Online Plan Modification in Uncertain Resource-Constrained Environments

Catherine A. Harris[a], Nick Hawes[b], Richard Dearden[c]

[a] *National Oceanography Centre, European Way, Southampton, SO14 3ZH*
[b] *Oxford Robotics Institute, University of Oxford,17 Parks Road, Oxford, OX1 3PJ*
[c] *Cambridge, UK*

Abstract

This paper presents an approach to planning under uncertainty in resource-constrained environments. We describe our novel method for online plan modification and execution monitoring, which augments an existing plan with precomputed plan fragments in response to observed resource availability. Our plan merging algorithm uses causal structure to interleave actions, creating solutions online using observations of the true state without introducing significant computational cost. Our system monitors resource availability, reasoning about the probability of successfully completing the goals. We show that when the probability of completing a plan decreases, by removing low-priority goals our system reduces the risk of plan failure, increasing mission success rate. Conversely, when resource availability allows, by including additional goals our system increases reward without adversely affecting success rate.

We evaluate our approach using the example domain of long-range autonomous underwater vehicle (AUV) missions, in which a vehicle spends months at sea with little or no opportunity for intervention. We compare the performance to a state-of-the-art oversubscription planner. Planning within such domains is challenging because significant resource usage uncertainty means it is computationally infeasible to calculate the optimal strategy in advance. We also evaluate the applicability of our plan merging algorithm to existing IPC domains, presenting a discussion of the domain characteristics which favour the use of our approach.

Keywords:
automated planning, decision making, resource uncertainty, contingency planning, plan repair, online planning

Email addresses: catherine.harris@noc.ac.uk (Catherine A. Harris), nick@robots.ox.ac.uk (Nick Hawes)

1. Introduction

Many logistics and robotics domains, including long-duration autonomous underwater vehicle (AUV) and rover missions, can be thought of as *over-subscribed planning problems*, where finite resources, such as battery power and data storage space, limit the number and duration of tasks achievable by the agent. In over-subscribed planning problems, the objective is therefore to select the set of tasks, or goals, which maximises reward given limited resource availability. In many robotics domains, the exact amount of resources required to complete each task is uncertain and not known in advance. This is due to the inherent uncertainty present during operations in the stochastic world, and the measurement uncertainty associated with the resources themselves, such as charge across multiple batteries [1]. We focus on robotics domains where a trade-off exists between the probability of successfully completing a mission and the desire to maximise an optimisation metric over the mission, e.g. expected reward. For instance, a robot may collect large quantities of valuable data but in doing so exhausts its resources, requiring human intervention. In hazardous environments, such as underwater, in space or within a nuclear reactor, human intervention may be extremely difficult or completely impossible, requiring the robot to be fully autonomous. This paper presents a novel approach to solving such problems. We evaluate the solution using an AUV domain, as AUV missions in the open ocean are subject to significant levels of resource uncertainty as a result of operating in a highly dynamic environment, e.g. battery consumption and thus vehicle endurance are greatly affected by environmental properties such as temperature, currents and biofouling (increasing drag), as well as faults in the vehicle itself. Prior knowledge of the operating environment is often limited to ship-based measurements and forecasts, with physical properties of the ocean and weather having a significant effect on AUV mission performance. This significant uncertainty, coupled with the need to maximise the data-gathering activities of the AUV due to the significant expense of and limited opportunity for deployments, results in an oversubscribed goal set. As a result, we feel the AUV domain is an exemplar of domains in which a trade-off exists between the probability of successfully completing a mission and the desire to maximise an optimisation metric over the mission.

Our approach enables an agent to autonomously refine its mission plan during execution, adding goals if observed resource levels are sufficiently high, whilst removing low-priority goals if resource usage is higher than expected. We employ an online plan modification and execution monitoring approach which seeks to utilise the available resources to maximise the reward achieved during the mission without compromising the safety of the agent. As the benefits of planning are often hard to quantify in advance [2], and any resources used by the agent for planning are no longer available for performing tasks, rather than generating a new full plan during execution, we have developed an online plan modification algorithm which sequentially modifies an existing plan using plan fragments generated prior to execution. These pre-computed fragments represent the actions required to complete each goal individually and their associated causal struc-

ture. We use this knowledge of the causal relationships between actions to either interleave additional actions into the plan if resources are plentiful (using a novel plan merging algorithm), or to remove actions if the probability of successfully completing the mission falls below a user-defined threshold of acceptable risk. By sequentially modifying the existing plan rather than replanning from scratch, the plan executed by the agent resembles the initial plan agreed prior to the start of the mission. Predictable behaviour is especially important in real-world autonomous robotics domains, where risks to the robot itself, humans and the environment are approved in advance by stakeholders and frequently mitigated through supervision by a human operator where possible. Investigating why the AUV community has yet to widely adopt adaptive mission planning, Brito *et al.* [3] found unpredictable vehicle behaviours to be the largest concern (39.7%) of expert AUV operators, which our approach addresses.

In this paper, we define the key characteristics of realistic domains that motivated the development of our approach (Section 3), define an illustrative AUV domain (Section 2) and outline the position of our work in relation to the existing literature (Section 4). In Sections 5-10, we present a detailed description of our online plan modification and execution monitoring approach. In Section 11, we present an empirical evaluation of our approach using our illustrative AUV domain, showing that when resources are limited, our approach prioritises the safety of the vehicle, increasing mission success rate by removing low-value goals. Conversely, the results show that our approach allows surplus resources to be exploited to increase mission reward, without jeopardising the success of the mission. Whilst our approach makes greedy sequential updates to the plan during execution, we also show that the quality of plans produced approximates that of a state-of-the-art over-subscription planner, OPTIC [4], achieving an average 93.2% of the latter's reward per unit battery. In Section 12, we illustrate the wide applicability of our novel plan merging algorithm by presenting and discussing its performance on two International Planning Competition (IPC) domains.

2. AUV domain representation

AUVs are becoming increasingly popular for a wide variety of applications, ranging from geological surveys and environmental monitoring to pipeline and hull inspection. Due to their ability to operate independently from a support ship and crew, AUVs provide unparalleled opportunities for conducting underwater missions. However, despite being fully autonomous, most ocean-going AUVs currently perform very little onboard reasoning. Despite notable work by McGann *et al* [5] and Pinto *et al* [6], who present decision-making architectures for AUVs both based on the NASA EUROPA planner, many operators continue to favour simple pre-scripted behaviours such as lawnmower surveys, which minimise the complexity of the mission and risk to the vehicle [7, 8]. By analysing data from the vehicle between missions, AUV operators can then select areas of interest for investigation during subsequent deployments [9]. These

conventional mission formats are inevitably over-conservative, designed to minimise the risk to the vehicle and its data cargo whilst ensuring the behaviour of the vehicle is always known to the operator, reserving a significant proportion of battery as a contingency should usage be higher than expected. Consequently, in the average case, the vehicle is not used to its full potential.

Our AUV problem scenario envisages a vehicle with limited battery power and on-board memory conducting a long-duration mission to collect scientific data from a network of specific pre-defined survey areas. The location of these survey areas along with an estimate of the value of each dataset are assumed to be provided in advance by a scientist. The relative value of each dataset should reflect any preference the scientist may have for collecting one dataset over another. Data-collection behaviours, such as mapping the extent of a chemical plume, can be performed at each survey area. The vehicle is able to travel between survey areas using a pre-defined network of waypoints and may surface at any point during the mission. Whilst on the surface, the vehicle may attempt to transmit datasets back to base via a satellite link. We assume datasets are only of value to scientists if they are successfully downloaded from the vehicle, either by being present on the vehicle at the point of recovery or by being transmitted by the vehicle mid-mission. Without this assumption, the potential value of the data and the cost of losing it (such as through the corruption of onboard storage or the total loss of the vehicle) would not be represented within the problem.

During the execution of each action we have assumed there is a small chance of mission failure. This failure rate represents the inherent risks to both the success of the mission and the safety of the vehicle itself. We assume that the vehicle is operating in open ocean (e.g. not under ice) and thus always has the option to surface, end the mission and await recovery. The vehicle receives a reward for finishing the mission without exhausting the battery. However, if the location of the vehicle does not equal the predefined end location, the vehicle does not receive the full reward as recovering the vehicle from a different location is costly as it requires the support vessel to change route. By heavily penalising plans which do not finish at the agreed end location, such plans should only be generated when strictly necessary to safeguard the vehicle and the success of the mission.

The optimisation metric used in our AUV domain is a function of expected plan value, representing trade-offs between reward and the probability of completing the plan successfully, and is further defined in Section 7.2.

2.1. Actions

We use the following seven parameterised *action schemas* to concisely represent the full action space:

- *CollectData*(d) — Collect dataset d if the vehicle is in the correct location.
- *Move*(l_1, l_2) — If the vehicle is at location l_1, move to l_2.
- *Dive()* — Move from the surface to depth at the current location.

- *Surface()* — Move from depth to the surface at the current location.
- *TransmitData*(d) — Attempt data transmission provided the vehicle is at the surface and has dataset d.
- *DeliverData*(d) — Deliver dataset d provided the mission has ended and the vehicle has collected the dataset d.
- *EndMission(l)* — Wait for recovery at current location l.

Continuous state variables are used to represent the amount of resources currently available to the vehicle. We assume battery power to be a monotonically decreasing variable which cannot be recharged during a mission and assume the battery usage probability distributions for each action to be independent. Therefore, following the nomenclature defined by Smith and Becker [10], battery is considered a *consumable* resource. All actions reduce the battery by a quantity modelled by a Gaussian distribution, individual to each action. Additionally, *CollectData* similarly reduces the amount of available memory, while *TransmitData* increases it by the observed size of the transmitted dataset. We assume differences in expected memory usage between datasets (i.e. as a result of sensor type or resolution) are implicitly represented by their associated means and standard deviations. We assume that partial datasets are worth nothing and so memory must not be exhausted during data-collection. Unlike battery power, memory is thus a constrained *reusable* resource (following nomenclature of Smith and Becker [10]), which becomes available for re-use following successful data transmission. All action preconditions require current available resources to exceed the mean resource requirement plus one standard deviation. Positive rewards are given when the vehicle both successfully delivers a dataset and ends the mission, with additional reward available for ending the mission at the agreed location.

The reward function favours plans which prioritise the collection of high value datasets without compromising the overall safety of the vehicle. This means that during a mission, should the vehicle find it has used significantly more battery power than expected to complete a series of actions, the option to travel to the recovery location and end the mission becomes more attractive than continuing to collect additional datasets.

A formalised representation of our AUV domain as an Markov decision process (MDP) [11], including a description of all state variables, can be found in Appendix A.

3. Motivating domain characteristics

Whilst this paper evaluates our approach in the context of AUV missions, this is just one of a larger set of realistic planning domains, including many robotics applications, which our approach may be suited to. In this section, we discuss the key characteristics of realistic planning domains that suit the use of our presented approach.

3.1. Uncertainty

When the outcomes of all actions in a domain are certain and known in advance, there is no benefit to be had from changing the plan at run-time. In such a situation, a fixed straight-line plan, consisting of a single sequence of actions with no branches, will execute exactly as the planner assumed during plan generation and so there are no alternative outcomes to consider. The approach we present is designed for domains where the outcomes of actions are *continuous* and *uncertain*, which we represent as probability distributions. Our system supports the use of any continuous probability distribution, provided a suitable discretisation may be found for use during plan generation (e.g. in our example AUV domain, during plan generation we assume action battery usage is equal to the mean of each Gaussian distribution). This allows a distribution to be selected that best represents the discharge characteristics of a specific battery, for instance. Our AUV domain considers a single reusable resource (memory) and a single monotonically decreasing consumable resource (battery), but the approach supports any number of reusable/consumable resources.

The presence of continuous resource uncertainty in a domain prevents or highly complicates the use of many existing planning approaches and techniques:

- Fixed plans may fail if states encountered during execution do not sufficiently reflect the assumptions made during plan generation [12].
- Many MDP-based methods require coarse discretisation of the transition function and are limited to those problems with a sufficiently small state space [13].
- Contingency planners require either a finite number of possible contingencies or suitable outcome discretisation.
- Replanning, such as that performed by FF-Replan [14], is likely to result in the generation of an entirely new plan after the execution of each action, as continuous outcomes prevent the use of all-outcome determinisation.

While these alternative approaches have limitations when applied to domains featuring uncertainty over continuous state variables, we would expect contingency planning and MDP-based approaches to out-perform our approach on domains which contain only discrete uncertainty, i.e. those with a finite number of possible outcomes (for example, a cycle action may move a cyclist between two locations with 95% probability but result in a puncture in the remaining 5% of cases). This is because when the number of possible outcomes is small and finite, it is computationally feasible to calculate the optimal course of action (e.g. a policy) to take in the event of each possible outcome in advance. Our AUV domain is a stochastic shortest path problem with unavoidable dead ends (e.g. exhausting battery). Whilst recent work has delivered efficient solutions for such problems with discrete states [15], good performance in our problem involves modelling the continuous dynamics of the vehicle's resources. In order to plan in this space, the state-of-the-art in partially observable MDPs would

suggest sampling-based methods [16] [17] are likely to be effective. However, our problem contains a large amount of uncertainty, therefore would require a very large number of samples to adequately characterise the problem, and would not provide performance guarantees for a finite computation budget. In contrast, our online plan modification and execution monitoring approach is approximate and thus, while it seeks a valid satisficing solution (i.e. one which meets all constraints) which maximises a specified metric, it does not perform a full search for an optimal plan/policy.

3.2. Over-subscribed problems

The extent of relationships and dependencies between goals in an over-subscribed problem is also an important factor to consider when determining how effective our approach is likely to be when applied to a new domain. Our approach is designed for domains where the causal structure is such that the goal literals are relatively independent — while resource availability may prevent the completion of a goal, there should be little to no goals whose completion is conditional upon the completion of any other. Consequently, all goals in the problem may be considered individually and removed in any number and combination, resulting in a wide range of potential modifications with their associated costs and benefits. If a wide range of alternative goal sets (and thus plans) does not exist for a given problem, this limited choice may force the system to be overly reactive when removing goals. For example, if a problem only has one goal which may be removed independently of all others, if the likelihood of successfully completing the plan drops slightly below the accepted threshold, the goal will have to be removed, however valuable it may be. Conversely, if a problem has many independent goals, more subtle changes to the goal set and thus the optimisation metric may be made.

As a counter example, in domains such as Blocksworld[1], which are not over-subscribed, the goal literals each represent components of a single goal state comprising an arrangement of blocks and thus are highly dependent on each other. While it is possible for our plan merging algorithm to successfully add goals to Blocksworld problems under very specific circumstances, such as adding a new block to the top of a stack, the goal characteristics of such domains mean that the use of both our merge algorithm and our wider online plan modification and execution monitoring approach is limited.

In summary, the best performance may be gained from our online plan modification and execution monitoring system when used with a large over-subscribed goal set, comprised of many independent goals.

3.3. Optimisation metrics

In over-subscribed domains, where the available resources are insufficient to complete all goals, a choice of goal set must be made. In many real-world situ-

[1]The Blocksworld planning domain is very well-known, having been used extensively over the years as a test domain. It was widely popularised by Sussman in his 1973 paper [18].

ations, this choice is informed by a measure of solution quality or metric which reflects characteristics of an optimal solution, such as minimising cost/time/-plan length or maximising reward. For example, in their 2016 paper, Tsiogkas *et al.* [19] emphasise the importance of time and energy constraints in the context of AUV missions. Instead of purely seeking a valid solution to a planning problem, a user-defined metric may be used to evaluate and constrain the quality of solutions. We consider domains where a trade-off exists between the probability of successfully completing the mission and maximising the optimisation metric. Consequently, to enable our system to make informed decisions when evaluating plans and goal sets, it is crucial that a suitable method for evaluating the probability of mission success and the optimisation metric is defined for use as a composite objective at run-time.

4. Related work

4.1. MDP-based approaches

MDP-based approaches [11] are a popular representation for non-classical planning problems: those which are stochastic, have a continuous number of states or where the state of the world is only partially observable. An MDP is a tuple $\langle S, A, T, R \rangle$ where S is the set of states, A the set of actions the agent can perform, T the transition function, which represents the probability distribution over states which may be reached by performing an action, and R the reward function, which specifies the reward for performing an action resulting in a transition to a given state. When solving an MDP, the planning process uses the reward function to find a sequence of actions which earns the maximum reward given the probability of each state transition, producing a policy for execution.

Representing their oversubscribed Mars rover domain as an MDP with continuous power usage and time, Bresina *et al.* [13] compute the optimal value function for all contingencies within a small branching plan. They computed the optimal policy for their domain using dynamic programming, discretising the continuous variables representing time and power into 420 and 200 partitions respectively. The optimal value function can be used as a policy, dictating the optimal branch to take at run-time, given current resource availability. However, Bresina *et al.* [13] state that in practice, standard methods of solving MDPs, such as dynamic programming, become computationally infeasible when the search space is large. In a follow-up paper, Feng *et al.* [20] address this computational complexity using an approximate MDP approach, classifying regions of the continuous state space into discrete states according to the value of their utility. However, this approach is restricted to very small problems as a value function over all continuous variables must be computed for each discrete state.

In their 2009 paper, Mausam *et al.* [21] represent the Mars rover problem as a 'hybrid' MDP, using both continuous and discrete state variables. However, due to the size of the state space they are unable to use a dynamic programming algorithm to produce a policy. To address this problem, they developed a

variant of the existing AO* search algorithm (described in [22]), HAO*, which uses forward heuristic search on an aggregate state space. In order to obtain an *admissible* heuristic to drive the search, Mausam *et al.* [21] assume that the resource consumption is monotonic and each action uses a fixed minimum amount of resource. Consequently, although the paper presents encouraging results, the HAO* algorithm will not generate good plans for domains which contain replenishment actions as the assumption that resource use is monotonic does not hold for every action and therefore the heuristic is no longer admissible.

4.2. Replanning

FF-Replan [14] capitalises on the speed and efficiency of the deterministic planner FF [23], replanning from scratch (i.e. replacing the entirety of the current plan with a new plan) mid-execution if an unexpected state is encountered. As FF is a deterministic planner, before generating either the initial plan or performing subsequent replanning, FF-Replan must first remove all probabilistic information from the planning problem, either via single-outcome determinisation or all-outcomes determinisation.

As we consider domains featuring continuous resource uncertainty (and thus each action has an infinite number of outcomes) the use of FF-Replan with all-outcomes determinisation would not be feasible. Instead, we would have to use single-outcome determinisation with a suitable heuristic to select the most appropriate outcome, which would intuitively be to use the mean of the expected resource usage distributions. However, as the number of outcomes for each action is infinite, it is very likely that the observed resource usage would not be equal to the mean. This would result in FF-Replan encountering an unfamiliar state, thus triggering replanning. It is therefore highly likely that replanning would occur after the execution of each action. As the domains we consider feature continuous resources, 'dead-end' states will be encountered if these resources are exhausted. Although it is possible to reason about the use of resources and plan to prevent their exhaustion, as FF-Replan does not reason about the probability of particular outcomes, it would not be possible to ensure that the plan does not put an agent at risk.

In their 2009 paper, Patron *et al* [24] calculate a plan for an AUV up to a given planning horizon, observing the state during execution. Their approach handles uncertainty in the domain by either recalculating a plan when the execution over the current planning horizon has finished, or by greedily recalculating the plan as soon as changes have been detected that would lead to plan failure.

4.3. Contingency planning

Solutions based on the construction and execution of branching plans are popular for domains with resource uncertainty [25, 26, 27, 28].

Citing the inefficiencies of using conventional over-conservative plans in domains with large resource uncertainty, Gough *et al.* [29] present an alternative approach which constructs and executes an 'opportunistic plan'. An opportunistic plan resembles a contingency plan although, in this case, a main plan is

augmented with branches which represent additional opportunities that may be taken at run-time to increase overall plan utility if resource availability allows. Gough *et al.* [29] consider opportunities as "ways of extending the main plan rather than as alternatives to it". Consequently, they do not include contingent branches requiring fewer resources than the main plan, to take in the event that resource usage is higher than expected.

Using a branching plan-based approach similar to that of Gough *et al.* [29], Coles [30] presents an algorithm which augments an initial plan with contingent branches offline, labelling each branch with the conditions of execution. The algorithm is designed for domains which share properties with ours, including over-subscribed goals and resource uncertainty. Both Gough *et al.* and Coles consider branches solely as points to capitalise on opportunities to achieve additional goals. Branches are therefore not used to avert failure, e.g. by reducing the number of goals achieved by the current plan. Both approaches instead guard against failure by ensuring initial plan and all branch conditions adhere to a high confidence level.

Whilst increasing the efficiency of a plan above that of conservative straight-line plans, a very high confidence threshold will, on average, inevitably result in over-conservative plans. Using the AUV domain example, whilst it is very important to safeguard the vehicle, AUV deployments are often very expensive, requiring the chartering of a support vessel to the area of interest. Such costs restrict the frequency of deployments and so it is hugely important that the vehicle fully utilises the time and resources available during the mission to maximise the return of scientific data. While a confidence threshold of 0.95 theoretically almost guarantees the safety of the vehicle, it also means many achievable opportunities are likely to be missed.

4.4. Plan repair

Despite the conclusions of Nebel and Koehler [31], who show that in theory, the costs of modifying an existing solution to suit a new problem are higher than generating an entirely new plan, many authors have investigated plan repair and reported promising results [32, 33, 34, 35]. A popular approach is to extend the ideas of refinement planning [36, 37], which models plan generation as the iterative addition of constraints or actions to restrict the set of all possible plans until only solutions to the current problem remain.

Patron *et al* [38] present a plan repair based technique for AUV operations, demonstrating their work on a REMUS 100 AUV. They utilise both domain independent and domain specific knowledge ontologies to discover and repair failures and gaps in the plan caused by observations of the vehicle state mid-mission. For example, varying the spacing of a lawnmower survey in response to sensor malfunction.

In the context of online plan repair, Fox *et al.* [39, p. 212] highlight the benefits of a modified plan which closely resembles the original, stating, "where it can be achieved, stability is an important property that allows confidence to be maintained in the safe operation of an executive within its environment". They define a 'plan stability' metric to measure the difference between the original

and repaired plans, and adapt the local-search based planner LPG [40] to use this metric when evaluating partial plans for refinement. They compare the result of their plan repair strategy to that of replanning and find that plan repair produces plans more efficiently and with much greater stability than replanning.

Long *et al.* [41] present a system that uses online plan modification to enable a planetary rover to perform opportunistic science. The execution of the initial plan, generated on Earth, is monitored by a component named TVCR [42]. Actions in this plan which relate to a particular task are grouped into 'plan fragments', along with any constraints on their execution. If the executive determines that the current plan is no longer valid, the plan is modified by first removing fragments (according to a predefined science priority value) until an executable solution is found. Following this, extra fragments (either from the original plan or a library) may be added to fully utilise the rover's resources. Plan fragments may also be re-ordered to utilise resources and meet temporal constraints. If re-ordering causes breaks in the causal structure, 'glue' activities, which are independent of the fragments and perform tasks such as changing the mode of an instrument, are selected for addition to the plan using means-end analysis. Long *et al.* [41] highlight the operational challenges of planning for such high-risk domains and echo the views of the AUV community [3] when stressing the importance of fostering trust between both the planning community and robot operators. With similar motivations to [35], Fox *et al.* [42] describe how the use of fragments and constraints was motivated by the desire to include and represent the knowledge used by human mission planners when constructing rover plans.

Sharing motivations with both ourselves and others in the literature, Bresina and Washington [43, p. 24] present a flexible on-board executive which monitors and reacts to changes in the expected utility of the rover's plan. An initial contingent schedule, represented as a straight-line plan annotated with a tree of alternative branches, is constructed offline by mission operators aided by an automated system. Contingent branches are added at specific points to define the actions to take should an 'expected deviation' occur during execution, such as a predictable action failure. While these branches are linked to particular points in the plan, any plan within a library of alternative plans, created in advance by human operators, may be used whenever the conditions for its execution are met. As the executive is utility-driven, a branch or alternative plan will only be taken if its expected utility exceeds that of the remainder of the current plan. The utility of the plan is modelled as a distribution which maps the temporal uncertainty of each action to the expected utility of executing the action followed by the remainder of the plan. Washington *et al.* [44] state that they do not currently include uncertainty in their resource models, leaving this as further work.

5. High-level overview

We devised a novel plan modification approach to planning under uncertainty which seeks to maximise reward whilst minimising online plan generation and computation. Our flexible approach allows a plan and goal set to be updated and refined during execution in response to fluctuating resource availability. During execution if, at pre-defined decision points, the executing agent is observed to have more resources available than previously expected, specifically enough to potentially complete an additional goal, the ability to update the plan to include such goals could significantly increase the reward of the mission. Conversely, if resource usage has been higher than expected and thus resource availability is low, the probability of successfully completing the mission will decrease. If it falls below a user-defined acceptance level, removing existing goals could potentially avert plan (and thus mission) failure.

A schematic of our full approach is presented in Figure 1 and the key elements discussed in detail in this paper. Instead of adopting an online planning approach, our approach pre-computes individual plan fragments for each goal in the over-subscribed problem offline, to assist with updating the plan during execution. To combine these fragments at run-time, we have developed novel algorithms which examine and exploit the causal relationships between actions in the plan and the goal set, a method inspired by concepts from classical planning. The online component of our approach is presented as pseudocode in Algorithm 1. Causal structure can be computed from the preconditions and effects of each action as defined in the domain action schema. For example, the preconditions of an action to transmit dataset d, require that d has first been collected, which in turn requires the vehicle to be at the survey location. Whilst different combinations of actions could be used to satisfy a goal (e.g. a different route between locations may be taken), the causal structure is associated with the goal itself and remains unchanged. After modification of the current plan using the fragments, we then evaluate the suitability of the resulting plan, given resource usage uncertainty, by estimating the probability of successfully completing the plan using the Gaussian distributions representing the expected resource usage of each action.

By combining precomputed fragments at run-time instead of computing a complete contingency plan, we aim to reduce the large number of possible contingent branches which would otherwise need to be computed and evaluated at run-time. With an over-subscribed goal set, the number of goal set combinations that would need to be considered and planned for in order to create a contingency plan with sufficient coverage is significant. By combining fragments at run-time, we do not have to exhaustively consider all contingencies, instead we only consider those which are relevant in the context of the states encountered during execution. In addition, by computing of a set of plan fragments at a finite number of decision points within the initial plan offline, our approach bounds the space of possible plans, reducing uncertainty over resulting vehicle behaviour when compared to a fully-online replanning approach.

Whilst the ultimate motivation of this work is to improve the mission plan-

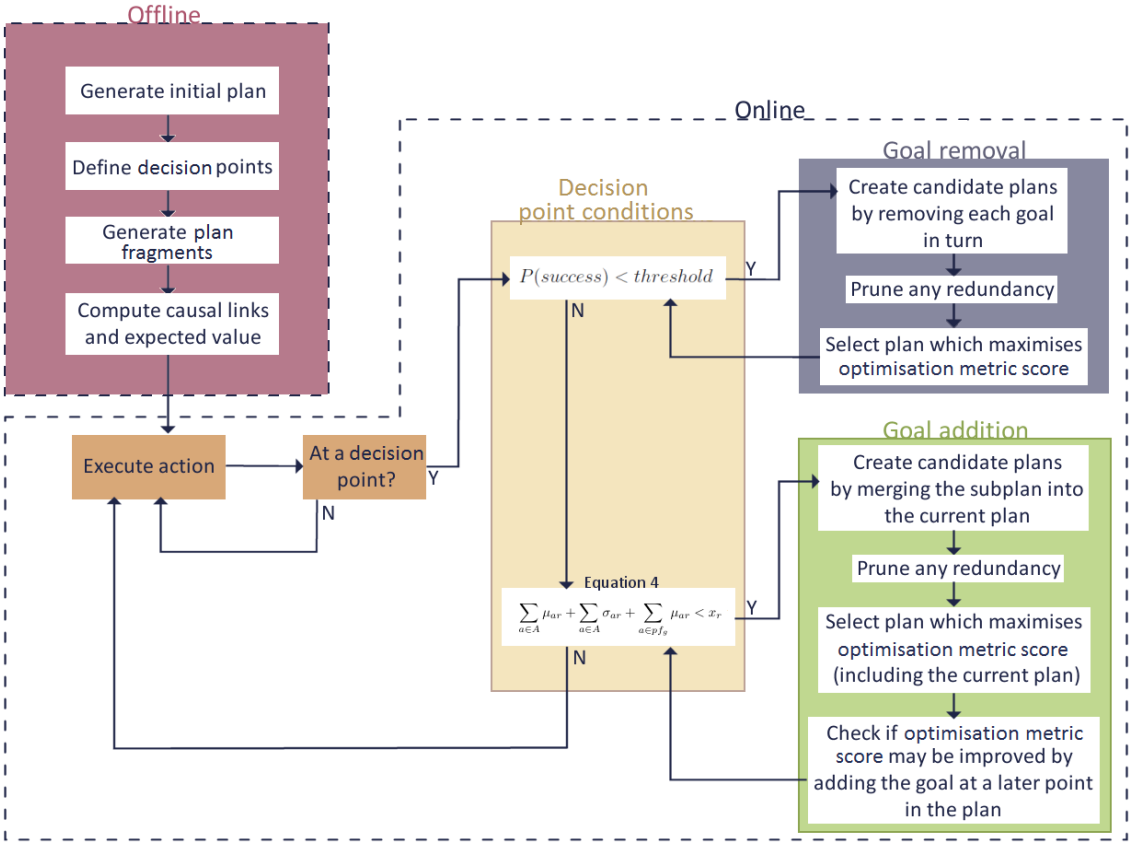

Figure 1: A high-level overview of our full approach, illustrating the ordering of key components within the system. The box labelled 'Offline' represents tasks which are performed prior to execution (see Section 6). All other tasks occur during execution. Parts of the system relating to goal removal (see Section 9) and goal addition (see Section 10) are also labelled. The central box represents the plan modification conditions for both goal removal and goal addition.

ning capabilities of operational robots in real-world scenarios, the implementation and testing of our system on a physical platform was outside the scope of this project. To this end, we have implemented a stochastic simulation alongside our system which allows current resource availability to be 'observed'.

6. Pre-computation

Prior to execution, an initial plan is generated and decision points are defined. At each decision point the initial plan is then augmented with plan fragments, each achieving an individual goal from the overall problem.

6.1. Initial plan generation

We assume that the initial plan generation (and goal set selection) occurs prior to the start of the mission, when computation is not constrained.

Our approach uses a deterministic representation for the generation of a straight-line plan, treating resource usage as discrete. In this paper, all experiments define this discrete usage to be the mean of the related distribution, although this may be adjusted depending on the domain used and the preferences of the user. Discretising the resource usage distributions allows the use of existing classical planners, avoiding the large continuous state space and consequent computational complexity which makes the use of MDP solvers infeasible for many realistically-sized domains. After plan generation, we evaluate the resulting plan using a probabilistic model of resource usage, calculating the probability of the plan's successful execution and its optimisation metric score. The probability of successfully completing the plan must exceed a minimum threshold, defined by the user. We use Metric-FF [45] for all plan generation within the experiments presented in this paper.

6.2. Definition of decision points and plan fragments

We place decision points in the plan where modifications to the goal set are likely to be the most beneficial. We gain the most information about the probability of a plan completing successfully, and thus its expected reward, after performing actions with the largest associated resource usage uncertainty. This is because the uncertainty surrounding the resource requirement of the remainder of the plan is significantly lower than it was prior to the execution of the uncertain action, resulting in improved, more-certain estimates of the likelihood of plan success. Given this reduction in resource uncertainty following the execution of the most uncertain actions, this is the most informed time to consider goal set modifications. Consequently, we define decision points after the n actions with the largest standard deviations, where n is defined by the user as a percentage of the plan length, such that $n = 0\%$ results in a straight-line plan with no opportunities for modification, while $n = 100\%$ would consider the modification options after every action in the initial plan. We evaluated the effect of varying the number of decision points n on the achieved reward and computation required, presenting the results in Section 11.2.

An important point to note is that decision points and plan fragments are computed offline, prior to plan execution, and so may only be defined at points following actions in the original plan. If a modification is made during execution and new actions are added to the plan, these new actions will not be followed by a decision point, regardless of the size of their associated standard deviations or the value of n. To add decision points mid-execution would require increased computation as new plan fragments would have to be generated online. This is disadvantageous as battery used for online computation is not then available for performing actions. Our model of resource usage does not currently represent resources used for computation and so the system would be unable to reason about when to halt the planning process. This meta-planning problem is outside

the scope of this work and so we instead monitor and evaluate the computational cost of our approach, with a view to minimising online computation. There is also a question of when to stop adding plan fragments and decision points. Contingency planning approaches are infeasible due to the significant number of contingent branches in domains we consider, where many independent goals may be both added and removed from the plan throughout execution. Our algorithm may add a goal at one decision point, before potentially removing it at the next, which would lead to many nested fragments being generated online. If resource availability dictates, such modifications to the goal set may theoretically then be repeated throughout the execution of the plan. Our approach also leads to predictable plans. In theory, the user can review the complete set of fragments in advance and so the resulting set of possible plans is finite. By creating plan fragments online (in a similar way to replanning), the predictability is lost, especially if the set of fragments (and nested fragments) is very large.

At each decision point, we generate a plan fragment for each goal in the overall problem. These fragments each achieve a single goal and may later be combined with both other fragments and the initial plan to form a plan that satisfies a set of goals. We assume all sub-problems are sub-sets of the overall problem and therefore all relate to the same domain definition. The initial state of the sub-problem is defined as the expected state at the associated decision point. When computing the expected state, the resource usage of each action during this state simulation is assumed to be the mean of its usage distribution. The goal of a sub-problem is defined as simply a single goal from the overall problem. All other goals (such as those in the initial problem) are not included.

7. Plan evaluation

Prior to execution, and then sequentially after both the completion of each action and any modifications, the current plan is evaluated to check it meets all goal and resource constraints. We compute a representation of the causal structure of the current plan using the definition of each action in the action schema, to be used at run-time to aid online plan modification.

7.1. Calculation of success probability

As all actions consume or renew the agent's available resources, the probability of successfully executing a plan is closely related to resource usage. If an action either causes the agent's resources to be exhausted or to fall below that required by the next action, execution will halt and the mission will fail. Consequently, prior to execution and then sequentially after both the completion of each action and any modifications, we compute the success probability of the plan and its optimisation metric score. If executing an action affects multiple resources (e.g. both battery and memory in our AUV domain), the usage of each resource is represented using separate distributions and the probability of success is calculated separately for each resource.

As an example, we present the calculation of success probability for the two resources in our example AUV domain. Battery decreases monotonically

throughout a mission, whilst memory is reusable following successful data transmission. The uncertainty over resource usage is represented using Gaussian distributions for both resources.

Battery

As the usage of battery resource across subsequent actions is independent, we sum the distributions for all actions into a single distribution T representing the battery usage of the plan, by summing their associated means and variances. Given this combined distribution T, we can then calculate the probability that executing the plan will be successful, requiring less battery than is available to the vehicle. First, we integrate the combined Gaussian distribution to produce the cumulative distribution function:

$$CDF = \int \left(\frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sigma\sqrt{2\pi}} \right) .dx = \frac{1}{2} \left[\text{erfc} \left(\frac{\mu - x}{\sqrt{2}\sigma} \right) \right] \tag{1}$$

If we solve the cumulative distribution function where x equals current battery availability and μ_T and σ_T are the mean and standard deviation of the combined distribution T, we calculate the probability of plan execution requiring less battery than is available. This result is the probability of success for battery, $P(success|battery)$:

$$P(success|battery) = \frac{1}{2} \left[\text{erfc} \left(\frac{\mu_T - currentBattery}{\sqrt{2}\sigma_T} \right) \right] \tag{2}$$

Memory

As memory is a reusable resource, the calculation of $P(success|memory)$ is more complicated. As in the case of battery, the Gaussian distributions associated with actions which consume memory have a positive mean, while actions which renew memory have a negative mean. If we employ the same approach as when calculating *P(success)* for battery, by summing the means and variances to create a combined distribution, we discover a problematic interaction between actions which consume and renew memory, e.g. by collecting and delivering/transmitting datasets. The memory usage for each of these actions is based on the size of the dataset involved. Therefore, the mean of the distribution when transmitting/delivering a dataset d is the exact negation of the mean associated with its collection. If we then sum the mean memory usage over a typical plan, which transmits/delivers all collected datasets, the combined mean memory usage of the plan is 0. Consequently, integrating this distribution and solving for current memory, as for battery in Equation 2, results in $P(success|memory) = 1$. This is not an accurate reflection of the true probability of plan success. To avoid this over-estimate of $P(success|memory)$, we instead subdivide the plan at the points where memory is renewed (i.e. where memory usage is at a local maxima) and calculate the probability of successfully reaching each of these points. We can then treat each of these subdivisions as we treat battery, integrating the combined distribution and solving for current

memory. To determine the success probability of the whole plan, we consider each subdivision separately, i.e. if the success probability of any subdivision falls below a predetermined threshold, the plan is deemed invalid.

As the size of a dataset is known at the point when it is transmitted, the uncertainty regarding its collection should be cancelled out by its successful transmission (i.e. the value of memory before and after a collect-transmit sequence should be equal). Theoretically, this means that the variance of a transmit/deliver data action should be subtracted from the combined total. However, a general formula for calculating $P(success|memory)$ was desired which does not assume plans/plan fragments contain both the collection and subsequent transmission/delivery of a dataset. If a plan fragment contains a transmit (replenishment) action but not the associated collect (consuming) action, the combined variance of the plan fragment may be negative, which would violate the definition of variance and standard deviation, preventing the calculation of success probability. Consequently, by always summing the variances (i.e. ignoring the reduction in variance when renewing memory), our calculation of success probability tends to over-estimate the variance of memory usage within a plan. However, we felt that this general solution is acceptable as the resulting $P(success|memory)$ will always be an under-estimation of the true value and thus does not increase the risk to the vehicle.

7.2. Evaluation of the optimisation metric

A plan with a high success probability is attractive as it suggests the risk to the agent and mission are minimal. However, in many domains it is also important to use the available resources to maximise the completion of valuable goals and consequently the associated reward. To represent and reason about the balance between risk and reward, our approach evaluates plans against an optimisation metric. As discussed in Section 3.3, the user-defined metric should reflect the characteristics of an optimal solution for the given domain.

To calculate the optimisation metric score for an AUV domain plan, the success probability is weighted by the reward:

$$E[v] = \sum_{x=1}^{n} \left[\left(P(success|battery_x) \times \prod_{q=1}^{x} P(success|memory_q) \right)^2 \times reward(x) \right] \tag{3}$$

At each subdivision of memory x, where $x = 1, 2, \ldots, n$, we calculate the $P(success|battery)$ for the same range of actions, $P(success|battery_x)$. To calculate the success probability for memory over the same interval as battery, rather than just over the most recent subdivision, we multiply the success probability for each subdivision prior to x, $P(success|memory_q)$, represented in Equation 3 as the product term. The resulting value is then multiplied by $P(success|battery)$ and squared to increase the influence of success probability over reward, as reward may be highly variable. This is repeated for all subdivisions of memory and the results summed to give a single result for the optimisation metric, which is used when comparing plans.

8. Plan execution monitoring

Provided the initial plan exceeds the required success probability threshold, execution of the plan may begin. After the successful execution of each action, the success probability and optimisation metric value of the plan are recalculated given observed resource usage. When at a decision point in the plan, if $P(success)$ falls below a predefined threshold, the algorithm will consider removing a goal from the current goal set in an attempt to reduce resource costs and consequently increase the probability of finishing the plan successfully. Conversely, if the current resources are sufficient to potentially allow the inclusion of an additional goal, the algorithm will attempt to merge a new plan fragment into the current plan. Provided the success probability of the resulting plan exceeds the required threshold and the optimisation metric score of the new plan exceeds that of the current plan, the new plan is selected for execution.

The $P(success)$ threshold may be defined by the user to reflect their tolerance for risk. As the threshold tends towards one, the system will be much less tolerant of risk, over-conservatively removing goals to safeguard the agent, prioritising mission success over reward. Conversely, given a much lower threshold the system will tolerate greater risks and thus is more likely to attempt to add in extra goals. Such behaviour is likely to increase the rewards achieved but this is potentially at the cost of mission success. To balance the trade-off between risk and reward, for experiments and analysis within the AUV domain we use a threshold for $P(success)$ of 0.841. Given the definition of the standard deviation of a normal distribution[2], if the success probability for either battery or one or more subdivisions of memory falls below 0.841, this implies the vehicle has less resources than the total combined mean plus one standard deviation of the remainder of the plan.

9. Online goal Removal

At each decision point, if the success probability of the current plan falls below the minimum threshold, goal removal is considered. Each goal in the current goal set is individually passed to the goal removal algorithm which removes the goal, producing a candidate alternative plan. When removing a goal from the planning problem, we can also remove any actions which were only present to achieve that goal and are thus no longer required.

An action may be removed if it produces an effect which satisfies the removed goal and none of its effects are required by any other goals or actions in the plan. If an action is removed, this process is then repeated for any action whose effects were required by the removed action. The goal removal algorithm recursively

[2]In a two-tailed normal distribution, there is a 68.2% chance of the value falling within 1σ of the mean (μ). However, we are only interested in the single-tailed case where usage is less than 1σ above the mean. There is a 34.1% chance of sampling a value between μ and $\mu + \sigma$, which when added to the 50% chance of sampling a value below μ gives us a total probability of 84.1% or 0.841.

examines the causal structure of the plan until no more actions may be removed — see Algorithm 2.

Formally, given a goal g to remove, we construct the set D_g of actions to remove by first setting $D_g = \{g\}$, then repeatedly adding to D_g any action a such that all causal links from a lead to elements in D_g. All actions in D_g only contribute to the goal we wish to delete, so can be removed from the plan without impacting other goals.

9.1. Removing redundant actions

Following the removal, or addition, of a goal and associated actions, the resulting plan may contain newly redundant actions. Using our AUV domain as an example, the following plan achieves the goals: *data_collected(d2)* and *at_loc(l1)*.

o_1: `move(l1, l2)`
o_2: `collect_data(l2, d2)`
o_3: `move(l2, l1)`
o_4: `surface()`
o_5: `end_mission(l1)`

Removing the goal *data_collected(d2)* results in the following plan:

o_1: `move(l1, l2)`
o_3: `move(l2, l1)`
o_4: `surface()`
o_5: `end_mission(l1)`

In this new plan, actions o_1 and o_3 are now redundant, consuming resources without furthering the completion of any goals. While it may seem obvious to the reader that neither o_1 nor o_3 are required within the resulting plan, the action o_3, *move(l2, l1)* provides for the goal of ending the mission at location *l1*, *at_loc(l1)*. For this reason, o_3 is not considered for removal by Algorithm 2. Action o_1 does produce an effect (*at_loc(l2)*) that is required by the removed action o_2 and is thus considered for removal. However, this literal is also required by action o_3, which prevents the removal of o_1 from the plan.

To avoid such redundant behaviour and the waste of resources in such scenarios, following the removal of a goal any redundant actions are detected and removed from the plan, see Algorithm 3. Redundant actions are detected by searching the plan for pairs of duplicated states, ignoring the continuous state variables representing resource usage as these will inevitably differ. If the discrete state variables of two states are identical, the actions between them do not further the completion of any goals (as there has been no change to the state variables either towards or away from the goal conditions) and are therefore redundant. Following the removal of redundant actions, the resultant plan is now shorter, requires fewer resources to execute and thus has a higher success probability and optimisation metric score.

9.2. Goal selection

Once a list of candidate alternative plans has been produced, these are analysed to select the best plan, and thus the goal to remove. Considering the set

of all candidate plans:

- Remove any plans which violate any of the resource constraints, i.e. where the resource preconditions of any actions are not met (see Section 2.1).
- Remove any plans where the probability of success, $P(success)$, does not exceed the user-defined minimum threshold (see Section 8).
- Of the remaining plans, select the plan which maximises the optimisation metric.

If none of the candidate plans exceed the $P(success)$ threshold, the system removes the goal which, post-removal, results in the plan which maximises the optimisation metric. It will then attempt to remove additional goals, using the same method as before, until the success probability of the resulting plan exceeds the threshold. In an extreme case, this may require all goals to be removed from the current problem, aborting the mission. Whilst drastic, this behaviour prioritises the safety of the agent in resource-constrained domains and ensures it operates within the user-defined constraint of risk, as defined by the minimum $P(success)$ threshold. Using the example of an AUV deployment, such a scenario would initiate emergency routines, for example by dropping an abort weight and floating to the surface.

10. Online goal addition

At each decision point, following consideration of the goal removal criteria (as described above), the algorithm iterates over all candidates for goal addition (excluding any goals which have just been removed and therefore may not be re-added at the same decision point) to re-allocate any surplus resources thus maximising reward. For each candidate goal g at the current decision point, the algorithm checks whether the mean resource requirement of the associated plan fragment pf, added to the expected resource usage of the current plan p, is less than the resource available in the current state (i.e. at the decision point). The algorithm considers adding a goal if:

$$\sum_{a \in A} \mu_{ar} + \sum_{a \in A} \sigma_{ar} + \sum_{a \in pf_g} \mu_{ar} < x_r \tag{4}$$

where A is the set of actions remaining in the current plan p; μ_{ar} and σ_{ar} are the mean and standard deviation of the expected usage of resource r by action a; x_r is the remaining quantity of resource r at the current state.

This check is only performed for consumable resources. For reusable resources, the current available resource will differ depending on where the new plan fragment is merged within the current plan. Instead, we check the validity of the plan following the merge, ensuring that all constraints are satisfied. If all resources within a domain are reusable, all plan fragments would need to be considered (which for certain domains may increase the cost). We present

a domain-independent approach in this paper, but domain knowledge could be exploited to further prune the space of plan fragments.

Once the set of candidate goals G ($g \in G$) has been pruned to remove those which do not meet the consumable resource constraints (see Equation 4), the states of the plan fragment are updated to include any additional literals from the current state (at the decision point). In our AUV example, these would include datasets which have already been collected.

10.1. Plan merging algorithm

Given the plan fragment associated with the additional goal pf, the current plan p, and the set of states S and causal links CL within p, the system attempts to merge pf into p, interleaving actions to create a valid solution for the new combined goal set. To create a valid solution, the recursive merge algorithm (MERGEPLANS, shown in Algorithm 4) takes the first action of the plan fragment pf and compares its preconditions with each state in the original plan, sequentially from the current state, producing a list of candidate *merge points*, i.e. states which meet the preconditions of the action. The algorithm then checks whether any causal links in the current plan would be threatened by the inclusion of the new action at each merge point. If merging an action causes a threat, the algorithm searches the plan fragment in an attempt to find an action whose effect subsequently restores the threatened link (e.g. by satisfying the precondition). If a restorative effect is found and this effect is then maintained until the end of the plan fragment, the restorative action may be inserted prior to the consuming action and thus the threat is marked as resolvable. If no links are threatened or all threats are resolvable, we add the action to the plan at the current merge point, update the plan's causal links and call the function again to insert the next action in the fragment. This continues until all valid merge points for each action in the plan fragment have been considered. The set of merged plans Z is then returned for evaluation. If a threat is not resolvable, provided the action does not achieve a goal, the recursive algorithm skips the action as it may not be required when the two plans are combined.

If no valid plans are found using the MERGEPLANS algorithm, a *stitching plan* is generated which uses the goal state of the plan fragment as its initial state and the unsatisfied preconditions of the remainder of the current plan as the goal. By returning the state to this point, the stitching plan resolves all threats caused by the plan fragment. After generation, the stitching plan is concatenated onto the end of the plan fragment pf and the newly extended pf is passed to the MERGEPLANS algorithm to be merged into the current plan. Computing a stitching plan is not guaranteed to be successful in all cases, as whether or not a plan can be generated depends on the domain, problem and the characteristics of the planner used. Heuristics used by some planners may prevent them from finding a plan for a given problem even if one exists. If the generation of a stitching plan fails, the merge will fail and the next goal and associated plan fragment will be considered. If no plan fragments can be successfully added, execution of the current plan will continue.

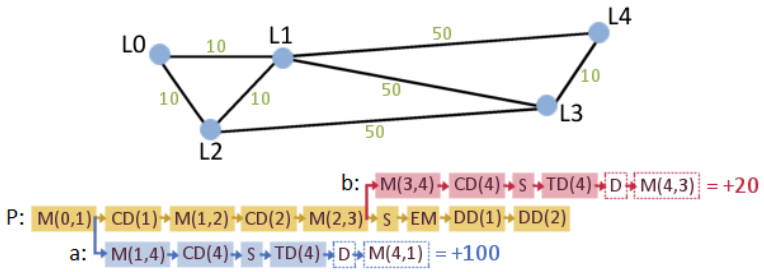

Figure 2: Schematic illustrating how delaying the addition of a goal may be beneficial, using an example from our AUV domain. The graph shows the connectivity of locations in a small problem and the distances between them. The current plan is labelled P. By including a new goal to transmit dataset 4 (TD4) at the current state, the move actions (M) of the resulting plan (labelled a) increase the cost of the current plan by 100 (all other action costs are ignored in this example). By instead delaying the addition of the new goal, the resulting plan (labelled b) only increases the cost by 20. This is because from location 3, the diversion to location 4 is much shorter than from location 1. As the reward for dataset 4 is the same for both plans, the cheaper red plan achieves a higher *expected reward*.

10.2. Goal selection

Once there exists a set of valid merged plans for each candidate goal at the current decision point, the system evaluates these new plans and selects the best to replace the current plan.

First, the success probability is calculated for each merged plan. If $P(success|resource)$ for any subdivision falls below the acceptable threshold, the plan is discarded and pruned from the set. Of the plans which exceed the $P(success)$ threshold, the plan with the highest optimisation metric score is chosen. If none of the new plans has a success probability greater than the *P(success)* threshold or an optimisation metric score which exceeds that of the current plan, the new plans are discarded and execution continues with the current plan i.e. with no additional goals.

10.3. Delaying goal addition

Once the new plan and associated goal have been selected, the algorithm considers the possibility that adding the goal at a later decision point, instead of at the current state, may increase the optimisation metric score of the resulting merged plan. Unlike removing a goal, which averts risk by remedying a violation of the $P(success)$ threshold and thus needs to occur sooner rather than later, delaying the addition of a new goal until a later state may result in a plan which achieves a higher score against the optimisation metric, as illustrated in Figure 2. Gough *et al.* [29, p. 25] share our rationale for delayed goal addition, stating that "it could be the case that a higher utility opportunity will be missed because the resources were spent earlier on a low-utility opportunity".

When considering whether to delay the addition of the selected goal, our algorithm selects any subsequent decision points, within the current plan, where

there exists a plan fragment for the goal. A list of valid merged plans is constructed for each decision point and the list of merges is then pruned according to the $P(success)$ threshold, as before. This time, the optimisation metric score of merging the new goal at a later decision point is compared to that of merging it at the current state, and whichever is highest is selected as the new current plan. An alternative option would be to initially consider all goals at all decision points, selecting a goal to add at a particular decision point from the full set. However, as resources used for computation are then not available for use when performing actions, we seek to minimise online computation. To this end, we instead greedily select the goal which provides the most benefit at the current state, before checking whether adding it at a later decision point would instead increase the optimisation metric score of the plan.

Following the addition of a goal, the entire goal addition process then continues, attempting to add further goals until either no candidate plan fragments meet the addition criteria or the optimisation metric score of the merged plans fails to exceed that of the current plan. Consequently, at a single decision point, 0:N goals may be removed before 0:N goals (excluding those removed at this decision point) may be added. As such, the space of combinations of plan fragments is greedily explored.

11. Empirical evaluation

In this section, we present the results of four experiments that evaluate key parts of our approach. These experiments use our example AUV domain, as defined in Section 2. Resource usage is essentially domain neutral and so we did not evaluate the performance of our full system on additional domains. However, the performance of the merging algorithm is domain specific, relying on the causal structure of individual plans. As the performance of the merge algorithm is a limiting factor in the success of our overall system, in Section 12 we analyse and discuss the merge algorithm when applied to existing International Planning Competition (IPC) domains.

11.1. Methodology and test problems

Four AUV domain problems were defined, each relating to the same topology of 20 survey locations shown in Figure 3, with one dataset per location. Each of the four problems is a different subset of the same overall problem. Problems one and two both have an initial goal set to return five datasets. The collection and return of the remaining 15 datasets in the overall problem may then be added as additional goals, should resources allow. During execution, all goals may also be removed from the current goal set. Problems three and four extend problems one and two respectively, increasing the size of the initial goal sets to 10 goals each, thus reducing the number of goals which may be added during execution to 10.

For each problem, we varied the resource availability, defining three levels of resource availability: *low* (L), *medium* (M) and *high* (H). Defining absolute

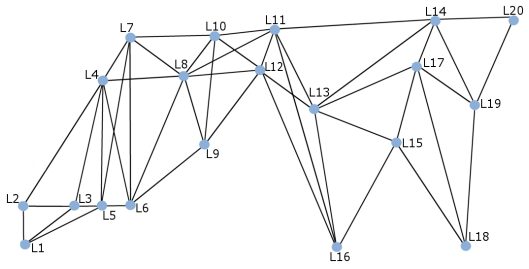

Figure 3: Connectivity of locations and datasets in the overall AUV problem. Circles represent data-collection opportunities.

values for both memory and battery for each resource level would not permit direct comparison of the results as each problem and plan has different resource requirements. Consequently, in these experiments the three resource levels are defined relative to the resource requirements of the initial plan for each problem. Representing the minimum required to execute the initial plan, the *low* (L) level was defined as:

- Initial battery (L) = the sum of the mean battery usage for all actions in the initial plan, plus the sum of the associated standard deviations.
- Initial memory (L) = the mean plus the associated standard deviation of the action with the highest expected usage in the initial plan.

Medium and *high* availability were then defined as 110% and 120% of *low*, respectively.

As the uncertainty in the domain is significant and the resource usage of each action is randomly sampled for each trial (from distributions where the standard deviations range from $< 5\%$ to approximately 25% of the mean), it is likely that a direct comparison of the results would not be fair, for example, when comparing the results of a run in which the sampled resource usage was very high, to one in which it was very low, despite the goal set and initial resource availability being identical. To facilitate a fair comparison, both between runs and across experiments, the resource requirements of each action and the rewards for each dataset were sampled and fixed prior to the execution of each run, e.g. when testing two approaches A and B, the first run of A will use the same sampled resource usages and rewards as the first run of B, the second run of A will use the same samples as the second run of B, and so on. These sets of fixed values were then used by all experiments. By fixing the sampled values for each action in each plan, if a plan contains multiple instances of the same action, all instances will use exactly the same amount of resources. This is not a true reflection of reality, but as the repetition of actions in the AUV domain is low, this was not deemed to be a significant problem.

In all experiments, the minimum accepted success probability threshold was defined as 0.841, as discussed in Section 8. Throughout our experiments, all

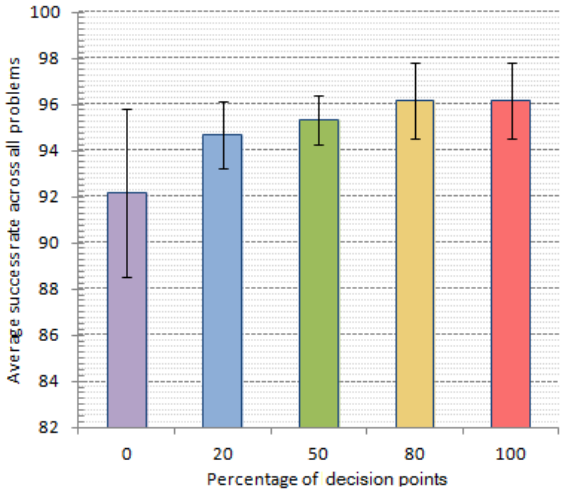

Figure 4: Graph showing the average success rate, across all problems, achieved whilst varying the number of decision points.

results were found to be statistically significant to within the 5% significance level unless otherwise stated. The chi-squared test [46] was used when evaluating success rate whilst the Wilcoxon signed rank test [46] for paired samples was used when analysing reward.

11.2. Experiment 1: Determining the quantity of decision points

We evaluated the effect of varying the number of decision points on the reward achieved and on the probability of successfully completing the mission. The experiment compared five different levels of decision points, defined as percentages of the total number of actions in the initial plan. The number of decision points was considered at set levels, varying from 0%, representing a fixed straight-line plan, to 100% which permits plan modification after any action in the plan. For each problem/resource combination, the experiment was repeated 50 times.

11.2.1. Analysis of results

As the runs are grouped to permit direct comparison, in the manner of paired tests, it was important that failed runs should not be compared with successful ones. As reward is heavily penalised during failed runs, the average reward would be significantly affected by the presence of failed runs. Consequently, the analysis of reward was treated separately to that of success rate.

11.2.2. Success rates

The average success-rates achieved for each proportion of decision points are shown in Figure 4. Over all problems, the inclusion of plan modification (i.e. 20-100%) outperformed the straight-line, 0% case, achieving an average of 95.6%

compared to 92.2%. When resources were *low*, 0% decision points resulted in a larger number of failures. This is to be expected as, with no opportunities for plan modification, the system is unable to remove goals to increase the chance of mission success. Considering only the *low* cases, the average success rate of the 0% runs is 83.5%. Also only considering *low* runs, 20-100% decision points achieved an average success rate of 96.63%, significantly higher than that of the 0% case. The success rate for 0% improves when the resource availability is *medium* (96%) or *high* (97%). This is to be expected as, in these cases, the additional resources are surplus to the requirements of the initial plan and are not reallocated to further the completion of additional goals. The success rates for runs using between 20% and 100% decision points are very close, with only 1.5% separating them — a difference which was not found to be significant.

In summary, when resources are plentiful, the presence of decision points has little effect on the success rate achieved. However, in higher-risk scenarios when resources are constrained, the presence of decision points within a plan is able to avert failure, avoiding the reduction in success rate suffered by the straight-line plan.

11.2.3. Average reward

When resource availability was *medium* or *high*, plans with decision points achieved, on average, 7.9% higher rewards than those with 0% decision points. However, when resource availability was *low*, the trend was reversed; plans with decision points achieved, on average, 7.7% lower rewards than those with 0% decision points. When resource availability is heavily constrained, decision points allow goals to be removed, sacrificing reward to increase the probability of mission success. A straight-line plan (0% decision points) is unable to make such updates and so the higher average in the *low* case indicates that during some runs, the plan modification algorithm removed goals, reducing the available reward.

To investigate the cases in which decision points were present, the average reward for each problem was calculated as the percentage improvement over the 0% case, as this provides an appropriate base-line. An overall average may then be calculated across all resource availabilities for each level of decision points, facilitating fair comparison across all problems (see Figure 5). Of the cases with decision points, 80% and 100% slightly outperform 20% and 50% on relative average reward, although the difference is small and was not found to be statistically significant. From these overall averages, we can conclude that any amount of decision points increases the overall average achieved reward above that of the 0% straight-line case. If we also include failed runs (whose achieved reward is subject to a penalty), the trend is the same.

In summary, where decision points were used, the average achieved reward was found to reduce when resources were constrained. This was expected as plan modification allows an agent to sacrifice reward to prioritise success rate. However, when resources were more plentiful, the average rewards achieved by plans using decision points were 7.9% higher than achieved by the straight-line plans. This shows that modifying the plan allows an agent to use surplus

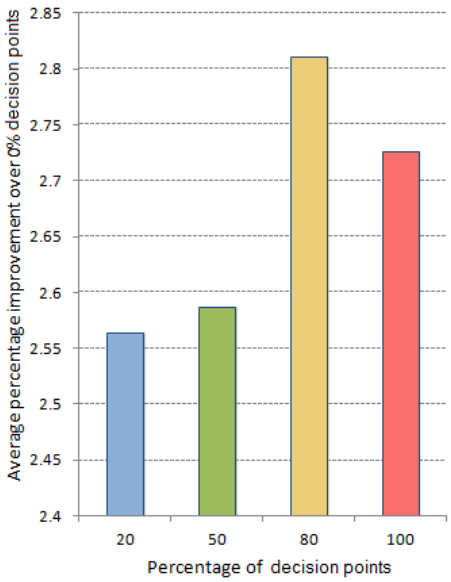

Figure 5: Graph showing the average percentage improvement in achieved reward over that of the straight-line plan (0% decision points).

resources to capitalise on additional rewards.

11.2.4. Computation time

As battery used by the agent for computation is then not available for collecting and delivering datasets, it is important to consider the computational cost of online plan modification. As all our experiments were performed in simulation, it was not possible to measure the battery usage directly. Instead, computation time was measured to approximate computational cost. The simulation assumes all actions are executed instantaneously.

The computation time for each percentage of branching points (0, 20, 50, 80, 100) given each problem was calculated as the average of 50 runs, each performed on the same machine — a laptop with a 2.40GHz Intel Core 2 Duo CPU and 3GB of RAM. The computation times are affected by the presence of timeouts imposed on plan generation. If Metric-FF fails to find a plan using enforced hill-climbing, it resorts to a greedy best-first search [45] and may take many hours to find a solution. This is a prohibitive length of time for both online planning and the practicalities of our experiment. Consequently, a two minute timeout was imposed on initial plan generation and a ten second timeout for the generation of each plan fragment and stitching plan. Due to the simplicity of plan fragments and stitching plan problems, ten seconds is more than sufficient for Metric-FF to complete enforced hill-climbing.

Figure 6 shows the average computation time for each level of decision points across all problems. Each bar is then subdivided to show the average proportion

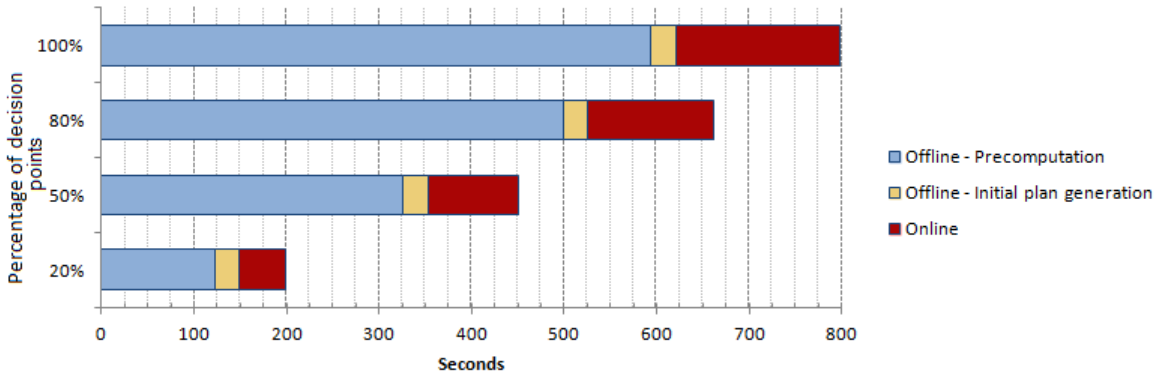

Figure 6: Graph showing the average computation time over all problems.

of time spent on both online and offline computation. The computation times for 0% decision points were not included in Figure 6 as both the pre-computation time (including the computation of plan fragments and decision points) and the online time were negligible. The vast majority of the time required by 0% runs consisted of the time taken to generate the initial plan which was common to all levels of decision points. The total computation time increased with the number of decision points. This was expected because additional plan fragments must be first generated offline prior to being merged and evaluated during execution. As all offline computation occurs prior to the deployment of the agent and the start of mission execution, offline computation time is not tightly constrained. As with total time, online computation time also increased with the number of decision points. The largest average online computation time recorded was 304.1 seconds for a given problem with 100% decision points. Whilst this may initially seem quite large, if this experiment had instead been performed in real-time using a vehicle such as Autosub Long Range 1500 (which runs at ROS-based control system on a 800MHz Dual Core ARM Cortex A9 with 1GB of RAM and has an endurance of six months [47]), a total online computation time in the order of 5 minutes would comprise a very small percentage of total mission time.

Given that the initial plan generation times were fast for these problems, the reader may question the value of online plan modification compared to online total replanning (discarding the current plan and computing a new one from scratch). To this end, we performed an experiment, again considering the AUV domain, to compare the computational cost of our plan modification approach to that of replanning from scratch using Metric-FF. The methodology and full results are presented in [48], with the key findings summarised below:

- When resources were plentiful, replanning with Metric-FF was very fast. However, when Metric-FF was unable to find a plan using enforced hill-climbing, the computational cost of replanning increased significantly,

above that of our online plan modification approach.

- In the average case, our online plan modification approach required less computation time than replanning from scratch, when either adding or removing goals.

- Our plan modification approach was highly consistent compared to replanning with Metric-FF, displaying significantly lower variability in required computation time in all tests.

- When adding goals, our system was able to find a plan in 84.6% of cases which did not require a stitching plan and 61.5% of those which did. Replanning achieved 62.1% and 98.2% respectively. However, replanning from scratch was, on average, 48.8 seconds slower when a stitching plan was not required, and 2.83 seconds slower when it was. Considering only successful runs which required a stitching plan, replanning was an average of 0.95 seconds faster. The extra time required by our plan modification approach in this case is due to the generation of the stitching plan, which first required the construction of a new PDDL [49] problem file.

  The performance of our online plan modification approach was very consistent, with a standard deviation of only 0.01 seconds, compared to 56.45 when replanning (owing to replanning reaching the two minute timeout).

- When removing goals, our online plan modification approach found a plan in 46.2% of cases, compared to 85.2% for replanning. However, replanning from scratch was, on average, 18.31 seconds slower than modifying the plan to remove the goal (again, owing to replanning reaching the timeout) and 0.13 seconds slower when comparing only successful runs. The standard deviation of our online plan modification approach (0.003 seconds) was considerably lower than for replanning (43.59 seconds).

11.3. Experiment 2: Determining the plan modification criteria

A second experiment was performed to evaluate the plan modification criteria of the system, i.e. defining when the system should consider the options at a decision point and when it should instead continue the execution of the current plan. We compared our approach, described in Section 8 and formalised in Algorithm 1, to an intuitive alternative approach, referred to as *ObservedVsExpected*. Whilst the modification criteria described in this paper (referred to as our *primary* approach) triggers the additional and removal of goals in response to resource availability and the probability of successfully completing the plan, *ObservedVsExpected* instead compares the resource usage observed during plan execution to the expected usage over the same interval. The differences between the two approaches are summarised below:

When to consider removing a goal:

- Our *primary* approach — Consider removing a goal if the probability of successfully completing the plan, $P(success)$, for either battery or memory availability, falls below the threshold of 0.841, i.e. $P(success|battery) < 0.841$ or $P(success|memory) < 0.841$.

- ObservedVsExpected — Consider removing a goal if the observed resource usage of either battery or memory exceeds the mean plus the standard deviation of the combined resource usage distribution since the start of the plan or the most recent plan modification, i.e. $Usage(battery) > \mu_{battery} + \sigma_{battery}$ or $Usage(memory) > \mu_{memory} + \sigma_{memory}$.

When to consider adding a goal:

- Our *primary* approach — Consider adding a goal if the current battery availability exceeds the mean plus one standard deviation of the expected battery usage of the remainder of the current plan, plus the mean expected usage of a given plan fragment (pf), i.e. $AvailableBattery > \mu_{battery} + \sigma_{battery} + \mu_{pf}$.

- ObservedVsExpected — Consider adding a goal if the observed resource usage of either battery or memory is less than the mean expected usage, minus one standard deviation (again, since the start of the plan or the most recent plan modification), i.e. $Usage(battery) < \mu_{battery} - \sigma_{battery}$ or $Usage(memory) < \mu_{memory} - \sigma_{memory}$.

In this experiment we recorded the success rate and reward achieved using the same set of four test problems as in experiment one. Following the results of the first experiment (see Section 11.2), we fixed the number of decision points at 100%. By defining decision points after every action in the initial plan, we remove a variable from this second experiment. As in the first experiment, we again used three resource levels: *high* (H), *medium* (M) and *low* (L), as defined in Section 11. To facilitate a fair comparison across all experiments, we also used the same sets of fixed resource usages as in the previous experiment and repeated the experiment 50 times.

11.3.1. Analysis of results

As in Experiment 1, we evaluated the success rates separately from achieved rewards. Prior to calculating the average achieved reward, any failed runs and their corresponding pairs were again removed, as explained in Section 11.2.1. The failed runs are instead used to determine the success rate.

11.3.2. Success rates

The average success rates achieved by both our *primary* approach and *ObservedVsExpected* for problems with *low*, *medium* and *high* resource availability are shown in Figure 7. When resource availability was *low*, our *primary* approach achieved a success rate of 98%, outperforming *ObservedVsExpected* by 5.5 percentage points. Our *primary* approach also achieved a significantly

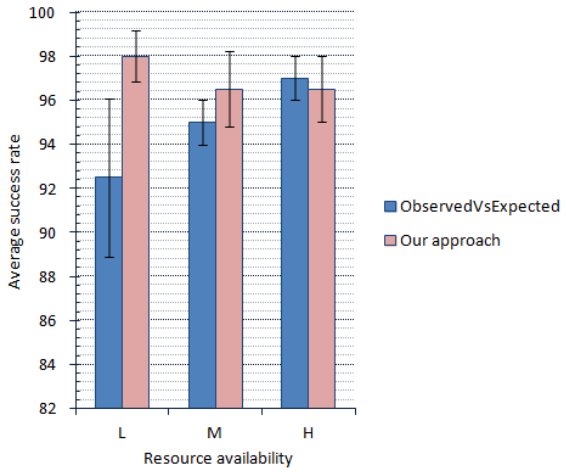

Figure 7: Graph showing the average success rates (and associated standard deviations) achieved by both approaches, given *low*, *medium* and *high* resource availability.

higher average success rate in the *medium* case (96.5%). In the case of *high* resource availability, *ObservedVsExpected* appears to slightly outperform our *primary* approach, achieving an average success rate of 97% compared to 96.5% for our *primary* approach, but this difference was not found to be statistically significant.

Over all problems and resource levels, our *primary* approach achieved a statistically significant 2.2% increase in success rate over *ObservedVsExpected* (using the chi-squared test [46]).

11.3.3. Average reward

The average rewards achieved by both approaches for varying levels of resource availability are shown in Figure 8. When resources were *low*, both approaches achieved a very similar average reward (the difference between them was not found to be statistically significant). However, as resource availability increased, the amount by which our *primary* approach achieved higher rewards than *ObservedVsExpected* also increased. In both the *medium* and *high* cases, the difference between the performance of the two approaches was found to be statistically significant.

Over all problems, our *primary* approach achieved higher rewards than *ObservedVsExpected* by a statistically significant margin of 10.5%.

11.3.4. Discussion

In the case of *low* constrained resource availability, the average rewards achieved by both approaches were very similar, but there were significant differences in success rate, where our *primary* approach outperformed *ObservedVsExpected* by 5.5 percentage points. This means that when compared to *Ob-*

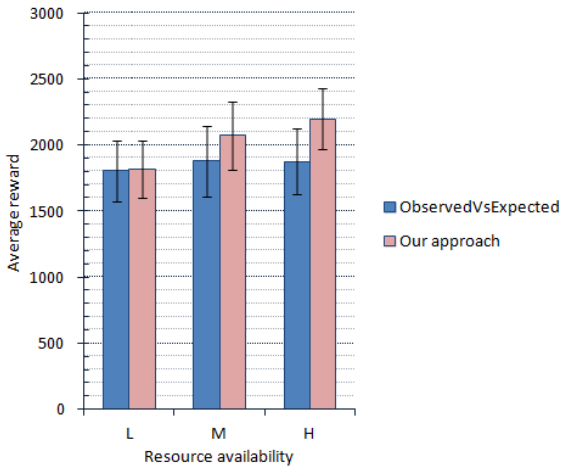

Figure 8: Graph showing the average rewards (and associated standard deviations) achieved by both approaches, given *low*, *medium* and *high* resource availability.

servedVsExpected, our *primary* approach was able to achieve a high success rate without compromising average reward. In the cases of *medium* and *high* resource availability, our *primary* approach was then able to achieve increasingly higher average rewards without sacrificing success rate. This suggests that our plan modification criteria are appropriately balancing the trade-off between capitalising on additional rewards and minimising risk to the mission and vehicle.

In Figure 9, we show the average number of goal additions and removals made by each approach given varying levels of resource availability. On average, our *primary* approach made more goal additions than *ObservedVsExpected* and fewer removals per run. This is especially noticeable in the case of *high* resource availability where, despite making a similar number of additions to our *primary* approach, *ObservedVsExpected* also removed a very high number of goals. On average, across all levels of resource availability *ObservedVsExpected* removed nearly as many goals as it added, compared to our *primary* approach which made many more additions. As we are only considering three resource levels, and the initial goal sets (and thus the options for immediate removal and addition) are individual to each problem, we did not expect to find a clear trend between the number of additions/removals and resource availability. However, we did expect the number of removals to be higher when resources are constrained (i.e. in the *low* case) than when they are plentiful (i.e. *high*). This was found to be true for our *primary* approach, but the number of goals removed by *ObservedVsExpected* was found to increase with resource availability. As both approaches used the same sets of sampled resource usages and rewards, and thus experienced the same states at each decision point, this would suggest that the *ObservedVsExpected* plan modification conditions resulted in poor decisions,

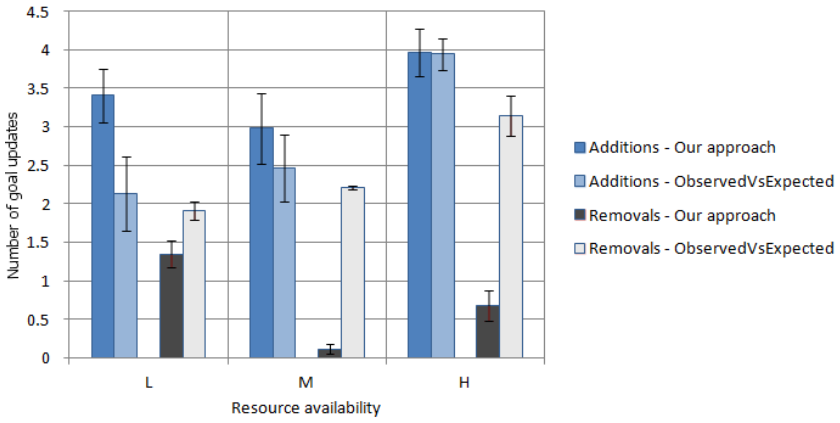

Figure 9: Graph showing the average number of goal additions and removals made per run by each approach, given *low*, *medium* and *high* resource availability. Error bars show the associated standard deviations.

such as potentially re-adding goals it had previously removed and vice versa. This limited the extent to which *ObservedVsExpected* was able to capitalise on opportunities for additional reward (e.g. in the *medium* and *high* cases) and compromised success rate when resources were *low*.

11.4. Experiment 3: Evaluating sequential plan modification and goal selection against an over-subscription planner

We then compared the performance of our goal selection and plan merging algorithm to a state-of-the-art over-subscription planner, OPTIC [4]. Whilst the results from our previous experiments showed that our online plan modification and execution monitoring approach offers improvements over both straight-line plans and online planning within our AUV domain [48], it was important to also evaluate the goal selection choices made by our online plan modification approach and the quality of the resulting plans against existing methods. As an over-subscription planner, OPTIC does not require a fixed goal set, instead allowing optional 'soft' goals to be expressed as 'preferences' in the PDDL problem encoding. While completion of these goals is optional, doing so may allow a plan to earn greater rewards. OPTIC reasons about the cost and reward of each 'soft' goal, searching for a plan which best satisfies a given metric, given resource constraints. In these experiments, we gave OPTIC a metric to maximise reward.

In the interests of a fair test, we nullified the benefits of online modification by removing resource usage uncertainty, instead assuming that all actions use exactly the mean plus one standard deviation of their resource usage distributions. By making such assumptions we ensured that plans generated by OPTIC were executed in exactly the way the planner anticipated and were not disadvantaged at run-time.

When using a plan repair-based approach, such as ours, the quality of the eventual solution is inevitably linked to the quality of the initial plan. As the aim of this experiment was to evaluate the performance of our online goal selection and plan modification algorithms, rather than the selection of an initial goal set (as this is assumed to be provided by an expert user), it was important that the performance of our system should not be unduly biased by the choice of initial goal set as this forms the base structure to which all future modifications are made. For example, if we defined all goals in an *overall problem* (i.e. initial goals and those which may be added if resources allow) as preferences, OPTIC may select a goal set which bears no resemblance to that used by our system when generating the initial plan, making direct comparison of the two approaches difficult. Instead, we defined the goals which comprise our system's initial goal set as 'hard' (non-optional) goals, whilst only those which may be added during execution were deemed preferences. To this end, OPTIC must generate a plan and choose a goal set that satisfies *at least* the initial goal set, adding more goals where and when possible. To allow direct comparison, we prevented our system from removing goals, thus preserving the concept of a 'hard' initial goal set.

As resource usage was assumed fixed and known for each problem, we invoked our algorithm at a single decision point prior to the execution of the first action. We recorded the mean battery usage and mean reward of the resulting plan along with the time taken by both approaches to find a solution. As OPTIC is an anytime planner, we recorded the best plan found within a timeout of one hour. The computation time recorded is therefore the time taken to find the best plan. Finally, as OPTIC does not support negative preconditions, it was necessary to make minor adjustments to our AUV domain, replacing the negative preconditions with new predicates. For example, the precondition $\neg$*on_surface(auv)* was replaced by *not_on_surface(auv)*.

The methodology of this experiment is not without drawbacks, as compromises, discussed above, had to be made in order to permit direct comparison of the two approaches. However, as we have yet to discover an alternative approach in the literature capable of solving the problems as presented to our system, determining suitable and fair compromises so as to evaluate the performance of our online plan modification approach against existing methods was essential. Whilst not optimal, as a deliberative over-subscription planner OPTIC provides the closest available approximation.

11.4.1. Analysis of results

OPTIC was unable to solve problems in the previously used test set and so we instead defined a set of smaller problems for use in this experiment. The topology of survey locations in the new problem set equates to half the number of locations used in the other experiments in this paper. Each of the five problems in the new test set comprised five initial goals plus five goals which may be added if resource availability is sufficient, so as to maximise eventual reward.

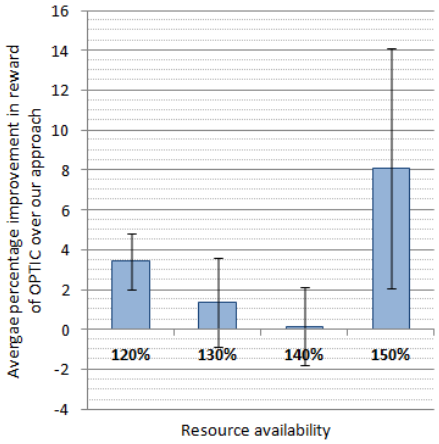

Figure 10: Graph showing the average percentage improvement in reward (and associated standard deviations) of plans produced by OPTIC over those produced by our online plan modification approach, given a range of resource availability.

11.4.2. Average reward

Using the new reduced test set, our system was able to find a valid plan in 100% of cases, while OPTIC found plans in 90%, failing on two problems where resource availability was relatively low (120%). Across all problems, OPTIC improved on the reward achieved by our system by 3.26% on average, as shown in Figure 10, a difference found for be statistically significant at the 5% level.

11.4.3. Plan quality

When attempting to judge the quality of a plan, we also must consider the expected resource usage of the plan. As the memory usage of a plan is not very informative as it is reusable, we solely consider resource usage in terms of battery.

Whilst the standard deviations were large across all cases, plans produced by OPTIC used, on average, 2.59% less battery than those produced by our online plan modification approach, a difference found for be statistically significant at the 5% level. Consequently, the reward achieved by OPTIC is slightly higher than that of our system on average (OPTIC achieving 3.26% higher rewards) whilst the battery used by OPTIC was lower by a comparable amount.

In summary, in comparison to OPTIC, our approach achieved lower rewards and used more battery to do so. If we divide the reward of each plan by its mean battery requirement to calculate the amount of reward achieved per unit of battery, we find a trend across all problems, shown in Figure 11. The reward per unit battery decreases as the surplus resources increase. This would suggest that both systems prioritise the 'best value' additional goals — first adding those whose cost to reward ratio is most beneficial, before choosing to add goals which represent progressively less good trade-offs as the surplus increases. It is

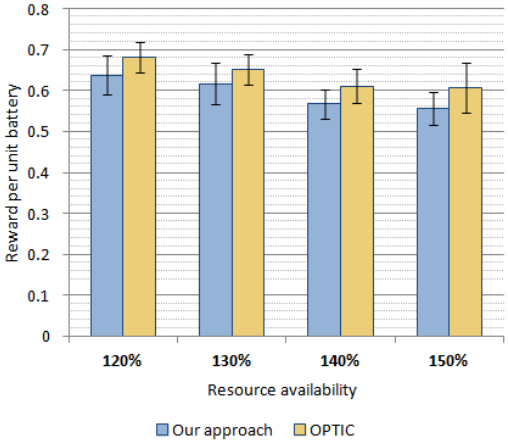

Figure 11: Graph showing the average reward per unit battery (and associated standard deviations) of plans produced using both our approach and OPTIC, for varying levels of resource availability.

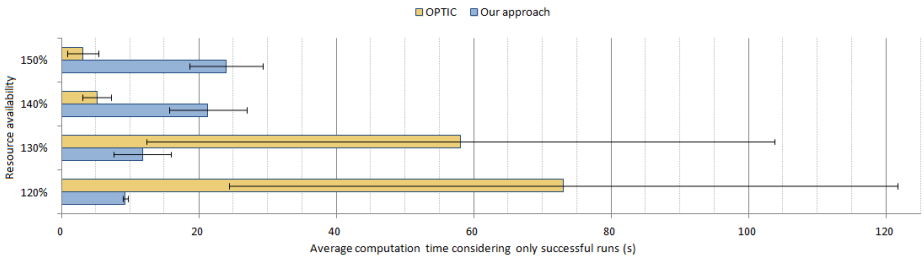

Figure 12: Graph showing the average computation time in seconds (and associated standard deviations) required by both our approach and OPTIC, considering only successful runs (i.e. ignoring timeouts).

very encouraging to see that, despite our system performing greedy plan repair, it achieves an average 93.2% of the reward per unit battery achieved by OPTIC which performs a more comprehensive search. Whilst never outperforming OPTIC, this result, and its consistency across all levels of resource availability, shows that our greedy system produces plans and associated goal sets whose value closely approximates that of those generated by a state-of-the-art oversubscription planner.

11.4.4. Computation time

Finally, we considered the computation time required by each approach. The average times of only the successful runs (i.e. ignoring the cases where OPTIC reached the timeout) are shown in Figure 12. When resources were very plentiful (i.e. in the cases of 140% and 150% resource availability), OPTIC was consistently fast, outperforming our approach by an average of 18.5 seconds.

However, when resource availability was lower (i.e. the 120% and 130% cases), OPTIC was 55 seconds slower on average, with a much higher variability. On average, our system requires 52.4% less time than OPTIC when considering only successful cases.

As our system is designed to capitalise on 'unexpected' additional resource availability, observed during the execution of a plan, we believe it is highly unlikely that surplus resources of 50% of the expected plan cost (i.e. as in our 150% availability test case) will arise in practice. This is both because an expert operator is likely to define a larger initial goal set, and because our system adds goal as soon as resources and success probability constraints will allow, without ever allowing large amounts of surplus resources to accumulate during execution. Our system is instead designed to make multiple minor adjustments to the goal set over the course of plan execution, in response to observed resource availability.

12. Application of our merge algorithm to existing domains

Given an over-subscribed goal set, our merge algorithm (presented in Section 10.1) is able to return valid plans for problems in many domains and thus is widely applicable. To illustrate this applicability, in this section we present example output from our merge algorithm when applied to two widely publicised International Planning Competition (IPC) domains — *Hiking* (see *IPC 2014 - domains*) and *Transport* (see *IPC 2014 - domains*). Both example domains are from the deterministic track of the competition, rather than either the specialist discrete probabilistic or continuous probabilistic tracks. This is because our plan merging algorithm does not consider resource usage and thus the deterministic domains may be used as is, without requiring any modification or discretisation. In all examples, Metric-FF [45] was used for both initial and sub-plan generation. After performing the merge algorithm, any redundant actions were removed, as in our full approach (see Section 9.1). We decided against using the *Rovers* domain (as used by [13]), as it shares many characteristics with our AUV domain example, including independent goals, a single agent and few constraints on the order in which goals are completed. To enable discussion of the wider application of the merge algorithm, we instead trialled it on sufficiently different domains to investigate its performance on domains outside the subset it was designed to solve. By doing this, we aim to highlight the merge algorithm's strengths and weaknesses, identifying domain characteristics which suit our approach.

12.1. Hiking

The *Hiking* domain (authored by Lee McCluskey, see *IPC 2014*) scenario involves a couple walking a multi-day hiking route, walking one leg of the journey each day. The hikers may only walk in one direction and must complete each leg of the walk together. Prior to starting each leg of their walk, the hikers must have first set up a tent at the end of the day's leg, ready for their arrival. Tents may not be carried by the hikers but must instead be moved in one of at least

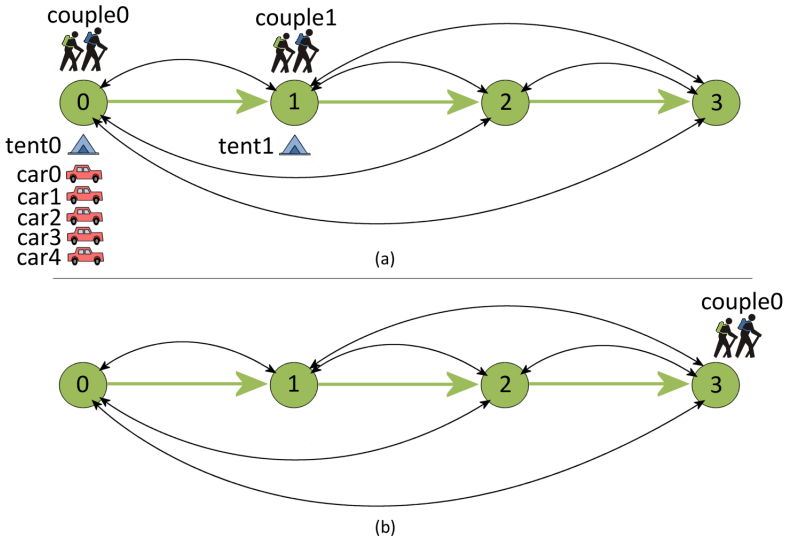

Figure 13: (a) Schematic illustrating the initial state of the initial plan in the *Hiking* domain. (b) Schematic illustrating the goal state of the initial plan. The location of cars, tents and *couple1* are not defined in the goal state. In both diagrams, green circles represent places, green arrows represent the hiking route and black arrows represent journeys which may be taken in a car.

two cars. Cars may be driven backwards and forwards by the hikers between any waypoints on the hiking route and may carry either one or two hikers at a time.

We chose *Hiking* as an example alternative domain because:

- It features independent goals.
- The domain scenario lends itself to being extended to include both resource usage and resource uncertainty, e.g. the cars could consume fuel or people could consume energy whilst walking (only needing to sleep in a tent to replenish their energy once it falls below a threshold).
- In contrast to our AUV domain, the *Hiking* domain is much more constrained: people may only walk as a couple and may only walk in one direction, whilst tents may only be moved using a car and must be set up at the next stop on the route before the hikers may walk to it. Dead-end states occur when the hikers are unable to set up a tent at the next location due to not having a car available at their current location.

12.1.1. Example output

Given the initial state and goals shown in Figure 13, Metric-FF generated the following plan:

o_1: `put_down(girl1, place1, tent1)`
o_2: `put_down(girl0, place0, tent0)`

o_3: drive(guy0, place0, place1, car4)
o_4: drive_tent(guy0, place1, place0, car4, tent1)
o_5: drive_tent_passenger(girl0, place0, place1, car4, tent0, guy0)
o_6: drive_passenger(girl0, place1, place0, car4, guy0)
o_7: put_up(girl1, place1, tent0)
o_8: drive_tent(guy0, place0, place1, car4, tent1)
o_9: drive_tent_passenger(guy0, place1, place0, car4, tent1, girl1)
o_{10}: walk_together(tent0, place1, guy0, place0, girl0, couple0)
o_{11}: drive_tent(girl1, place0, place2, car4, tent1)
o_{12}: put_up(girl1, place2, tent1)
o_{13}: walk_together(tent1, place2, guy0, place1, girl0, couple0)
o_{14}: put_down(girl0, place2, tent1)
o_{15}: drive_tent(girl1, place2, place3, car4, tent1)
o_{16}: put_up(girl1, place3, tent1)
o_{17}: walk_together(tent1, place3, guy0, place2, girl0, couple0)

However, while actions o_{10} onwards are straightforward, the plan is unnecessarily convoluted as the first nine actions result in swapping the locations of *tent0* and *tent1*, which there is no need for.The only thing which needs to happen prior to *couple0* walking from *place0* to *place1* (o_{10}) is that either *girl1* or *guy1* is moved to *place0*, ready to pick up a car to move a tent to future locations once *couple0* arrives at *place1*.

Prior to the execution of the first action in the initial plan, we added a new goal for *couple1* (*girl1* and *guy1*) to walk to *place3*. Metric-FF generated the following sub-plan:

pf_1: put_down(girl1, place1, tent1)
pf_2: put_down(guy0, place0, tent0)
pf_3: drive_tent_passenger(guy0, place0, place2, car4, tent0, girl0)
pf_4: put_up(guy0, place2, tent0)
pf_5: walk_together(tent0, place2, guy1, place1, girl1, couple1)
pf_6: put_down(girl1, place2, tent0)
pf_7: drive_tent(girl0, place2, place3, car4, tent0)
pf_8: put_up(girl0, place3, tent0)
pf_9: walk_together(tent0, place3, guy1, place2, girl1, couple1)

This time, *guy0* and *girl0* move *tent0* to each location along *couple1*'s route. It is pure coincidence that *tent0* was used in the sub-plan and *tent1* in the initial plan. The sub-plan is generated using only the state at the branch point and a single goal — the external planner (in this case, Metric-FF) has no knowledge of the causal structure of the initial plan when generating the sub-plan.

A single plan was returned by our merge algorithm which required a stitching plan (actions annotated s_n). N.B. '...' indicates where contiguous sections of the initial and sub-plans have been removed from this write-up for the sake of brevity:

- - - current state - - -
pf_1: put_down(girl1, place1, tent1)
...
pf_9: walk_together(tent0, place3, guy1, place2, girl1, couple1)

s_1: drive(guy1, place3, place2, car4)
s_2: drive(guy0, place2, place3, car4)
s_3: put_down(guy0, place3, tent0)
s_4: drive_tent_passenger(guy0, place3, place1, car4, tent0, girl1)
s_5: drive_tent(guy0, place1, place3, car4, tent0)
s_6: drive_tent_passenger(guy0, place3, place0, car4, tent0, girl0)
✂ s_7: put_up(girl1, place1, tent1) ✂
✂ s_8: put_up(guy0, place0, tent0) ✂
✂ o_1: put_down(girl1, place1, tent1) ✂
✂ o_2: put_down(girl0, place0, tent0) ✂
o_3: drive(guy0, place0, place1, car4)
o_4: drive_tent(guy0, place1, place0, car4, tent1)
...
o_{17}: walk_together(tent1, place3, guy0, place2, girl0, couple0)

The stitching plan resolves threats to causal relationships in the initial plan by returning *guy0*, *girl0* and *tent0* back to *place0*, and *girl1* from *place3* to *place1*. Four actions were deemed redundant and removed, indicated by the red ✂...✂ symbols.

Whilst our merge algorithm was able to return a valid plan, this example illustrates the importance of a high quality initial plan. The convoluted action sequence from the initial plan is preserved in the resulting plan as the merge algorithm modifies the existing plan, maintaining causal structure, rather than throwing away the existing plan and starting from scratch, as in replanning approaches (see Section 4.2).

12.2. Transport

The *Transport* domain (see *IPC 2014 - domains*) scenario is that of multiple trucks collecting packages from various locations in a graph and delivering them to associated goal locations. Each truck has a limited capacity for carrying packages and may only collect packages while it still has space for them. Delivering packages creates space within the truck. Moving between locations has an action cost, as do collecting and delivering packages. Each problem in the domain has an associated optimisation metric requiring a planner to minimise the total cost of the plan.

We chose *Transport* as an alternative domain because:

- The goals are relatively independent.
- The *Transport* scenario may be logically extended to include resource uncertainty and rewards, making it a suitable candidate for use with our full approach.
- It shares characteristics with our AUV domain, including: bi-directional edges between locations, action costs, no constraint by logical preconditions on the order in which goals are completed and both domains seek to optimise a metric.

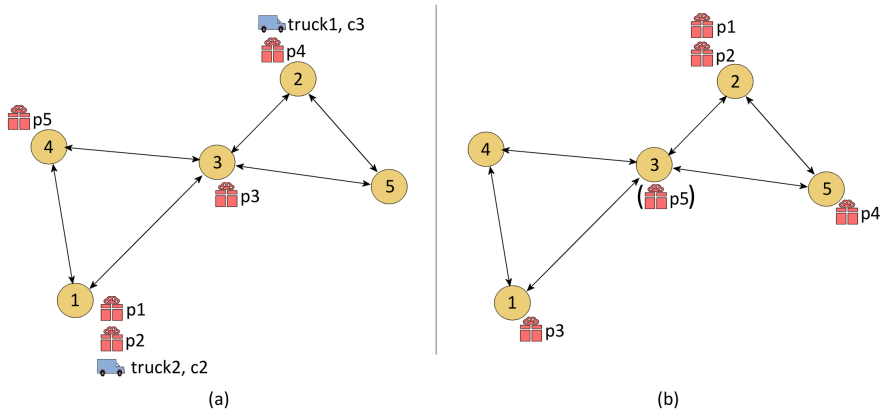

Figure 14: (a) Schematic illustrating the initial state of the initial plan in the *Transport* domain. (b) Schematic illustrating the goal state of the initial plan (the additional goal, to move $p5$ is shown in brackets). The location of trucks are not defined in the goal state. In both diagrams, yellow circles represent locations and arrows represent roads between locations. The initial and goal locations of each package p are shown and trucks are annotated with their initial capacity, c.

- It is suitably different from our AUV domain, in that there are multiple trucks as opposed to a single AUV and the completion of each goal is tied to a specific location.

12.2.1. Example output

Given the initial state and four delivery goals to complete with two trucks, as illustrated in Figure 14, Metric-FF generated the following initial plan (`p` denotes packages, `loc` locations and `c` values which represent a measure of the current and previous capacity of the truck) — note the redundancy between actions o_8 and o_9:

o_1: `drive(truck2, loc1, loc3)`
o_2: `pick-up(truck1, loc2, p4, c2, c3)`
o_3: `drive(truck1, loc2, loc5)`
o_4: `drop(truck1, loc5, p4, c2, c3)`
o_5: `pick-up(truck2, loc3, p3, c1, c2)`
o_6: `drive(truck2, loc3, loc1)`
o_7: `drop(truck2, loc1, p3, c1, c2)`
o_8: `drive(truck2, loc1, loc3)`
o_9: `drive(truck2, loc3, loc1)`
o_{10}: `pick-up(truck2, loc1, p1, c1, c2)`
o_{11}: `pick-up(truck2, loc1, p2, c0, c1)`
o_{12}: `drive(truck2, loc1, loc3)`
o_{13}: `drive(truck2, loc3, loc2)`
o_{14}: `drop(truck2, loc2, p1, c0, c1)`
o_{15}: `drop(truck2, loc2, p2, c1, c2)`

Prior to the execution of the first action, we added a new goal to collect package five from location four and deliver it to location three. Metric-FF generated the following sub-plan:

pf_1: drive(truck2, loc1, loc4)
pf_2: pick-up(truck2, loc4, p5, c1, c2)
pf_3: drive(truck2, loc4, loc3)
pf_4: drop(truck2, loc3, p5, c1, c2)

Two plans were returned by the merge algorithm, both requiring the generation of an additional stitching plan that, along with an action from the initial plan, was later removed as redundant. In the first ordering, shown below, the sub-plan was added as a single contiguous block prior to the initial plan:

\- - - current state - - -
pf_1: drive(truck2, loc1, loc4)
pf_2: pick-up(truck2, loc4, p5, c1, c2)
pf_3: drive(truck2, loc4, loc3)
pf_4: drop(truck2, loc3, p5, c1, c2)
✂ s_1: drive(truck2, loc3, loc1) ✂
✂ o_1: drive(truck2, loc1, loc3) ✂
o_2: pick-up(truck1, loc2, p4, c2, c3)
o_3: drive(truck1, loc2, loc5)
o_4: drop(truck1, loc5, p4, c2, c3)
o_5: pick-up(truck2, loc3, p3, c1, c2)
o_6: drive(truck2, loc3, loc1)
o_7: drop(truck2, loc1, p3, c1, c2)
✂ o_8: drive(truck2, loc1, loc3) ✂
✂ o_9: drive(truck2, loc3, loc1) ✂
o_{10}: pick-up(truck2, loc1, p1, c1, c2)
o_{11}: pick-up(truck2, loc1, p2, c0, c1)
o_{12}: drive(truck2, loc1, loc3)
o_{13}: drive(truck2, loc3, loc2)
o_{14}: drop(truck2, loc2, p1, c0, c1)
o_{15}: drop(truck2, loc2, p2, c1, c2)

As the sub-plan resulted in *truck2* moving from *loc1* to *loc3*, the first action in the initial plan, o_1 is not strictly necessary to meet the preconditions of o_2. However, as the merge algorithm may only skip actions in the sub-plan, it was not possible to simply skip o_1 and so the generation of a stitching plan was necessary to resolve the threat to o_1, which required *truck2* at *loc1*. After the merge was complete, redundant actions were removed between duplicated states (see Section 9.1). As the state preceding s_1 is identical to that following o_1, both actions were pruned from the plan (indicated by the red ✂...✂ symbols).

Whilst the resulting plan is valid and no longer contains redundant actions, the steps taken by the merge algorithm probably seem unnecessarily convoluted to the reader — why could the algorithm not just skip o_1 in the first place, preventing the need for a stitching plan? However, while it is straightforward to detect redundancy in a completed merge, knowing which action to skip in the current plan mid-merge is not so trivial as the causal structure may be

complicated with many interdependencies.

In the second ordering, shown below, the sub-plan was added halfway through the initial plan and again, a stitching plan was first generated and then pruned along with an action from the initial plan, o_8:

- - - current state - - -
o_1: drive(truck2, loc1, loc3)
o_2: pick-up(truck1, loc2, p4, c2, c3)
o_3: drive(truck1, loc2, loc5)
o_4: drop(truck1, loc5, p4, c2, c3)
o_5: pick-up(truck2, loc3, p3, c1, c2)
o_6: drive(truck2, loc3, loc1)
o_7: drop(truck2, loc1, p3, c1, c2)
pf_1: drive(truck2, loc1, loc4)
pf_2: pick-up(truck2, loc4, p5, c1, c2)
pf_3: drive(truck2, loc4, loc3)
pf_4: drop(truck2, loc3, p5, c1, c2)
✂ s_1: drive(truck2, loc3, loc1) ≻
✂ o_8: drive(truck2, loc1, loc3) ≻
o_9: drive(truck2, loc3, loc1)
o_{10}: pick-up(truck2, loc1, p1, c1, c2)
o_{11}: pick-up(truck2, loc1, p2, c0, c1)
o_{12}: drive(truck2, loc1, loc3)
o_{13}: drive(truck2, loc3, loc2)
o_{14}: drop(truck2, loc2, p1, c0, c1)
o_{15}: drop(truck2, loc2, p2, c1, c2)

As the inclusion of the sub-plan met the preconditions of o_9 and all subsequent actions, o_8 was no longer required. However, as before, o_8 was only able to be removed as a redundant action following the completion of the merge.

If we instead choose to include the new goal following the execution of the first action in the initial plan, o_1, the causal structure is such that we avoid this complexity entirely. In this case, the sub plan generated by Metric-FF is as follows:

pf_1: drive(truck2, loc3, loc4)
pf_2: pick-up(truck2, loc4, p5, c1, c2)
pf_3: drive(truck2, loc4, loc3)
pf_4: drop(truck2, loc3, p5, c1, c2)

The merge algorithm is able to merge the sub-plan, without the need for a stitching plan. In total, 36 unique orderings were found when merging this second sub-plan into the initial plan. While this may sound computationally expensive, the merge algorithm required a total of 1.82 seconds to generate all 36 orderings.

As sub-plans are generated to achieve a single goal, situations may occur in which the addition of a new goal increases the amount of actions/cost required to achieve an existing goal. For example, if a new delivery goal is added when all trucks are full, a sub-plan may specify that a package is temporarily dropped off at a location which is not its goal to make space in the truck to enable

the collection and delivery of the package associated with the newly added goal. However, while this may seem to be a disadvantage when the ordering is considered in isolation, within the context of our wider system the goal will only be added if the resulting plan is better (according to the optimisation metric) than the current plan. If there was no uncertainty in the domain, and thus no need for online modification, given sufficiently high resources we would expect an over-subscription planner such as OPTIC to include package five from the start, producing a higher quality plan which takes advantage of cross-over/helpful interactions between the goals.

13. Conclusions

In this paper, we have presented a novel online plan modification and execution monitoring approach to planning under uncertainty in resource-constrained environments.

We defined the key characteristics of realistic domains, common to many robotics applications, that suit the use of our approach (uncertainty, over-subscribed goals and optimisation metrics). Uncertainty over continuous state variables such as resources, coupled with a significantly large number of possible goal combinations, prevents the use of existing offline methods for planning under uncertainty, such as Markov decision problem solvers and contingency plans. By using an online approach, we are able to reason about goals and reduce the risk and uncertainty associated with the rest of a plan by using observations made during plan execution. We presented our online plan modification approach which sequentially modifies a plan during execution using pre-computed plan fragments, each comprising the actions required to complete a single goal along with the associated resource costs and causal structure.

We evaluated our full approach using our AUV domain and the results showed that altering the plan during execution allowed surplus resources to be used for exploiting additional opportunities, increasing the overall reward without adversely affecting the success rate. Conversely, when resources were constrained, our algorithm was able to increase success rate by removing low-value goals and their associated actions. This is especially important when considering robot operations in hazardous environments, where human intervention is very limited or impossible, as data collection capabilities may be fully utilised without compromising the safe return of the robot and its data cargo. We then compared the goal selection and plan merging capabilities of our greedy online modification algorithm to plans produced by a state-of-the-art over-subscription planner, finding that our approach was able to closely approximate the quality produced by the more comprehensive approach. When considering the computational cost of our approach, in the average case our online modification algorithm was found to require less computation time than both the over-subscription planner and replanning from scratch [48].

Whilst implementing our online plan modification approach on a real robot was outside the scope of this work, the results from testing our approach in simulation are encouraging and confirm it has promise as an effective solution

to planning for a large class of oversubscribed planning domains which feature significant resource uncertainty.

Acknowledgements

This work was funded as a PhD studentship at the University of Birmingham, School of Computer Science.

Appendix A. MDP representation of AUV domain

An MDP is a tuple $\langle S, A, T, R \rangle$ where S is the set of states, A the set of actions the agent can perform, T the transition function and R the reward function. As S is infinite due to the continuous variables representing resources, T is also infinite. Consequently, without first discretising the continuous variables, the use of a standard MDP solver is infeasible. As a result, we do not define T, instead using this representation as a convenient formalism. The AUV domain is defined as follows:

- A complete system state $s \in S$ defined by the following set of *state variables*:
  - Location of the vehicle:
    $at\_loc(l)$ where $l \in Locations$ and $Locations = \{\text{start}, \text{end}, l_1, l_2, \ldots\}$.
  - Whether the vehicle is on the surface or at operating depth:
    $on\_surface() = \{\text{true}, \text{false}\}$.
  - Datasets which have been collected:
    $data\_collected(d) = \{\text{true}, \text{false}\}$ where $d \in Datasets$ and $Datasets = \{d_1, d_2, \ldots\}$.
  - Datasets which have been transmitted:
    $data\_transmitted(d) = \{\text{true}, \text{false}\}$ where $d \in Datasets$ and $Datasets = \{d_1, d_2, \ldots\}$.
  - Datasets which have been delivered at the point of vehicle recovery:
    $data\_delivered(d) = \{\text{true}, \text{false}\}$ where $d \in Datasets$ and $Datasets = \{d_1, d_2, \ldots\}$.
  - Datasets with the scientists — true if dataset has either been transmitted or delivered:
    $data\_with\_scientists(d) = data\_transmitted(d) \vee data\_delivered(d)$.
  - The size of each on-board dataset — zero for uncollected sets, unknown at planning time and thus is assumed to be the mean of the associated distribution:
    $data\_size(d) = (positive\ real)$ where $d \in Datasets$ and $Datasets = \{d_1, d_2, \ldots\}$.
  - Status of the mission:
    $mission\_ended() = \{\text{true}, \text{false}\}$.

- Available battery power: $battery = (positive\ real)$.
- Available memory: $memory = (positive\ real)$.
- Achieved reward: $reward = (positive\ real)$.

To represent the world in which the vehicle operates, the following constants are also included in the state representation:

- Traversable graph of locations, represented as neighbouring pairs: $is\_neighbour(l_i, l_j) = \{\text{true}, \text{false}\}$
- The predefined end-location, at which the vehicle is to await recovery by a support vessel: $is\_end\_location(l)$
- The location of each collectable dataset: $data\_to\_collect(l, d) = \{\text{true}, \text{false}\}$
- The mean battery usage when moving between two locations: $move\_battery\_usage(l_i, l_j) = (positive\ real)$.
- The mean battery usage when collecting a dataset: $collect\_battery\_usage(d) = (positive\ real)$.
- The standard deviation of the battery usage distribution when collecting a dataset: $sd\_collect\_battery\_usage(d) = (positive\ real)$.
- The mean memory usage when collecting a dataset: $mean\_memory\_usage(d) = (positive\ real)$.
- The standard deviation of the memory usage distribution when collecting a dataset: $sd\_memory\_usage(d) = (positive\ real)$.
- The mean reward associated with each dataset, awarded upon $data\_with\_scientists(d) = true$: $mean\_data\_reward(d) = (positive\ real)$.
- The mean battery usage when surfacing: $surface\_battery\_usage() = (positive\ real)$.
- The standard deviation of the battery usage distribution when surfacing: $sd\_surface\_battery\_usage() = (positive\ real)$.
- The mean battery usage when diving: $dive\_battery\_usage() = (positive\ real)$.
- The standard deviation of the battery usage distribution when diving: $sd\_dive\_battery\_usage() = (positive\ real)$.
- The mean battery usage when transmitting a dataset: $transmit\_battery\_usage(d) = (positive\ real)$.
- The reward achieved for ending the mission at the pre-defined end location, $is\_end\_location(l)$: $reward\_end\_location() = (positive\ real)$.

- A, the set of discrete actions available to the vehicle is represented by seven parameterised action schemas:
  - *CollectData*(d) — Collect dataset d if vehicle is in the correct location.
  - *Move*(l_1, l_2) — If the vehicle is at location l_1, move to l_2.
  - *Dive*() — Move from the surface to depth at the current location.
  - *Surface*() — Move from depth to the surface at the current location.
  - *TransmitData*(d) — Attempt data transmission provided the vehicle is at the surface and has dataset d.
  - *DeliverData*(d) — Deliver dataset d provided the mission has ended and the vehicle has collected the dataset d.
  - *EndMission*(l) — Wait for recovery at current location l.
    Upon executing this action, only DeliverData actions may subsequently be performed. The vehicle receives a reward for finishing the mission. However, if l does not equal the predefined end location (*is_end_location*(l)) the vehicle does not receive the full reward. Recovering the vehicle from a different location is costly as it requires the support vessel to change route.
- R is the reward function $R(s, a, s')$ which specifies the immediate reward for performing action a in state s and transitioning to s'. Positive rewards are given when the vehicle successfully delivers a dataset and ends the mission, with additional reward available for ending the mission at the agreed location, defined by *is_end_location*(l). For example, successfully performing *TransmitData*(d) in a state where dataset d has been collected but not transmitted would result in a reward.

References

[1] M. Fox, D. Long, D. Magazzeni, Plan-based policies for efficient multiple battery load management, Journal of Artificial Intelligence Research 44 (2012) 335–382, aAAI Press (2012).

[2] S. Russell, E. Wefald, Principles of Metareasoning, Artificial Intelligence 49 (1–3) (1991) 361–395, elsevier (1991).

[3] M. Brito, N. Bose, R. Lewis, P. Alexander, G. Griffiths, J. Ferguson, The role of adaptive mission planning and control in persistent autonomous underwater vehicles presence, in: Proceedings of IEEE/OES Autonomous Underwater Vehicles, AUV'12, Institute of Electrical and Electronic Engineers, Southampton, UK, 2012 (September 2012).

[4] J. Benton, A. J. Coles, A. Coles, Temporal planning with preferences and time-dependent continuous costs, in: Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS'12, AAAI Press, São Paulo, Brazil, 2012 (June 2012).

[5] C. Mcgann, F. Py, K. Rajan, H. Thomas, R. Henthorn, R. McEwen, T-rex: A model-based architecture for AUV control, in: Workshop on Planning and Plan Execution for Real-World Systems, International Conference on Automated Planning and Scheduling, 2007 (01 2007).

[6] F. Py, J. Pinto, M. A. Silva, T. A. Johansen, J. Sousa, K. Rajan, Europtus: A mixed-initiative controller for multi-vehicle oceanographic field experiments, in: D. Kulić, Y. Nakamura, O. Khatib, G. Venture (Eds.), 2016 International Symposium on Experimental Robotics, Springer International Publishing, Cham, 2017, pp. 323–340 (2017).

[7] M. Pebody, The Contribution of Scripted Command Sequences and Low Level Control Behaviours to Autonomous Underwater Vehicle Control Systems and Their Impact on Reliability and Mission Success, in: Proceedings of OCEANS – Europe, Institute of Electrical and Electronic Engineers, Aberdeen, UK, 2007, pp. 1–5 (June 2007).

[8] M. Seto, L. Paull, S. Saeedi, Introduction to autonomy for marine robots, in: M. L. Seto (Ed.), Marine Robot Autonomy, Springer, 2013, pp. 1–46 (2013).

[9] C. R. German, D. R. Yoerger, M. Jakuba, T. M. Shank, C. H. Langmuir, K. Nakamura, Hydrothermal exploration with the Autonomous Benthic Explorer, Deep Sea Research Part I: Oceanographic Research Papers 55 (2) (2008) 203–219, elsevier (2008).

[10] S. Smith, M. Becker, An ontology for constructing scheduling systems, in: Proceedings of Working Notes of 1997 AAAI Symposium on Ontological Engineering, AAAI Press, 1997 (March 1997).

[11] R. Howard, Dynamic Programming and Markov Processes, Massachusetts Institute of Technology Press, Cambridge, MA, 1960 (1960).

[12] C. Harris, R. Dearden, Contingency planning for long-duration auv missions, in: Proceedings of IEEE/OES Autonomous Underwater Vehicles, AUV'12, Institute of Electrical and Electronic Engineers, Southampton, UK, 2012 (September 2012).

[13] J. Bresina, R. Dearden, N. Meuleau, S. Ramakrishnan, D. Smith, R. Washington, Planning under continuous time and resource uncertainty: A challenge for AI, in: Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence, UAI'02, Morgan Kaufmann, Alberta, Canada, 2002, pp. 77–84 (August 2002).

[14] S. Yoon, A. Fern, R. Givan, FF-Replan: A Baseline for Probabilistic Planning, in: Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling, ICAPS'07, AAAI Press, Providence, RI, 2007, pp. 352–378 (September 2007).

[15] F. W. Trevizan, F. Teichteil-Königsbuch, S. Thiébaux, Efficient solutions for stochastic shortest path problems with dead ends, in: G. Elidan, K. Kersting, A. T. Ihler (Eds.), Proceedings of the Thirty-Third Conference on Uncertainty in Artificial Intelligence, UAI 2017, Sydney, Australia, August 11-15, 2017, AUAI Press, 2017 (2017).

[16] D. Silver, J. Veness, Monte-carlo planning in large POMDPs, in: J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, A. Culotta (Eds.), Advances in Neural Information Processing Systems, Vol. 23, Curran Associates, Inc., 2010, pp. 2164–2172 (2010).

[17] G. Flaspohler, V. Preston, A. P. M. Michel, Y. Girdhar, N. Roy, Information-guided robotic maximum seek-and-sample in partially observable continuous environments, IEEE Robotics and Automation Letters 4 (4) (2019) 3782–3789 (2019).

[18] G. J. Sussman, A computational model of skill acquisition, Tech. rep., Massachusetts Institute of Technology, Cambridge, MA (1973).

[19] N. Tsiogkas, V. De Carolis, D. M. Lane, Energy-constrained informative routing for AUVs, in: OCEANS 2016 - Shanghai, 2016, pp. 1–5 (2016).

[20] Z. Feng, R. Dearden, N. Meuleau, R. Washington, Dynamic programming for structured continuous Markov decision problems, in: Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence, UAI'04, AUAI Press, Banff, Canada, 2004, pp. 154–161 (July 2004).

[21] Mausam, E. Benazera, R. Brafman, N. Meuleau, E. A. Hansen, Planning with continuous resources in stochastic domains, in: Proceedings of the 19th International Joint Conference on Artificial Intelligence, Vol. 19 of IJCAI'05, Morgan Kaufmann, Edinburgh, UK, 2005, pp. 1244–1251 (July 2005).

[22] J. Pearl, Heuristics: Intelligent Search Strategies for Computer Problem Solving, Addison-Wesley, 1985 (1985).

[23] J. Hoffmann, B. Nebel, The FF planning system: Fast plan generation through heuristic search, Journal of Artificial Intelligence Research 14 (1) (2001) 253–302, AAAI Press (May 2001).

[24] P. Patrón, D. Lane, Y. Petillot, Continuous mission plan adaptation for autonomous vehicles: balancing effort and reward, in: Workshop on Planning and Plan Execution for Real World Systems, 19th Int. Conference on Automated Planning & Scheduling, 2009, pp. 1–8 (01 2009).

[25] R. Dearden, N. Meuleau, S. Ramakrishnan, D. Smith, R. Washington, Incremental contingency planning, in: Proceedings of the Workshop on Planning under Uncertainty at ICAPS'03, AAAI Press, Trento, Italy, 2003, pp. 38–47 (June 2003).

[26] L. Pryor, G. Collins, Planning for contingencies: A decision-based approach, Journal of Artificial Intelligence Research 4 (1996) 287–339, aAAI Press (1996).

[27] D. Draper, S. Hanks, D. Weld, Probabilistic planning with information gathering and contingent execution, Tech. Rep. 93-12-04, University of Washington, Seattle, WA (1993).

[28] P. R. Conrad, J. A. Shah, B. C. Williams, Flexible execution of plans with choice, in: Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling, ICAPS'09, AAAI Press, Thessaloniki, Greece, 2009 (September 2009).

[29] J. Gough, M. Fox, D. Long, Plan execution under resource consumption uncertainty, in: Proceedings of the Workshop on Connecting Planning Theory with Practice at ICAPS'04, AAAI Press, Whistler, Canada, 2004, pp. 24–29 (June 2004).

[30] A. J. Coles, Opportunistic branched plans to maximise utility in the presence of resource uncertainty., in: Proceedings of the Twentieth European Conference on Artificial Intelligence, ECAI'12, IOS Press, Montpellier, France, 2012 (August 2012).

[31] B. Nebel, J. Koehler, Plan reuse versus plan generation: A theoretical and empirical analysis, Artificial Intelligence 76 (1-2) (1995) 427–454, elsevier (Jul. 1995).

[32] S. Hanks, D. S. Weld, A domain-independent algorithm for plan adaptation, Journal of Artificial Intelligence Research 2 (1995) 319–360, aAAI Press (1995).

[33] G. Boella, R. Damiano, A replanning algorithm for a reactive agent architecture, in: Proceedings of the 10th International Conference on Artificial Intelligence: Methodology, Systems, and Applications, AIMSA'02, Springer, Varna, Bulgaria, 2002, pp. 183–192 (September 2002).

[34] R. van der Krogt, M. de Weerdt, Plan repair as an extension of planning, in: Proceedings of the 15th International Conference on Automated Planning and Scheduling, ICAPS'05, AAAI Press, Monterey, California, 2005, pp. 161–170 (June 2005).

[35] W. Stephan, S. Biundo, Deduction-based refinement planning, in: Proceedings of the Third International Conference on Artificial Intelligence Planning Systems, AIPS'96, AAAI Press, Edinburgh, Scotland, 1996, pp. 213–220 (May 1996).

[36] S. Kambhampati, C. A. Knoblock, Q. Yang, Planning as refinement search: a unified framework for evaluating design tradeoffs in partial-order planning, Artificial Intelligence 76 (1-2) (1995) 167–238, elsevier (1995).

[37] S. Kambhampati, Refinement planning as a unifying framework for plan synthesis, AI Magazine 18 (2) (1997) 67–97, aAAI Press (1997).

[38] P. Patron, E. Miguelanez, Y. R. Petillot, D. M. Lane, J. Salvi, Adaptive mission plan diagnosis and repair for fault recovery in autonomous underwater vehicles, in: OCEANS 2008, 2008, pp. 1–9 (Sep. 2008).

[39] M. Fox, A. Gerevini, D. Long, I. Serina, Plan stability: Replanning versus plan repair, in: Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, ICAPS'06, AAAI Press, Cumbria, UK, 2006, pp. 212–221 (June 2006).

[40] A. Gerevini, I. Serina, LPG: A planner based on local search for planning graphs with action costs, in: Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, AIPS'02, AAAI Press, Toulouse, France, 2002, pp. 13–22 (April 2002).

[41] D. Long, M. Woods, A. Shaw, D. Pullan, D. Barnes, D. Price, On-board plan modification for opportunistic science, in: Proceedings of the IJCAI-09 Workshop on Artificial Intelligence in Space, AAAI Press, Pasadena, CA, 2009 (July 2009).

[42] M. Fox, D. Long, L. Baldwin, G. Wilson, M. Wood, D. Jameux, R. Aylett, On-board timeline validation and repair: a feasibility study, in: Proceedings of 5th International Workshop on Planning and Scheduling in Space, Space Telescope Science Institute, Baltimore, MD, 2006 (October 2006).

[43] J. L. Bresina, R. Washington, Robustness via run-time adaptation of contingent plans, Proceedings of the AAAI-2001 Spring Syposium: Robust Autonomy (2001) 24–30AAAI Press (March 2001).

[44] R. Washington, K. Golden, J. L. Bresina, Plan execution, monitoring, and adaptation for planetary rovers., Electronic Transactions on Artificial Intelligence 4 (A) (2000) 3–21, royal Swedish Academy of Sciences (2000).

[45] J. Hoffmann, The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables, Journal of Artificial Intelligence Research 20 (2003) 291–341, aAAI Press (2003).

[46] F. Caswell, Success in Statistics, 2nd Edition, Richard Clay Ltd., Bungay, UK, 1989 (1989).

[47] D. T. Roper, A. B. Phillips, C. A. Harris, G. Salavasidis, M. Pebody, R. Templeton, S. V. S. Amma, M. Smart, S. McPhail, Autosub long range 1500: An ultra-endurance AUV with 6000 km range, in: OCEANS 2017 - Aberdeen, 2017, pp. 1–5 (June 2017).

[48] C. Harris, R. Dearden, Run-time plan repair for AUV missions, in: Proceedings of the 7th European Starting AI Researcher Symposium (STAIRS), ECAI'14, IOS Press, Prague, Czech Republic, 2014, pp. 151–160 (August 2014).

[49] M. Fox, D. Long, PDDL2.1: An extension to PDDL for expressing temporal planning domains, Journal of Artificial Intelligence Research 20 (1) (2003) 61–124, aAAI Press (Dec. 2003).

Algorithm 1 Online Plan Modification

Require: p—current plan, k—user-defined success threshold, DP—set of decision points within p, CL—set of causal links within p, G—set of all goals in the overall problem with associated plan fragments pf_g, G_p—set of goals met by p, s_g—goal state of p

1: **function** OnlinePlanModification($p, k, G_p, G, CL, s_g, DP, pf_g$)
2: **while** $|p| > 0$ **do**
3: $a \leftarrow$ pop next action from p
4: $s_c \leftarrow$ EXECUTE(a) ▷ Update current state s_c
5: **if** $s_c \in DP$ **then**
6: **while** $P(success|p) < k$ **do** ▷ (Sec. 7.1)
7: $CP \leftarrow \emptyset$ ▷ Set of candidate plans
8: **for all** $g \in G_p$ **do**
9: $cp \leftarrow$ removeGoal(p, g, CL, s_g) ▷ Remove goal to increase $P(success)$ above k
10: $s_{cp} \leftarrow$ Calculate states in cp
11: $CP \leftarrow \{CP,$ removeRedundantActions(cp,$s_{cp})\}$
12: **end for**
13: Prune CP to remove plans which violate any resource constraints ▷ (Sec. 9.2)
14: Prune CP to remove plans where $P(success|cp) < k$
15: Select $p_{new} \in CP$ which maximises optimisation metric score ▷ (Eq. 3)
16: $p \leftarrow p_{new}$
17: Update G_p, CL, s_g, $P(success|p)$ to reflect p_{new}
18: **end while**
19: $G_{rem} \leftarrow$ goals removed at this DP
20:
21: $CP \leftarrow \{p\}$
22: **repeat**
23: **for all** $g \in \{G \setminus \{G_p \cup G_{rem}\}\}$ **do** ▷ Add goals excluding those just removed
24: **if** (Mean resource usage of pf_g and p) < available in s_c **then** ▷ (Eq. 4)
25: $cp \leftarrow$ mergePlans($\emptyset, pf_g, p$)
26: **if** $cp = \emptyset$ **then** ▷ Merge was not possible
27: Generate stitching plan p_{stitch} ▷ (Sec. 10.1)
28: $pf_g \leftarrow$ Append p_{stitch} to pf_g
29: $cp \leftarrow$ mergePlans($\emptyset, pf_g, p$)
30: **end if**
31: **if** $cp =$ valid plan **then**
32: $s_{cp} \leftarrow$ Calculate states in cp
33: $CP \leftarrow \{CP,$ removeRedundantActions(cp,$s_{cp})\}$
34: **end if**
35: **end if**
36: **end for**
37:
38: Prune CP to remove plans which violate any resource constraints
39: Prune CP to remove plans where $P(success|cp) < k$ ▷ (Sec. 10.2)
40: Select $p_{new} \in CP$ which maximises optimisation metric score ▷ (Eq. 3)
41: $g \leftarrow$ goal added in p_{new}
42: $CP \leftarrow \{p_{new}\}$
43: **for all** $dp \in DP$ **do** ▷ Evaluate delaying addition of goal (Sec. 10.3)
44: **if** pf_g exists at dp **then**
45: $cp \leftarrow$ mergePlans($\emptyset, pf_g, p$)
46: **Repeat lines 26 to 35**
47: **end if**
48: **end for**
49: Select $p_{new} \in CP$ which maximises optimisation metric score ▷ (Eq. 3)
50: $p \leftarrow p_{new}$
51: Update G_p, CL, s_g, $P(success|p)$ to reflect p_{new}
52: **until** No more goals can be added
53: **end if**
54: **end while**
55: **end function**

Algorithm 2 Removing a goal.

Require: p—current plan, g—goal to remove, CL—set of causal links within p, $lastRemoved$—initially the goal state.

function REMOVEGOAL($p, g, CL, lastRemoved$)
  $gProducers \leftarrow \emptyset$
  for all $cl \in CL$ **do**
    if $cl.consumer = lastRemoved$ AND $cl.literal = g$ **then**
      $gProducers \leftarrow gProducers + cl$ ▷ cl produces the literal g, consumed by $lastRemoved$
    end if
  end for
  for all $c \in gProducers$ **do**
    if c has only one consumer **then** ▷ causal links may have multiple consumers
      for all actions $a \in p$ **do**
        if $a = c.producer$ **then**
          $p \leftarrow p - a$ ▷ c is only consumed by $lastRemoved$, so may be removed
        end if
      end for
      for all preconditions $pre \in a.preconditions$ **do**
        $p \leftarrow removeGoal(p, pre, cl, c.producer)$ ▷ recurse, considering each precondition of $c.producer$
      end for
    end if
  end for
  return p
end function

Algorithm 3 Removing redundant actions.

Require: p—current plan, S—set of all discrete states in p

function REMOVEREDUNDANTACTIONS(p, S)
  for all $s \in S$ **do**
    if $|s \cap S| > 1$ **then** ▷ s appears multiple times in S
      Remove actions between first and last occurrence of s in p
    end if
  end for
  Update S to reflect changes to p
  return p, S
end function

Algorithm 4 Merging a plan fragment with an existing plan.

Require: pf—plan fragment, p—current plan, Z—list of valid merges, initially $\emptyset$, filled during recursion.

function MERGEPLANS(Z, pf, p)
  $valid \leftarrow false$
  if p satisfies all goals **then**
    $Z \leftarrow Z + p$, **return** true ▷ p is a solution, backtrack
  else if $pf = \emptyset$ **then return** false ▷ p is not a solution, backtrack
  else
    $a \leftarrow$ first action in pf,
    $pf \leftarrow pf - a$
    $X \leftarrow$ all states in p that meet preconditions of a
    for all $x \in X$ **do**
      if inserting a at x causes no threats OR all threats are resolvable **then**
        $merge \leftarrow p$ with a inserted at x
        Update S, CL following the insertion of a
        $valid \leftarrow mergePlans(Z, pf, merge)$ ▷ recurse, considering the remainder of pf
      end if
    end for
    if $valid =$ **false** AND a achieves no goals **then**
      $pf \leftarrow pf - a$ ▷ skip a
      $valid \leftarrow mergePlans(Z, pf, merge)$ ▷ recurse, considering the remainder of pf
    else, **return** valid
    end if
  end if
end function