats, objects and places of interest, hostile vehicles etc., each of which may affect its ability to achieve its goals. The AUV must be able to decide which goals to pursue when such dynamic events occur and plan tasks to accomplish the goals in an agile manner.

When there are uncertainties in the environment, planning becomes an even harder problem. In this case, the agent's ability to achieve its goals may be af-

fected; thus, both selecting new goals and re-planning are necessary. There have been various formalisms that attempt to solve planning problems in a dynamic environment, including hierarchical planning methods, such as hierarchical task networks (HTN) [1] and hierarchical goal networks (HGN) [2]. There also exist various goal reasoning systems such as the Metacognitive Integrated Dual-Cycle Architecture (MIDCA) [3] and the goal lifecycle model [4, 5].

Although the above formalisms have been successfully applied to solve real-life problems, the assurance aspects of the problem remains to be addressed. Usually, planning is solved by heuristic search, but heuristics may miss some cases and produce sub-optimal or even undesired results. The correctness, safety, and security issues of autonomous systems are particularly crucial in mission-critical use cases. This work leverages formal methods to tackle the above problem. These methods were initially designed to address the validation and verification of hardware and software systems and have since been used to solve seemingly unrelated problems. For example, Giunchiglia et al. proposed to solve planning problems using model checking [6] and Bensalem et al. [7] used verification and validation (V&V) methods to solve planning.

Building upon the above ideas, this work considers using model-checking to solve not only the planning problem but also the goal reasoning problem for autonomous systems. Model-checking is a technique that can automatically verify whether specific properties are satisfied in a model using exhaustive search. It is especially strong in addressing uncertain and concurrent actions [8]. As such, it is an approach that is well-suited to address the modelling of complex and evolving environments with multiple entities acting concurrently. This formal framing of goal reasoning and planning has many advantages among which the possibility to

formulate inconsistency and incompatibility of plans and goals as reachability and LTL properties [9] that can be verified in action. For instance, when a new goal is generated during execution, the system can reliably check whether the new goal conflicts with existing goals, and select a subset of goals that are compatible with each other. Further, the planning model itself can be verified such that a given planning model does not output plans that may lead to undesired events.

Contributions. In this paper, a novel modelling framework – Goal task networks (GTNs) – which unify and extend previous work on hierarchical task networks and hierarchical goal networks [10, 11] in a formal structure suited for model checking is proposed. Further, a planning and goal reasoning approach based on model-checking and the introduced modelling framework (GTNs) is described. Details on how to practically address complex issues such as planning under resource constraints, goal reasoning in the context of multi-objective missions, and dealing with uncertainty are given. The proposed approach is illustrated over a motivating example which considers the case of a typical AUVs surveying mission. This paper also introduces the design of an automated system framework for Goal Reasoning And Verification for Independent Trusted Autonomous Systems (GRAVITAS) where the proposed planning and goal reasoning via model-checking approach is integrated with other components of autonomous systems. A notable aspect considered in this work is the interaction with untrusted components. Additionally, this paper presents empiric results obtained via simulation that demonstrate the feasibility of the approach in the considered scenario and highlights its benefits and limits.

Outline. The following section presents the related work. Section 3 presents elements that are required for the comprehension of the paper. It notably recalls gen-

eral model-checking notions and presents PAT (i.e. an industrial model-checking tool) and its modelling language. Section 4 gives a motivating example that is used throughout the rest of the paper. Section 5 introduces a novel planning and goal reasoning based on a novel modelling framework (i.e. GTNs) and model-checking. Section 6 presents the design of GRAVITAS, and Section 7 describes an implementation of GRAVITAS in a simulation environment that has been used to produce experimental results discussed in Section 8.

2. Related Work

Different approaches exist according to the assumptions about the domain, the goals, the plans and the planning algorithm. Conceptually, the domain evolves according to the performed actions, and a controller provides the actions according to the observations on the domain and a plan [12]. An example of applying automated reasoning techniques on planning is Kress-Gazit et al.'s framework which automatically translates high-level tasks defined in linear temporal logic formulae to hybrid controllers [13]. This framework allows for reactive tasks, which may change depending on the information the robot gathers at runtime. This is similar to the goal reasoning literature where goals may change depending on the environment at runtime.

This work follows the *planning as model checking* paradigm, which dates back to 1990s, e.g., see work by Giunchiglia and Traverso [6]. They proposed to solve (classical) planning problems model-theoretically, where planning domains are formalised as semantic models, properties of planning domains are formalised as temporal formulae, and planning is done by verifying whether temporal formulae are true in a semantic model. This idea has been studied and improved in

their subsequent work [14, 15, 16], which involves using Binary Decision Diagram based heuristic symbolic search. Similar ideas have been used in planners such as MIPS [17], which can handle the STRIPS subset of the PDDL, and some additional features in ADL.

Closely related to the above work is the verification and validation (V&V) based method of Bensalem et al. [7]. They argue that constructing correct and reliable planning systems is error-prone due to the non-deterministic nature of planning problems. Thus it is essential to develop V&V methods for planners to ensure that the generated plans are correct. To achieve this, the authors proposed to *use* V&V techniques to perform planning . The work presented in this paper is similar in the sense that it uses model checking techniques to perform planning. Since the planning system is built upon the model checker, it can also be used to verify correctness and safety issues of the plans and goals. As a result, the framework presented in this paper not only output plans that are efficient in specific criteria among those that are verified safe and correct, which is essential in building the trusted intelligent agents that are required in mission-critical operations.

Goal reasoning has been used in a number of projects about controlling autonomous machines in a dynamic environment. Many goal reasoning systems follow a *note-assess-guide* procedure, and extend it with a cycle of executions to handle the dynamics of the environment and perform goal reasoning and re-planing on-the-fly. Cox et al. [3] propose to use classical planning to formalise goal reasoning. They present an architecture with a cognitive layer and a metacognitive layer to model problem-solving and dynamic event management in self-regulated autonomy. The architecture is realised in the Metacognitive Integrated Dual-Cycle

Architecture (MIDCA) version 1.3, which is shown useful in experiments. Dannenhauer [18] gives a detailed account.

Roberts et al. [4] give more detailed definitions of goal reasoning in their framework. They divide the states and goals into two parts: the external part is a modified or incomplete version of the transition system, and the internal part represents the predicates and state required for the refinement strategies. The authors use a data structure called *goal memory* to represent the relationship between goals, subgoals, parent goals, etc., and propose to solve the goal reasoning problem using refinement. They use a *goal lifecycle* model to capture the evolution of goals and the decision points involved in the process. The goal lifecycle includes the formulation, selection, expansion, execution, dispatch, evaluation, termination, and discard of goals. This model is adapted by Johnson et al. [5], who give a system called Goal Reasoning with Information Measures. In the scenario of controlling Unmanned Air Vehicles to survey specific areas, the goals are formulated with parameters such as *maximum uncertainty* in the search area, *acceptable uncertainty* under which the goal is considered complete, and *deadline* by which the search must complete. The goal reasoning method is shown useful for unmanned aerial vehicles operating in dynamic environments.

A more theoretical foundation about planning and goal reasoning is surveyed by Alford et al. [11]. The authors unify HGN planning and HTN planning into GTN planning. They also provide plan-preserving translations from GTN problems to HTN semantics. Several computability and tractability results are given. For example, GTN, HTN, and HGN are semi-decidable, and a restricted form called GTN_I is NEXPTIME. An application of HTN planning realised by symbolic model checking is presented by Kuter et al. [19]. While their work is focused

on the theoretical foundation of the problem, and they assume full-observability, this paper is more concerned with a more concrete real-life problem: the execution of the AUV in an uncertain environment. Thus this paper is more focused on practical issues that arise when solving the AUV survey problem.

One compelling use case of goal reasoning is goal selection. Rabideau et al. [20] give a tractable goal selection method algorithm specialised for selecting goals at runtime for re-planning in a system where computational resources are limited, and the complete goal set oversubscribe available resources. Kondrakunta and Cox [21] also consider the situation where an agent has more goals than can complete in a given time constraint and show how an intelligent agent can estimate the trade-off between performance gains and resource costs. Another important aspect of goal reasoning is to detect inconsistency or incompatibility of goals and plans. Tinnemeier et al. [22] propose a mechanism to process incompatible goals which have conflicting plans. They argue that the agent should not pursue goals with conflicting plans, and their mechanism can help the agent choose from incompatible goals.

An important application of our project is applying the planning and goal reasoning framework to AUVs. Among the many relevant papers, goal reasoning for AUVs [23] is particularly interesting. The authors use a goal-driven autonomy conceptual model which has three parts: the planner, the goal controller, and the state transition system. The goal reasoning problem is formalised in PDDL, which is the standard language for representing classical planning problems and is widely used by many planners. The authors test their approach in simulations where the AUV surveys a defined area and has to respond (change the goal) to the actions from a nearby unmanned surface vehicle dynamically. Cashmore et

al.'s work [24] describe a planning algorithm for AUVs. Like many other related papers, their work assumes specific requirements that are slightly different from our settings. For example, they are focused on temporal planning with time constraints, whereas our mission does not have such constraints.

For a broader survey on hierarchical approach to planning see [25].

3. Preliminaries

Model checking [26] is an automated technique for formally verifying finite-state systems. In model checking, specifications of finite-states systems, i.e., properties to be verified, are often expressed in temporal logic whereas the system to be checked is modelled as a state transition graph. Model-checking involves a search procedure which is used to determine whether the model can reach a state that satisfies the specifications. We briefly introduce the modelling language and the specification language in PAT below.

Modelling language.. Models that can be verified using PAT [27] may take several forms, including CSP# models, timed automata, real-time models and probabilistic models. The latter ones are extensions of the CSP# language. In this paper, all the examples only use CSP# – a high-level modelling language that extends Tony Hoare's Communicating Sequential Processes [28] with C#. Formally, a CSP# model is a tuple $\langle Var, init_G, P \rangle$ where Var is a finite set of global variables, $init_G$ is the initial valuation of global variables, and P is a process. Variables are typed: either by a pre-defined type (e.g., boolean, integer, array) or by any user-defined data type. If the type of a variable is not explicitly stated, then, by default, the variable is assumed to be an integer. For instance, an integer variable v and an integer array a can be defined respectively as follows:

```
var v = 0;
var a[3]:{0..5} = [0(3)];
```

The range of a variable can be specified in the definition. For instance, the annotation ‘: $\{0..5\}$’ specifies that the value of each element in a must be in the close interval $[0,5]$. The three values of a are initialised as 0s, as denoted by right-hand side of a’s declaration – i.e., $[0(3)]$ is equivalent to $[0,0,0]$.

A CSP# process is defined using the following syntax:

```
P(x1,x2,...) = Exp;
```

where P is the process name, $x_1, x_2, \ldots$ are the optional parameters of the process, and Exp is a process expression, which defines the computation of the process. The running example in this paper uses the following *subset* of CSP#, shown in Backus–Naur form:

```
Exp ::=  Stop | Skip | Ev{Prog} → Exp | Exp ; Exp | Exp || Exp
       | Exp [] Exp | Exp <> Exp | if (Cond) {Prog1} else {Prog2}
       | [Cond] Exp
```

Interested readers can refer to Sun et al.’s paper [29] for the complete syntax and semantics of CSP#. The process $Stop$ terminate the execution of a process. The process $Skip$ does nothing. Let P and Q be CSP# processes. The process expression $e\{Prog\} \rightarrow P$ first activates the event labelled by e and executes the statements given by $Prog$, then it proceeds with the execution of P. The statements of $Prog$ are defined by the syntax and semantics of C# and can manipulate complex data types. The processes $P;Q$ and $P||Q$ respectively express the sequential and parallel composition of processes P and Q. We use $P[]Q$ to state that either P or Q may execute, depending on which one performs an event first. On the other hand, $P <> Q$ non-deterministically executes either P or Q. The expression $if\ (Cond)\ Prog_1\ else\ Prog_2$ is self-explanatory.

Finally, the expression $[Cond]\ P$, where *Cond* is a boolean expression, defines a guarded process such that P only executes when *Cond* is satisfied.

Specification language.. We can check whether a CSP# process P satisfies a given specification using the following expression:

```
#assert P(x1,x2,...) property;
```

where *property* can be $deadlockfree$ (the process does progress until reaching a terminating state), $divergencefree$ (the process performs internal transitions forever without engaging any useful events), $deterministic$ (the process does not involve non-deterministic choices), and $nonterminating$ (no terminating states can be reached).

Also, we can check whether the transition system can reach a state where a boolean expression *Cond* is satisfied using:

```
#assert P(x1,x2,...) reaches Cond;
```

Additionally, we can check whether a process P satisfies a LTL (cf. Huth et al.'s book [30, Section 3.2.1]) formula F using:

```
#assert P(x1,x2,...) |= F;
```

PAT output.. When checking LTL properties, PAT produces a counter-example when the property to be checked cannot be satisfied, and only outputs "yes" when the property can be satisfied. For reachability properties, which are widely used in the planning technique of this paper, PAT outputs different information. When the desired states cannot be reached, PAT outputs "no". When the desired states can be reached, PAT produces a witness trace of actions that leads to the desired states. When model checking reachability properties, the user can specify one of the following *verification engines*: depth-first search or breadth-first search. If a

breadth-first search based engine is used and the desired states can be reached, then PAT will output the shortest witness trace, which is useful when finding specific "optimal" plans. Furthermore, the user can tell PAT to output the witness trace that optimises a particular criterion. For example, the following code will produce witness traces that respectively yield maximum reward and minimum penalty, assuming that *Cond* is reachable and *reward* and *penalty* are predefined variables:

```
#assert P(x1,x2,...) reaches Cond with max(reward);
#assert P(x1,x2,...) reaches Cond with min(penalty);
```

4. Motivating Example

Surveying marine areas and reporting back the locations of potential objects of interest are essential usages of AUVs. For instance, in the search for the missing aircraft from Malaysia Airlines Flight 370, AUVs were deployed in deep ocean areas to locate the debris of the aircraft [31]. There are also demands and interests from the defence industry to demonstrate the abilities to scan underwater areas for naval mines and dumped arms, as shown in the Wizard of Aus Autonomous Warrior Trial [32].

In this paper we run an example with the following context that demonstrates an ordinary survey mission for the AUV: the AUV is to be deployed at the *initial position* and to be recovered at the *final position*. During the mission, the AUV is expected to scan two *survey areas* and record the locations of *objects of interest* upon identification. Although our technique is general and could be used on all forms of AUVs and UAVs, we specifically target a torpedo-shaped AUV named REMUS-100 [33] equipped with side scanners that can detect surrounding objects. The side scanners have a scan range of about 15 meters, and therefore, in

order to cover a large area, the AUV should perform a *lawn mowing* pattern so that the survey area is fully covered. The overall mission is visualised in Figure 1.

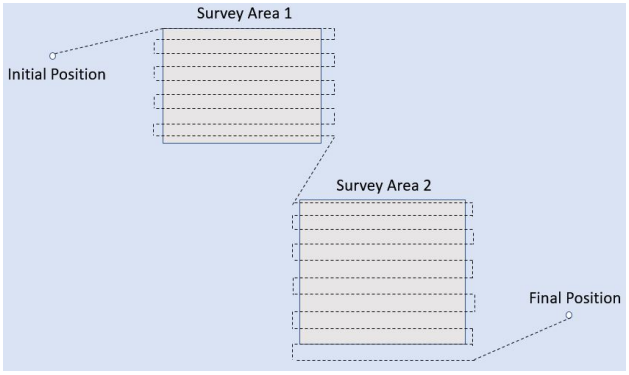


Figure 1: An illustration of the overall survey mission.

Dealing with uncertainties of the environment, such as changes in survey areas, unexpected events during the transit from one location to another, requires an agile enough technique to accommodate the dynamics of the environment. The AUV also needs to make smart decisions autonomously; these include the order in which to visit the survey areas and the entry and exit point of each survey area that maximize trajectory efficiency.

5. Planning and Goal Reasoning via PAT

This section discusses how to solve Goal Task Network planning problems using model checking. We first give a new formalism of the GTN that is suitable for modelling in CSP #. We then propose a model checking based approach to model GTN and solve the planning problem. We also discuss how goal selection – a vital aspect of goal reasoning – can be done in this approach.

5.1. Goal Task Networks

Goal task networks (GTNs) are an extension and unification of hierarchical task networks and hierarchical goal networks [10, 11]. The main conceptual advantage of hierarchical task networks (HTNs), when compared to flat-structured task networks, is their ability to describe dependencies among actions in the form of nested task networks. HTNs have an explicit task hierarchy which generally reflects the hierarchical structure of many real-world planning applications. This hierarchy has decomposition methods which can then be used during the planning phase following the well-known *divide and conquer* scheme. Due to this, HTNs planners are much more scalable and performant than classical planners in practice if the hierarchy is well-designed.

Goal task networks of Alford et al. [11] are similar to hierarchical task networks but also consider goals and sub-goals in addition to tasks and sub-tasks. As a result, they inherit the advantages of HTNs but also provide flexibility and reasoning capabilities in goal reasoning. We give an adaptation of the original GTN below with a focus on guarded state transitions, which are in the same form as processes in CSP#.

Let $\mathscr{V} = \{v_0, \cdots, v_{d-1}\}$ be a finite set of variables. Without loss of generality, the state s of a goal task network over $\mathscr{V}$ is defined as a function $s : \mathscr{V} \to \mathbb{N}$ assigning a non-negative integer to each variable of $\mathscr{V}$. The set of goal task networks $\mathscr{E}$ is recursively defined as $e \in \mathscr{E} \Leftrightarrow e = \langle E_e, \mathrm{g}_e, \tau_e \rangle$ where:

- $E_e \subseteq \mathscr{E}$ is a finite set of sub-tasks/goals,
- $\mathrm{g}_e : (\mathscr{V} \to \mathbb{N}) \to \{\bot, \top\}$ is the guard associated with e, and
- $\tau_e : (\mathscr{V} \to \mathbb{N}) \to (\mathscr{V} \to \mathbb{N})$ is the state transition function associated with e.

Let $e = \langle E_e, \mathrm{g}_e, \tau_e \rangle$ be a goal task network. Then e can conceptually represent a task or a goal. In what follows, we shall loosely refer to e as a task or a goal when the context is clear. When e is a task, its guard models the conditions necessary for the task to begin. When e is a goal, its guard models the conditions under which the goal is achieved.

Goal task networks whose set of sub-tasks/goals is empty are called *primitive tasks/goals* and describe the elementary block of goal task network executions.

The state of a goal task network evolves during its execution according to the following *firing rules*: A task/goal e is *enabled* in state s if and only if $\mathrm{g}_e(s) = \top$. A task/goal enabled in state s can be *fired*, when it does so, it leads to a new state $s' = \tau_e(s)$.

If e is a primitive task/goal then $s \xrightarrow{e} s'$ denote the fact that e is enabled in state s and that its firing leads to state s'. If e is not a primitive task/goal then $s \xrightarrow{e} s'$ denote the fact that there exists a *valid execution* of e starting in state s and leading to state s'.

Given an initial state s_0, a valid execution of e is a sequence $e_0, \cdots, e_{n-1}, e$, of tasks/ goals, where $n \in \mathbb{N}$, such that $\{e_0, \cdots, e_{n-1}\} \subseteq E_e$ and $s_0 \xrightarrow{e_0} \cdots \xrightarrow{e_{n-1}} s_n \xrightarrow{e} s_{n+1}$. The set of all valid execution starting from a given state s is denoted by Σ_s.

A *GTN planning problem* is tuple $P = \langle e, i \rangle$ where e is goal task network and i is the initial state of e. The set of solutions for P is the the set of all valid plans Σ_i, i.e., the set of all valid executions of e starting in state i.

A formalised model of our GTN definitions in Isabelle/HOL is available online [1]. Note that our GTNs can be used to represent the GTNs of Alford et al [11].

[1] `https://figshare.com/articles/GTN_thy/6964394`

Proposition 1. *Given a GTN $(I, \prec, \alpha)$ in Alford et al.'s notation [11] where I is the set of goals and tasks, $\prec$ is a preorder between goals and tasks, and α is a set of labels/names of goal/task instances, there is a corresponding GTN $\langle E, \mathrm{g}, \tau \rangle$ in the above definition.*

5.2. Translating GTN Into CSP#

Let $e = \langle E_e, \mathrm{g}_e, \tau_e \rangle$ and $\{e_0, \cdots, e_{n-1}\} \subseteq E_e$ be GTNs defined over the set of variables $\mathscr{V} = \{v_0, \cdots, v_{d-1}\}$. Further, let i be the initial state of e. The GTN planning problem $P = \langle e, i \rangle$ is modelled as follows.

First, the variables are declared and initialised to their initial values.

```
var v0 = i(v0);···
var v_{d-1} = i(v_{d-1});
```

Second, the GTN e and its sub-GTNs, as well as their sub-GTNs, are recursively defined using to the following template.

```
trans_e() = [g_e] event_e {τ_e} → Skip
sub_e() = e_0() <> ··· <> e_{n-1}() ; (sub_e() <> Skip)
e() = sub_e() ; trans_e()
```

The process $trans_e()$ is guarded by the boolean predicate g_e and, if executed, transforms the current state into its successor state according to the state transition τ_e. Note that τ_e can be effectively encoded by the set of C# statements as C# is Turing complete. Further, the process $sub_e()$ models the set of e's sub-GTNs that can be executed before executing $trans_e()$. In $sub_e()$, we use non-deterministic choices $<>$ to connect the execution of sub-GTNs. The last part of sub_e, i.e., $(sub_e() <> skip)$, allows the process to repeat zero or more times, which effectively chooses and executes any sub-GTN zero or more times. This general translation allows the execution of GTN e to be decomposed to the execution of any subset of sub-GTNs $\{e_0, \cdots, e_{n-1}\}$ for any number of times. Finally, the CSP#

process $e()$ links the processes $sub_e()$ and $trans_e()$ to model the behaviour of the GTN e.

Theorem 1. *For every GTN e, there is a corresponding model $e()$ in CSP# that model all valid plans of e.*

Proof. (Sketch) By the construction of the CSP# process $e()$ and according to the definition of the GTN e, all valid transition sequences of the CSP# process $e()$ correspond to valid plans. □

Example. Take the overall control of the AUV survey mission as an example. At this granularity, the GTN is responsible for making high-level decisions regarding the mission such as: which survey area to visit, in which order, and how to visit it (enter from which direction and exit from which direction). Assuming all the predefined locations are stored in an array, a primitive task at this level is *goto(i)*, which moves the AUV to location i.

```
goto(i) = [visited[i] == 0]go.i {
    currentPosition[0] = position[i][0];
    currentPosition[1] = position[i][1];
    visited[i] = 1;
} → Skip;
```

In the $goto(i)$ task, the vector $visited[]$ records the status of each location. The precondition $visited[i] == 0$ ensures that each location is visited only once. Since $goto(i)$ is a primitive task, it does not contain subtasks/subgoals. Therefore, its formulation only involves the guard condition and the transition.

The compound task $survey(i)$ dictates which locations to visit for survey area i. This task does not have explicit state transitions but instead performs state transitions in its subtasks ($goto()$ tasks). Following the translation template, $survey(i)$ is formulated as:

```
survey(i) = (goto(i_0) <> ··· <> goto(i_n)); (survey(i) <> Skip);
```

where $i_0, \cdots, i_n$ are the indices of the locations in survey area i.

Similarly, the survey mission is formulated as below:

```
mission() = (survey(0) <> ··· <> survey(m)); (mission() <> Skip);
```

where $0, \cdots, m$ are the indices for survey areas.

The overall GTN involves initialising the start position of the AUV, performing the survey mission, and returning to the final position for recovery. This is modelled as below where we omit the code of *initialise*():

```
rendezvous() = goto(finalPosition);
main() = (initialise() <> mission() <> rendezvous()); (main() <>
    Skip);
```

However, since the motivating example specifies that the three sub-GTNs of *main*() should be executed sequentially, the above definition can be optimised as below:

```
main() = initialise(); mission(); rendezvous();
```

5.3. Planning Under Resource Constraints

For most planning applications, considering resource constraints, such as the limited amount of available energy, is critical to the quality and relevance of the produced plan. This is particularly true in the application domain we consider as strategic commanders aim at launching AUVs that are meant to operate autonomously for an extended period with limited resources. Therefore, these resource constraints must be correctly modelled in order to be able to produce plans that can be fully realised, i.e., plans that do not require more resources than available. Also, as unexpected events may arise during the execution of plans, it is necessary to formulate plans that minimise resource consumption in order to maximise the AUV's resilience.

Suppose we wish to consider a finite set of m resources $R = \{r_0, \cdots, r_{m-1}\}$ and certain tasks that may consume or produce a finite and discrete amount of one

or several of these resources. To do so we introduce, to the GTN modelling of a planning problem, a set of m new variables $\mathcal{V}_R = \{v_{r_0}, \cdots, v_{r_{m-1}}\}$ modelling the amount of available resources. In the initial state, the values of these variables correspond to the number of resources available on launch. When a task e consumes one, or several resources, its guard g_e is extended so that it can only be executed if the resources needed to perform it are available before it executes. Additionally, its state transition function τ_e is also extended so that it decreases the values of the resource variables in order to reflect the resources consumed. Similarly, when a task e produces one of several resources, its state transition function τ_e is extended so that it increases the values of the resource variables in order to reflect the resources produced.

Example. In the motivating example, we wish to model the AUV energy consumption while moving based on the distance to travel. To do so, we introduce the variable *energyLevel*, which models the amount of energy left in the battery. We also introduce the function *dist* that returns the distance of the trajectory between two positions and the constant *energyRequiredByMeter*, which is used to scale the energy consumption linearly with respect to a travelled distance. We then modify the $goto(i)$ implementation as follows:

```
goto(i) = [visited[i] == 0 &&
energyLevel >= dist(currentPosition, position[i])] go.i {
    energyConsumed = dist(currentPosition, position[i]) *
    energyRequiredByMeter;
    energyLevel -= energyConsumed;
    currentPosition[0] = position[i][0];
    currentPosition[1] = position[i][1];
    visited[i] = 1;
} → Skip;
```

These changes allow the states of the GTN modelling of a planning problem to encompass available resource quantities and guarantee that valid plans do not,

at any time, consume more resources than available. Furthermore, these changes also enable us to minimise resource consumption by maximising the available resource quantities. However, as several resources may be considered, this leads to a multi-objective optimisation problem that is unfortunately not readily supported by PAT.

We solve this problem by modelling the connections between resources. Note that some resources might be more valuable than others with respect to the mission objectives. Therefore, to avoid the need for multi-objective optimisation capability, we propose to reduce the problem to a single-objective optimisation. To do so, we suggest the use of an extra variable Λ acting as a *common currency* which is used, among other things, to evaluate the overall state of resources. To update the value of Λ we require, for each resource $r \in R$, a *conversion function* $\lambda_r : \mathbb{N} \to \mathbb{N}$ relating the basic unit of a resource as modelled by variable v_r to the basic unit of value of Λ. Conversion functions used in practice include linear functions, logistic functions as well as exponential and logarithmic functions depending on the nature of the resources. Using these conversion functions, we further extend the state transition functions of tasks producing (respectively consuming) resources so that they increase (respectively decrease) the value of Λ accordingly. An important aspect of this approach is that it enables the comparison of any two sets of quantified resources by transitivity. As a result, maximising the value of Λ minimises the overall resources consumption while accounting for the relative importance of the considered resources. Another important aspect of this approach is that it provides mission operatives with an economic perspective on the complex relations that govern the relative importance of available resources – a familiar perspective people can relate to in everyday life.

To illustrate the use of a conversion function we integrate the common currency into the motivating example by inserting the following line after line 3 of the above code modelling the movement of the AUV, where r_{energy} and e_{energy} are user-defined constant:

```
1 Λ -= ⌊r_energy * energyConsumed^e_energy⌋;
```

Continuing with the AUV survey mission example, our model described above already takes the energy cost into account. To find a plan for the modelled GTN with respect to the energy cost, we first need to define the condition for the overall goal:

```
1 #define goal (∀i. visited[i] == 1) && (currentPosition[0] ==
    finalPosition[0] && currentPosition[1] == finalPosition[1]);
```

which states that all the locations are visited, and the AUV's current location is the final position. We then use PAT to find a plan that yields minimal energy cost by model checking the following assertion:

```
1 #assert main() reaches goal with max(Λ);
```

5.4. Goal Reasoning

In this section, we further discuss the concepts that enable our model checking based approach to deal with run-time goal reasoning.

5.4.1. Reasoning About Rewards/Penalties of Goals

Due to environmental constraints and resource constraints, the completion of one or several goals may not be possible, or perhaps not worthwhile. Further, goals may not have the same priority. Some goals may be more critical to the success of the mission than others. Additionally, as one of the underlying directives is to minimise resources consumption, the produced plans may not consider secondary objectives and only fulfil the minimum requirements in order to complete the mission if the incentive to do so is not correctly modelled.

To cope with these challenges, we propose to associate the achievement of a goal with a reward function relating the goal completion to an amount of the basic unit of value of Λ – the previously introduced variable acting as the common currency. In this setting, maximising the value of Λ prioritises and incentivises the completion of goals providing the most rewards while compromising with the resources they require to be completed. Further, as the resources conversion functions and the reward functions can be arbitrarily complex arithmetic functions, this provides a way to assess trade-offs between complex and competing criteria for a large number of resources and goals.

These economic notions, therefore, lead to the formulation of a highly cost-effective plan. Additionally, when multi-agents missions are considered, they provide further benefits as market-based mechanisms [34] can be leveraged to obtain greater collaboration among agents as well as to optimise resources and tasks allocation. These mechanisms also provide non-technical operatives with the means to leverage their day to day economic knowledge to specify technical details of the missions that have to be accomplished by the agent.

Example. Returning to the motivating example, we wish to prioritise the recovery of the vehicle (*rendezvous*()) over the completion of the survey (*mission*()). To achieve this, we first insert the following code into *goto*(i) (between the curly braces):

```
Λ += reward_survey;
```

We then modify the definition of *rendezvous*():

```
rendezvous() = rend{Λ += reward_rendezvous;} → goto(finalPosition);
```

We set $reward_{rendezvous}$ to be far greater than $reward_{survey} \times N$ where N is the total number of positions in the model. We also have to ensure that $reward_{survey}$ is

greater than $\lfloor r_{energy} * energyConsumed^{e_{energy}} \rfloor$, otherwise PAT will choose not to visit any position at all. Finally, we modify the *goal* so that visiting all positions and returning to recovery position is no longer mandatory. Rather, we use a more flexible goal, defined as below:

```
#define goal ∀i∈C. visited[i] == 1;
```

where C is a subset of positions that are critical and will override the optimisation on reward/penalty. Now when we model check

```
#assert main() reaches goal with max(Λ);
```

if the *energyLevel* is sufficient to visit all positions and go to the recovery position, then PAT will output such a plan with minimal energy consumption. Otherwise, suppose the *energyLevel* is insufficient due to unexpected events such as strong current, energy spent on detour or surveying uncertain objects, etc. In that case, PAT will try to find a plan that ensures that the positions in C are visited, and that *rendezvous*() is far more likely to be executed than visiting a few more positions.

5.4.2. Reasoning About Consistency of Goals

Consider the following scenario: the AUV has finished the survey mission and now has to report the results. There are two ways to report: (1) acoustic communication with a nearby friendly surface vessel; (2) surface and use satellite communication. Suppose there is no friendly surface vessel nearby, then the AUV will choose the second method. However, suppose there is a hostile surface vessel, which the AUV should avoid. Now the AUV has two goals: report to the station using satellite communication and avoid the hostile surface vessel. The underlying plans for these two goals have conflicts, and the two goals should not be pursued at the same time.

Since PAT can determine whether a condition is satisfiable or not in execution,

we can also use PAT to determine the satisfiability of the conjunction of several conditions. To solve the above issue, we first formulate the goals as the conditions below:

```
#define goalCompleteSurvey auvCom == 1;
#define successfulSurvey goalCompleteSurvey && hvContact == 0;
```

The first goal says that the AUV has done the communication, the second goal is a compound goal that consists of the first goal and that the AUV does not surface when the hostile vessel is nearby ($hvContact == 0$). We define a task *auvReport* which consists of subtasks *auvAcousticCom* and *auvSurfaceCom*, which represents communication with friendly surface vessel and with a satellite respectively.

```
auvAcousticCom() = [fvInRange]comFV{auvCom = 1;} → auvReport();

auvSurfaceCom() = [!fvInRange]comS{auvDepth = 0; energyLevel =
    10; auvCom = 1; if (hostileInRange) hvContact = 1;} →
    auvReport();

auvReport() = auvAcousticCom() [] auvSurfaceCom();
```

The condition *fvInRange* checks whether the friendly vessel is in range for acoustic communication. Verifying the below assertion, which states that *auvReport* can reach a state where the communication has been done whereas the AUV has not had contact with the hostile vessel, would return negative by PAT.

```
#assert auvReport() reaches successfulSurvey;
```

This means that the above two goals are incompatible, and PAT cannot find an execution path to satisfy both. To resolve this issue, we can add a new task that moves the AUV away from the hostile vessel, as coded below:

```
auvAvoidContact() = case {
        hostileInRange: auvMove();auvReport()
        default: auvReport()
};
auvReport() = auvAcousticCom() [] auvSurfaceCom() []
    auvAvoidContact();
```

Algorithm 1 A simple algorithm for finding minimal unsatisfiable core (MUC). To find a MUC of S, call $Minimise(S, \emptyset)$.

procedure MINIMISE(S, S_0)
  Randomly partition S into two sets S' and S'' of the same size.
  if $S' \wedge S_0$ is unsatisfiable **then**
    return $Minimise(S', S_0)$;
  else if $S'' \wedge S_0$ is unsatisfiable **then**
    return $Minimise(S'', S_0)$;
  else ▷ $S' \wedge S_0$ and $S'' \wedge S_0$ are both satisfiable
    $S'_{min} \leftarrow Minimise(S', S_0 \wedge S'')$;
    $S''_{min} \leftarrow Minimise(S'', S_0 \wedge S'_{min})$;
    return $S'_{min} \wedge S''_{min}$;
  end if
end procedure

Now PAT returns affirmative for the above verification and gives a plan to achieve the goal $successfulSurvey$.

We can extend this solution to check incompatibility of a set of goals. Given a set S of goals, we can use PAT as a black-box and implement Algorithm 1 [35] to find the minimal set of goals that are incompatible. We can also find the set of achievable goals, and update the model to resolve unachievable goals if necessary. Algorithm 1 is an elementary method for efficiently finding the minimal unsatisfiable core of a set of formulae by divide and conquer.

5.5. Performance Testing

To judge the feasibility and scalability of the model-checking based approach, we have tested two levels of planning details. (i) The first level consists of finding an order of the areas to survey so that it minimises the energy cost of the mission. At this level, we abstract away the entry, the internal path and the exit point of each survey area. The second level (ii) enables the entry and exit point of each

survey area to be determined. These levels respectively correspond to two GTNs of increasing complexity. The resolution of this second is higher. As such, a more efficient plan may be formulated. These two levels are chosen to illustrate the scalability of the proposed approach with respect to the size of the model (i.e. the number of survey areas) as well as the levels of planning details.

We ran the testing on the NVIDIA Jetson TX2 – a power-efficient embedded chip that is equipped in a customised REMUS-100 underwater vehicle at Defence Science and Technology (DST) Australia. We report the results in Table 1, in which each configuration is run five times. We note that the variance of results is quite low ($<1\%$) and report the average of the CPU time and memory usage are displayed. One could theoretically also model the *lawn mowing* path inside each survey area, but it is more of an actuation problem than a planning problem; thus, we do not test it here.

Level	# of Survey Areas	avg. CPU Time (s)	avg. Memory Usage (MB)
1	2	0.005	8.4
	3	0.01	8.4
	4	0.03	8.4
	5	0.16	11.7
2	2	0.14	11.2
	3	3.20	66.9
	4	40.26	383.8
	5	290.70	796.3

Table 1: Performance testing for planning and goal reasoning in two levels of PAT models. Level 1 decides which survey areas to visit and the order to visit them. Level 2 further decides the entry and exit points of each survey area.

Discussion. The model complexity has a significant impact on the run-time and memory usage of the goal reasoning and planning phase. This is not surprising and is mainly due to the explosion of the state-space size – an issue commonly en-

countered by model checkers [36]. In short, as the complexity of a computational model increase the size of its underlying state-space grows exponentially. Further, the complexity of the model checking procedures employed is non-trivial. For instance, checking the validity of an LTL property is in PSPACE [37]. This means that heuristic-based approach such as [38] outperform PAT in term of efficiency; however, they do not provide any formal guarantee.

Nonetheless, in our case, the REMUS-100 AUV only has a cruising speed of 5.4 km/h, which means that the software has plenty of time to perform re-planning during the mission. The other targeted hardware, the Ocean Glider, is even slower since it relies on water movement to generate forward thrust. We conclude that Level 1 is feasible, and Level 2 is feasible only when the number of survey areas is less than 3. Note that both these levels are high-level operations. We still need to convert high-level operations to low-level operations which can be actuated by the hardware.

The above results highlight the trade-off between performance and guarantees. An approach based solely on model checking is at the moment intractable, whereas an approach based solely on heuristics do not provide sufficient guarantees about missions critical elements. Therefore, the above empiric results support the design choice of a hybrid approach for goal reasoning and planning. That is, PAT is suitable for making critical high-level decisions, whereas we need to rely on heuristics-based approaches to translate the high-level plans into low-level plans. The verification of this translation is non-trivial: it includes details such as showing that turning the rudder of the AUV at a certain degree corresponds to going a specific direction in the high-level plan. Such details are hardware-dependent and are not in the scope of this paper. Nonetheless, a carefully designed GTN at

an appropriate level of details can, in the context of a hybrid approach, provide better trustworthiness and reliability for the high-level decision-making.

6. GRAVITAS: A Trustworthy Framework for Planning and Goal Reasoning

This section describes the *Goal Reasoning And Verification for Independent Trusted Autonomous Systems* (GRAVITAS) – an automated system which enables autonomous agents to operate with trustworthy high-level plans in a dynamic environment.

6.1. An Overview of GRAVITAS

The GRAVITAS framework follows a cyclic pattern composed of four main phases: Monitor, Interpret, Evaluate and Control, which are illustrated in Figure 2.

The primary operative cycle of GRAVITAS begins with the Monitor (1). This component perceives the environment through the signal processing and fusion of the raw outputs of available sensors. For AUVs, examples of sensors include accelerometers, gyroscopes, pressure sensors and GPS. It is also in charge of processing these data and provide information such as the estimated position and the speed of the agent to the Interpreter (2). This step notably involves techniques such as target tracking which we will not detail here [39]. Once the Interpreter (2) receives the required information, it updates the agent's local model of the system and its environment. This formally defined local model is then forwarded to the Evaluator (3) – a component in charge of assessing the validity of the previously established plan with respect to pre-defined specifications. If the Evaluator assesses the plan as valid, the Controller (5) is tasked with executing the plan. Otherwise, if the Evaluator (3) finds the plan invalid, e.g., an uncertain event creates inconsistencies in the previously established plan and the mission requirements, a

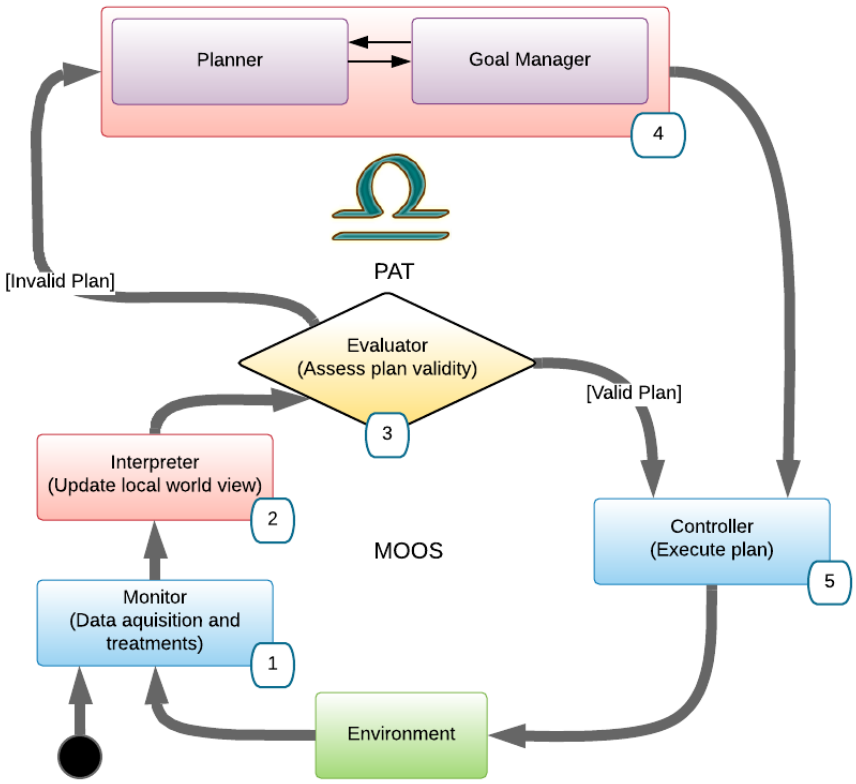


Figure 2: Overall workflow of GRAVITAS.

new plan needs to be formulated. The formulation of a new plan is accomplished by the joint operation of the Planner and Goals Manager components (4). After a new plan is formulated, the Controller (5) is tasked with executing this plan. This step involves processing based on control theory [40], which we do not discuss here.

The components in the lower loop in Figure 2 are orchestrated via the Mission Oriented Operating Suite [41] (MOOS) – a middleware mainly in charge of the communication. The main computational workload of the Evaluator (3), the Plan-

ner and the Goal Manager (4) components are powered by PAT. Note that although conceptually the planner and the goal manager are two separated components, in our implementation, they are realised in the same PAT model, as discussed in the examples throughout Section 5. Also, to achieve high efficiency in real-life applications, we use a hybrid approach discussed in Section 5.5.

6.2. Verification of Planning and Goal Reasoning Models

The key advantage of the model checking based approach is that we can formally verify specific properties for the planning and goal reasoning model. This verification guarantees that the model only permits "correct" high-level plans. Since the verified model is directly used to generate high-level plans in the planning and goal reasoning phase, we can ensure that the generated high-level plans are optimised by for max rewards (resp. min penalties) and are "correct" with respect to the verified properties.

The verification itself is straightforward since the model is already in CSP#. We only need to formulate the properties in the specification language (cf. Section 3) and use model checking to verify them.

Example. In the AUV survey example, we are interested in checking whether the model would permit an execution sequence in which the AUV hits an obstacle. The below Boolean condition expresses that the position of AUV does not overlap with any position of obstacles.

```
#define dontRunIntoObstacle (&& index:{0..iNumberOfObstacles-1}@
    (obstacles[index][0] != auvPosition[0] || obstacles[index][1]
     != auvPosition[1]));
```

Using LTL, we can check whether this condition holds *for all* subsequent states in the execution. This is realised by an assertion of the form

$$p \vdash \Box c$$

where p is a process in CSP#, c is the condition we need to check, and $\Box$ is a modality in LTL that means c holds for all subsequent states. The above verification is realised in the code below

```
#assert main() |= [] dontRunIntoObstacle;
```

and PAT can automatically return "yes" as the result. Thus we obtain the following lemma:

Lemma 1. *The example planning and goal reasoning model described in Section 5 does not generate plans where the AUV runs into any obstacle.*

Using the same technique, we have verified the following lemmas:

Lemma 2. *The example model described in Section 5 does not generate plans where the AUV runs out of battery during the mission.*

Lemma 3. *The example model described in Section 5 does not generate plans where the AUV surfaces at a location within 3 units of distance of a hostile vessel.*

6.3. Interacting with Untrusted Components

Although the Level 2 planning and goal reasoning model in Table 1 suffices in our demonstration of the AUV survey mission, there might be other applications where model checking cannot provide detailed plans in time. For instance, the user may need to adopt heuristic-based planning techniques for UAVs and land vehicles because they run faster.

Inspired by Clarke et al.'s counterexample-guided abstraction refinement [42], we propose to integrate heuristic-based planning techniques as an "un-trusted component" as follows: We treat the heuristic method as a high-level plan generator. Whenever the heuristic method generates a plan, we simulate this plan using the corresponding high-level planning and goal reasoning model, i.e., the *CSP#*

model, in PAT. This simulation is much faster than model checking because we only need to check one path of actions instead of checking all paths. If the simulation is successful, then this plan is in the set of plans that can be generated by the CSP# model. If the CSP# model has been verified as described in Section 6.2, then this plan is correct with respect to the verified properties. If the simulation fails, then we add the old plan into a set of *disabled plans* and constraints the heuristic method such that it does not generate one of the disabled plans. This procedure provides plans that have the same formal guarantee as those generated by PAT, but this procedure may not yield optimal plans. Nonetheless, this procedure provides the means to interact with existing heuristic-based planning techniques generally employed without safe-guards in a reliable way.

7. Implementation of GRAVITAS

In situ experimentation is very expensive and slow. While it is mandatory to the final evaluation of the implementation, in this paper, we focus on assessing and demonstrating the feasibility of the proposed goal reasoning and planning approach in a virtual environment. Notable challenges include controlling the complexity of the GTN so that the embedded hardware of the AUV is able to carry the computational load in a reasonable time (i.e., less than a minute), and the transposition of a discrete plan as issued by PAT into its continuous counterpart so that it can be enacted by the AUV.

We have implemented the proposed approach and integrated it within a virtual environment closely simulating the mission described in Section 4. We first introduce the integration of PAT within a *community* of MOOS applications [41]. We then report the obtained results and discuss the conclusion drawn from them.

The experimental setup is composed of a MOOS application community (referred to as the "community" in the sequel) that corresponds to the AUV internal software environment as well as a set of applications that aim at simulating the external environment. This community includes the following:

MOOSDB: All communication happens via this central server application.

uSimMarine: A 3D vehicle simulator that updates vehicle state, position and trajectory, based on the present actuator values and prior vehicle state.

pMarinePID: A PID controller for heading, speed and depth.

pMarineViewer: A GUI rendering vehicles and associated information during operation or simulation.

pSideScanner: A simulator that reports objects identified by the side scanners of the simulated AUV.

pPATApp: The application that integrate PAT and provide goal reasoning and planning ability to the AUV.

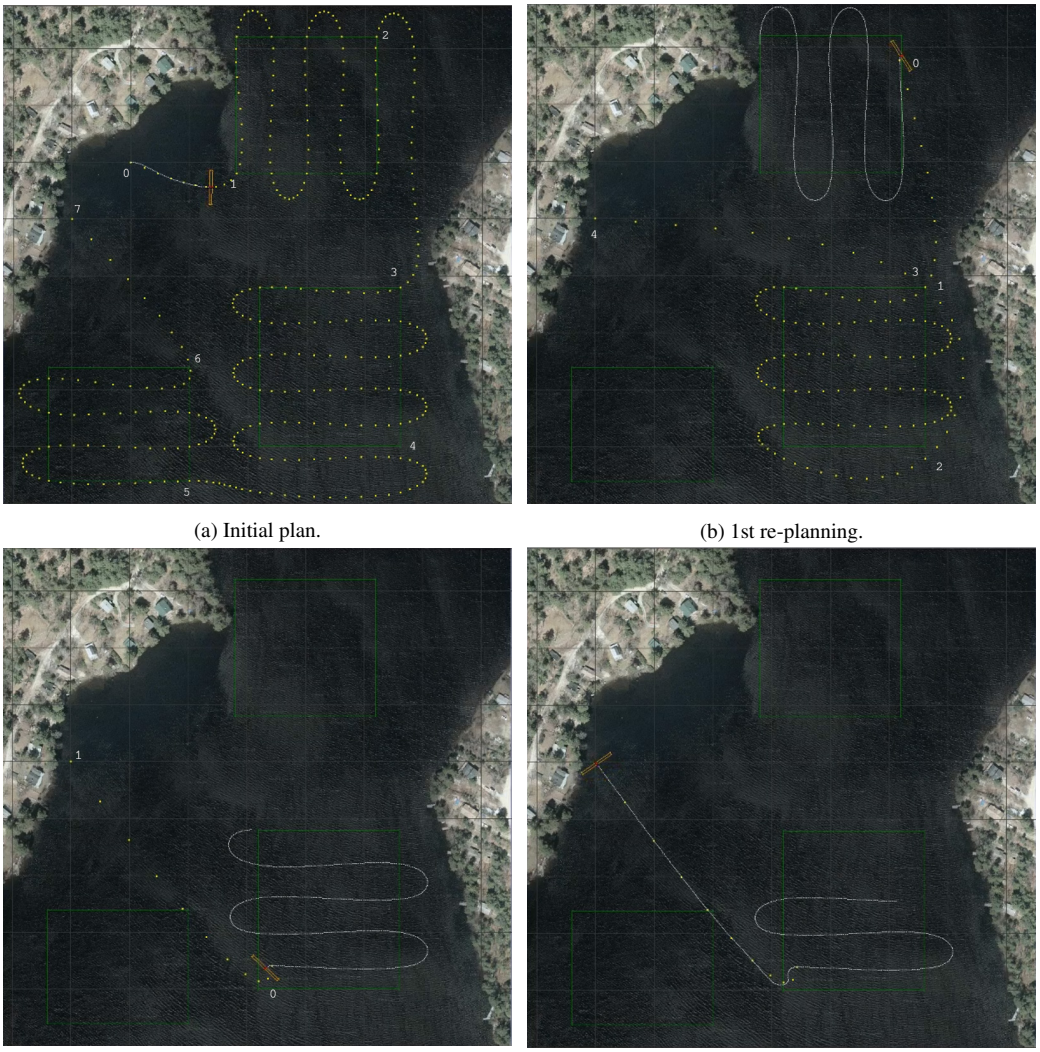
pPATApp implements the GRAVITAS framework as described in Section 6. It subscribes to and monitors channels which broadcast information about the general state of the AUV (e.g., position, speed, heading) as well as information about the objects detected by the side scanners. Then, at each iteration of its internal loop, it interprets this information and models a local world view of the environment. Based on this internal representation and according to the proposed planning approach, it evaluates the actual plan being enacted and, if required, updates it before enacting it by publishing the desired heading, speed and depth of the AUV to the community.

The plan issued by PAT as a part of the re-planning step is a discrete sequence of primitive tasks (e.g. go to 3D position) that require some processing in order to be enacted by the AUV as actuators commands (e.g., set heading, set speed). For instance, the trajectory between several way-points set by the plan has to be compliant with the maximum turn-rate of the AUV. To solve this issue, as a proof of concept, we implemented an algorithm based on piecewise Bezier curves composition with continuous curvature constraint for continuous path planning [43]. In the future we plan on using a more advanced low-level planning approach such as the FMT* algorithm [44] that will enable us to consider trajectories based on 3D current dynamics as well as uncertainty in the AUV position.

8. Simulation in MOOS pMarineViewer

We demonstrate a case study scenario in a simulation in MOOS. In this scenario, we intend to capture GRAVITAS's capabilities in dealing with dynamic events during execution. We create a survey mission similar to Figure 1. Note that although the following screenshots are in 2D, the simulation is actually in a 3D environment.

We set 3 survey areas: lower-left (LL), upper-right (UR), and lower-right (LR), with rewards 22807, 51918, 31313, respectively. Initially, the AUV has an energy level of 60000. These numbers where chosen so that the balance the rewards given for surveying areas are relatively different while being balanced with the AUV energy consumption. Further, during execution, we randomly generate a strong water current *of the opposite direction* with a chance of 20%, that doubles the energy consumption for an uncertain period. Since the AUV under study is mainly driven by water current, when the simulation activates *strong current* in the opposite direction, the AUV runs with the motor and consumes more energy. In



(a) Initial plan.

(b) 1st re-planning.

(c) 2nd re-planning.

(d) Mission finished successfully.

Figure 3: Screenshots of a survey mission simulated in MOOS pMarineViewer.

reality, the chance to have a water current that deviates from the predicted current is rather small and depends on the geographic location. We chose 20% in this experiment because it generates a large amount of uncertainties that justify the need for goal selection and planning as the mission proceed. In this example, goal

reasoning and re-planning is indeed triggered at the end of each survey area.

Figure 3a shows the initial plan computed by GRAVITAS. The numbers indicate the high-level plan computed by PAT and the dots indicate the low-level plan generated by the actuator. That is, PAT finds the optimal order as well as the entry and exit points for the survey areas, and the actuator computes a smooth path that the AUV can follow. The *wing* of the AUV indicates the coverage of the side-scan sonar.

In this run, the random generator creates a strong current of the opposite direction during the first survey. The expected energy consumption for the first survey is 14400, which is the average energy consumption in the same distance under *normal current*, but the actual consumption measured is 23284. Consequently, the Interpreter in GRAVITAS uses simple *linear model* to learn and update the expected energy consumption for future execution (increase the energy consumption rate by 62%). This unexpected change triggers a re-planning in which PAT decides that there is insufficient energy to complete three survey areas, and then finds a new plan to optimise the outcome, as shown in Figure 3b. In the new plan, PAT chooses only to survey area LR because it yields a greater reward.

At the end of the second survey, the Controller in GRAVITAS discovers that survey area LR has been entirely covered before going through the last pass in the *lawn-mowing* pattern. Therefore it triggers re-planning, and PAT and the actuator enact a new plan, shown in Figure 3c, to directly go to the rendezvous point. At this time, the Interpreter captures that the energy consumed to survey the second area is 17411 units, whereas the anticipated energy was 27216 units. As a result, the AUV energy consumption rate is lowered. On the way to the rendezvous point the expected energy consumption is roughly the same as the actual value, and the

AUV successfully finishes the mission, as shown in Figure 3d. This simulation illustrates well the degree of adaptability and trust that GRAVITAS offers. The AUV was able to note that its energy consumption was abnormally high. Although the reason for this discrepancy was unknown, it updated its world view and re-planned its course accordingly. Due to this unforeseen situation choices had to be made and the AUV had to cut-off an area from its survey in order to guarantee its safe journey to the extraction point. This new plan has not been reviewed by any human; however, GRAVITAS verification as a planning approach meant that this plan was formally guaranteed to meet the mission requirements.

9. Conclusion

This paper introduced Goal Task Network – a novel modelling that unifies goal reasoning and planning under a rigorous formal framework suited for analysis. It detailed how such GTNs problem could be translated into a model-checking problem so that advanced verification tools such as PAT can be used to produce plans that are formally guaranteed to meet the mission requirements. It covered how economic notions can be leveraged to address complex modelling aspect such as planning under resource constraints and goal reasoning in the context of multi-objective missions. This approach was illustrated in a typical survey mission. Experimental results pointed out that due to its high complexity, this approach is well-suited for high-level, but is impractical for low-level, planning and goal reasoning. We concluded that a hybrid approach where high-level and high-assurance plans could be obtained via model-checking, whereas traditional heuristic-based planning approaches are suited to translate the obtained high-level plans into low-level and actionable plans. As such, the proposed approach is advantageous when dealing with long-term and global objectives of autonomous systems.

Further, the proposed planning and goal reasoning approach was integrated within a comprehensive framework for Goal Reasoning And Verification for Independent Trusted Autonomous Systems (GRAVITAS). It was shown that model-checking could be coupled with the traditional a *note-assess-guide* procedure, and extend it with a cycle of executions to handle the dynamics of the environment and perform goal reasoning and re-planing on-the-fly. The paper notably addresses how such system may interact with un-trusted components and deal with unknown external effects.

Finally, the paper reports on experimental work were the proposed framework GRAVITAS has been successfully deployed on the hardware chip that runs on the REMUS-100 AUV.

In the future, we plan on formalising the high-level plans to low-level plans translation process in order to fully transfer the high-level assurances provided by the proposed approach down to the physical level at which autonomous systems operate. Further, we also intend to extend GTNs with probability transitions and apply statistical model-checking in order to handle uncertainty in a strait-forward manner. Finally, we plan on leveraging the nested structure of GTNs to scale their verification by using the newly introduced nested model-checking approach [45].

References

[1] K. Erol, J. A. Hendler, D. S. Nau, UMCP: A sound and complete procedure for hierarchical task-network planning., in: AIPS, volume 94, 1994, pp. 249–254.

[2] V. Shivashankar, U. Kuter, D. Nau, R. Alford, A hierarchical goal-based formalism and algorithm for single-agent planning, in: Proceedings of the 11th

International Conference on Autonomous Agents and Multiagent Systems-Volume 2, International Foundation for Autonomous Agents and Multiagent Systems, 2012, pp. 981–988.

[3] M. T. Cox, Z. Alavi, D. Dannenhauer, V. Eyorokon, H. Munoz-Avila, D. Perlis, MIDCA: A metacognitive, integrated dual-cycle architecture for self-regulated autonomy, in: AAAI, 2016, pp. 3712–3718.

[4] M. Roberts, T. Apker, B. Johnston, B. Auslander, B. Wellman, D. W. Aha, Coordinating robot teams for disaster relief, in: Proceedings of the Twenty-Eighth International Florida Artificial Intelligence Research Society Conference, FLAIRS 2015, Hollywood, Florida. May 18-20, 2015., 2015, pp. 366–371.

[5] B. Johnson, M. Roberts, T. Apker, D. W. Aha, Goal reasoning with informative expectations, in: ICAPS Workshop, London, UK, 2016.

[6] F. Giunchiglia, P. Traverso, Planning as model checking, in: European Conference on Planning, Springer, 1999, pp. 1–20.

[7] S. Bensalem, K. Havelund, A. Orlandini, Verification and validation meet planning and scheduling, 2014.

[8] G. Bai, J. Lei, G. Meng, S. S. Venkatraman, P. Saxena, J. Sun, Y. Liu, J. S. Dong, Authscan: Automatic extraction of web authentication protocols from implementations., in: NDSS, 2013.

[9] E. M. Clarke, E. A. Emerson, A. P. Sistla, Automatic verification of finite-state concurrent systems using temporal logic specifications, ACM Trans-

actions on Programming Languages and Systems (TOPLAS) 8 (1986) 244–263.

[10] V. Shivishankar, Hierarchical Goal Network Planning: Formalisms and Algorithms for Planning and Acting, Ph.D. thesis, PhD thesis, Department of Computer Science, University of Maryland College Park, 2015.

[11] R. Alford, V. Shivashankar, M. Roberts, J. Frank, D. W. Aha, Hierarchical planning: Relating task and goal decomposition with task sharing, in: IJCAI, 2016, pp. 3022–3029.

[12] M. Ghallab, D. Nau, P. Traverso, Automated Planning and Acting, 1st ed., Cambridge University Press, New York, NY, USA, 2016.

[13] H. Kress-Gazit, G. E. Fainekos, G. J. Pappas, Temporal-logic-based reactive mission and motion planning, IEEE Transactions on Robotics 25 (2009) 1370–1381.

[14] A. Cimatti, M. Roveri, Conformant planning via symbolic model checking, J. Artif. Intell. Res.(JAIR) 13 (2000) 305–338.

[15] A. Cimatti, E. Giunchiglia, F. Giunchiglia, P. Traverso, Planning via model checking: A decision procedure for ar, in: European Conference on Planning, 1997, pp. 130–142.

[16] P. Bertoli, A. Cimatti, M. Roveri, Heuristic search+ symbolic model checking= efficient conformant planning, in: IJCAI, Citeseer, 2001, pp. 467–472.

[17] S. Edelkamp, M. Helmert, Mips: The model-checking integrated planning system, AI magazine 22 (2001) 67.

[18] D. Dannenhauer, Self monitoring goal driven autonomy agents, Ph.D. thesis, Lehigh University, 2017.

[19] U. Kuter, D. S. Nau, M. Pistore, P. Traverso, A hierarchical task-network planner based on symbolic model checking, in: ICAPS, 2005.

[20] G. Rabideau, S. A. Chien, D. McLaren, Tractable goal selection for embedded systems with oversubscribed resources, JACIC 8 (2011) 151–169.

[21] S. Kondrakunta, M. T. Cox, Autonomous goal selection operations for agent-based architectures, in: IJCAI GRW, Melbourne, Australia, 2017.

[22] N. A. M. Tinnemeier, M. Dastani, J.-J. C. Meyer, Goal Selection Strategies for Rational Agents, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 54–70.

[23] M. A. Wilson, J. McMahon, A. Wolek, D. W. Aha, B. Houston, Toward goal reasoning for autonomous underwater vehicles: Responding to unexpected agents, in: 25th International Joint Conference on Artificial Intelligence (IJCAI) Workshop on Goal Reasoning, New York, NY, 2016.

[24] M. Cashmore, M. Fox, D. Long, D. Magazzeni, B. Ridder, Opportunistic planning in autonomous underwater missions, IEEE Transactions on Automation Science and Engineering 15 (2018) 519–530. doi:10.1109/TASE.2016.2636662.

[25] P. Bercher, R. Alford, D. Höller, A survey on hierarchical planning-one abstract idea, many concrete realizations., in: IJCAI, 2019, pp. 6267–6275.

[26] E. M. Clarke, Jr., O. Grumberg, D. A. Peled, Model Checking, MIT Press, Cambridge, MA, USA, 1999.

[27] J. Sun, Y. Liu, J. S. Dong, J. Pang, PAT: Towards flexible verification under fairness, volume 5643 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 709–714.

[28] C. A. R. Hoare, Communicating sequential processes, Communications of the ACM 21 (1978) 666–677.

[29] J. Sun, Y. Liu, J. S. Dong, Model checking csp revisited: Introducing a process analysis toolkit, in: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods, Verification and Validation, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 307–322.

[30] M. Huth, M. Ryan, Logic in Computer Science: Modelling and Reasoning About Systems, Cambridge University Press, New York, NY, USA, 2004.

[31] news.com.au, New ocean infinity search for mh370 encountering big problems, http://www.news.com.au/travel/travel-updates/incidents/new-ocean-infinity-search-for-mh370-encountering-big-problems/news-story/89efc71d393585bb3efc1fcd10b765aa, 2018.

[32] D. S. Institute, Wizard of aus 2018 an autonomous warrior 2018 trial, http://www.defencescienceinstitute.com/2017/11/06/wizard-aus-2018-autonomous-warrior-2018-trial/, 2018.

[33] W. H. O. Institution, Remus 100, http://www.whoi.edu/main/remus100, 2018.

[34] S. H. Clearwater, Market-based control: A paradigm for distributed resource allocation, World Scientific, 1996.

[35] I. Dillig, Lecture notes on automated logical reasoning, https://www.slideshare.net/pvcpvc9/lecture17-31382688, Accessed 2018.

[36] A. Valmari, The state explosion problem, in: Advanced Course on Petri Nets, Springer, 1996, pp. 429–528.

[37] P. Schnoebelen, The complexity of temporal logic model checking., Advances in modal logic 4 (2002) 35.

[38] D. Nau, Y. Cao, A. Lotem, H. Munoz-Avila, Shop: Simple hierarchical ordered planner, in: Proceedings of the 16th international joint conference on Artificial intelligence-Volume 2, 1999, pp. 968–973.

[39] S. Challa, M. R. Morelande, D. Mušicki, R. J. Evans, Fundamentals of Object Tracking, Cambridge University Press, 2011.

[40] E. B. Lee, L. Markus, Foundations of optimal control theory, Technical Report, Minnesota University Minneapolis Center for Control Sciences, 1967.

[41] P. M. Newman, MOOS-mission orientated operating suite (2008).

[42] E. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith, Counterexample-guided abstraction refinement, in: E. A. Emerson, A. P. Sistla (Eds.), Computer Aided Verification, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 154–169.

[43] J.-W. Choi, R. Curry, G. Elkaim, Piecewise bezier curves path planning with continuous curvature constraint for autonomous driving, in: Machine learning and systems engineering, Springer, 2010, pp. 31–45.

[44] L. Janson, E. Schmerling, A. Clark, M. Pavone, Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions, The International journal of robotics research 34 (2015) 883–921.

[45] H. Bride, C. Cai, J. S. Dong, R. Goré, Z. Hóu, B. P. Mahony, J. McCarthy, N-PAT: A nested model-checker - (system description), in: Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Proceedings, volume 12167, Springer, 2020, pp. 369–377.