ONLINE PLAN MODIFICATION IN UNCERTAIN RESOURCE-CONSTRAINED ENVIRONMENTS

by

CATHERINE HARRIS

A thesis submitted to
University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
University of Birmingham
2015

Abstract

This thesis presents a novel approach to planning under uncertainty in resource constrained environments. Such environments feature in many real-world applications, including planetary rover and autonomous underwater vehicle (AUV) missions. Our focus is on long-duration AUV missions, in which a vehicle spends months at sea, with little or no opportunity for intervention. As the risk to the vehicle and cost of deployment are significant, it is important to fully utilise each mission, maximising data return without compromising vehicle safety. Planning within this domain is challenging because significant resource usage uncertainty prevents computation of an optimal strategy in advance.

We describe our novel method for online plan modification and execution monitoring, which augments an existing plan with pre-computed plan fragments in response to observed resource availability. Our modification algorithm uses causal structure to interleave actions, creating solutions 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 the mission decreases, by removing low-priority goals our system reduces the risk to the vehicle, increasing mission success rate. Conversely, when resource availability allows, by including additional goals our system increases reward without adversely affecting success rate.

To James, Mum, Dad and Jen,

Finishing this degree would not have been possible without your love, support and encouragement. Thank you so much. xxx

ACKNOWLEDGEMENTS

Firstly, I would like to thank my supervisors, Richard Dearden and Nick Hawes. Richard provided a great deal of expertise and enthusiasm when developing the ideas and algorithms in this thesis. I enjoyed and learnt a lot from our many hours of discussion. Nick has been an invaluable help to me during the final stages of this PhD. I'm very grateful for all his support, advice and unwavering positivity throughout the evaluation of this work and the writing-up process. Thank you also to Peter Hancox for keeping an eye on my progress and encouraging me.

I would like to thank past and present members of both the Autosub and Delphin AUV teams at the National Oceanography Centre and the University of Southampton. My time spent working with you was incredibly inspiring and so much fun. Thank you for introducing me to the world of AUVs and for providing the inspiration for this thesis.

Many thanks to Zeyn Saigol for the use of his GraphPlan implementation and PDDL parser, both of which were very valuable, especially during my preliminary research.

I'd also like to thank my wonderful friends in both the School of Computer Science and the Wayfarers. Thank you all for many memorable and fun opportunities, discussions and adventures throughout this PhD.

I owe a huge thank you to my boyfriend James, who has been incredibly understanding and supportive throughout the ups and downs of this degree. From your mathematical advice to knowing just when cake was needed, you've been a star. I look forward to spending lots of thesis-free time together from now on!

Last, but not least, I am immensely grateful to my family. Thank you for all your love, encouragement and guidance — and for spending many *many* hours helping me proof-read this thesis. I love you all lots and can't begin to thank you enough!

CONTENTS

LIST OF PUBLICATIONS ARISING FROM THIS THESIS

Catherine Harris and Richard Dearden. Contingency planning for long-duration AUV missions. In *Proceedings of IEEE/OES Autonomous Underwater Vehicles*, AUV'12, Southampton, UK, September 2012. Institute of Electrical and Electronic Engineers.

Catherine Harris and Richard Dearden. Online plan modification for autonomous underwater vehicle missions: Dissertation abstract. In *Doctoral Consortium at the 23rd International Conference on Automated Planning and Scheduling*, ICAPS'13, June 2013.

Catherine Harris and Richard Dearden. Run-time plan repair for AUV missions. In *Proceedings of the 7th European Starting AI Researcher Symposium (STAIRS)*, ECAI'14, pages 151–160, Prague, Czech Republic, August 2014. IOS Press.

CHAPTER 1

INTRODUCTION

1.1 Motivation

In recent years, the use of autonomous underwater vehicles (AUVs) has become 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 AUVs currently have very little onboard artificial intelligence. Instead, operators typically favour simple pre-scripted behaviours such as lawnmower surveys, which minimise the complexity of the mission and risk to the vehicle (Pebody, 2007; Seto et al., 2013). By analysing data from the vehicle between missions, AUV operators can then select areas of interest for investigation during subsequent deployments (German et al., 2008). 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, conventional mission formats are inevitably over-conservative, 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.

There is an increasing demand from both the scientific (German et al., 2012) and commercial (Gilmour et al., 2012) communities for persistent ('field-resident') and long-

Figure 1.1: The University of Southampton's Delphin AUV.

range AUVs, capable of travelling thousands of kilometres in a single deployment. The ongoing development of such systems has the potential to revolutionise the collection of oceanographic data. Mission proposals for long-range vehicles include: a complete transect of the Drake Passage (a trip distance of approximately 2500km, proposed using Autosub Long-Range (Furlong et al., 2012)), long-range exploration of the mid-ocean ridge (Devey et al., 2010) and monitoring of seafloor environments over the course of many months, with the potential of collecting data during rare events, such as benthic storms (Wynn et al., 2014). In order to fully capitalise on their increased capabilities, such systems will require much greater levels of autonomy, enabling them to operate effectively without the opportunity for human intervention. Recent work has investigated the implementation of AI solutions on board AUVs (including Fox et al. (2012b) and Ahmadzadeh et al. (2013)), however most of this work is in its early stages and has yet to be used on operational AUVs during science missions (as opposed to technical demonstrations and trials). Inspired by the opportunities that this latest generation of AUVs provide, in this thesis we consider the problem of high-level mission planning during long-duration missions.

1.2 Overview of approach

Long-duration missions can be thought of as over-subscribed planning problems, where finite amounts of battery power and data storage space limit the number and duration of tasks achievable by the vehicle. Tasks can be represented as goals with rewards indicating the value of their completion. In over-subscribed planning problems, the objective is therefore to select the set of goals which maximises reward given limited resource availability. During a mission, the exact amount of resources required to complete each task is uncertain and not known in advance. This is due to both the inherent uncertainty present during operations in the deep ocean, an extreme environment of which we have relatively little prior knowledge, and the measurement uncertainty associated with the resources themselves, such as charge across multiple batteries (Fox et al., 2012a).

The problem of planning for long-range AUV missions is part of a larger class of over-subscribed planning problems in uncertain resource-constrained environments which includes Mars rover operations and some logistics problems. In this thesis, we present a novel approach to solving such problems, focussing on the motivating example of an AUV domain. Our solution enables a vehicle (or other agent) to autonomously refine its mission plan during execution, adding goals if current resources 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 vehicle. An online approach allows us to reason about goals and reduce the uncertainty of future states using observations made during the execution of the plan. As the benefits of planning are often hard to quantify in advance (Russell and Wefald, 1991) and any resources used by the vehicle for planning are no longer available for performing tasks, rather than generating a new full plan during execution, we have developed an online 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 structure.

We use this knowledge of the causal relationships between actions to either interleave additional actions into the plan if resources are plentiful, 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 vehicle is more likely to resemble the initial plan agreed prior to the deployment of the vehicle, compared to when the existing plan is discarded, as in the case of total replanning. Investigating why the AUV community has yet to widely adopt adaptive mission planning, Brito et al. (2012) found uncertain vehicle behaviours to be the largest concern (39.7%) of expert AUV operators, which our approach addresses.

In this thesis, we evaluate our plan repair approach in terms of the success rate and average reward it achieves on a number of problems. We outline its suitability within a wider class of related planning problems and discuss options for future work. We review related work from the planning literature and discuss its applicability to our motivating problem.

1.3 Contributions

The key contributions of this thesis are:

- A new online plan modification and execution monitoring approach to planning under uncertainty in resource-constrained environments. When resources are limited, our approach is able to remove low-value goals, sacrificing reward to prioritise the safety of the vehicle. Conversely, 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 (Benton et al., 2012), achieving an average 93.2% of the latter's reward per unit battery.

- A novel plan merging algorithm which uses causal structure to interleave actions

from new plan fragments into existing plans. We show the applicability of this algorithm to existing, widely-published domains.

- A formal model of our AUV planning domain as a Markov decision problem with continuous resources. The task of developing effective planning algorithms for long-duration AUV missions is a current challenge facing both the AI planning and AUV science communities.

1.4 Thesis structure

The remaining chapters of this thesis are outlined as follows:

- Chapter 2 provides background information on relevant planning representations and introduces our AUV planning domain. We discuss the assumptions made when defining the planning domain before presenting a formalised representation of our AUV domain in Section 2.5.

- Chapter 3 reviews relevant work within the AI planning literature.

- Chapter 4 discusses the results of our preliminary research, in which we investigated the suitability of key approaches from the literature for use in our AUV domain.

- Chapter 5 describes our novel solution approach. We first present an overview of the full system before discussing the development of the main components and algorithms in detail.

- Chapter 6 defines our evaluation strategy, presenting and analysing the results of each experiment.

- Chapter 7 contextualises our work within the wider field of AI planning, giving examples using widely published domains and discussing the scope for future work. We outline the applicability of both our approach and alternatives in the literature, specifying the exact domain characteristics which suit our approach.

- Finally, Chapter 8 provides a summary of this thesis, reiterating the main findings and contributions.

CHAPTER 2

REPRESENTING THE AUV PROBLEM AS A PLANNING DOMAIN

2.1 Introduction to Planning Representations

In life, when performing an unfamiliar task, such as travelling a long distance, it is often necessary to decide on a promising course of action in advance, especially if we have constraints or preferences which may influence our decision, such as wishing the journey to be as fast or as cheap as possible. In this travel scenario, we seek a *plan* of actions, starting from our current location, which will result in our safe arrival at our desired destination. To construct this plan we consider all possible actions for our journey, searching train, bus and ferry timetables until we find a promising combination of options which achieves our goals and preferences. The automation of this deliberative process by an algorithm or agent is called automated planning.

In order to produce a sensible plan, the agent must be able to reason about the world it operates in and the problem it is to solve. We represent the parts of the world which are relevant to solving the problem, such as the initial position of the traveller, the goal destination and a map of how the two are connected, and consider the world as a combination of these, or as being in a particular *state*, at any given point in time. *Actions* performed by the agent change the state of the world; for example, a travel action will cause the location of the passenger to be changed.

2.1.1 Preconditions and Effects

An early planning representation was that associated with the STRIPS (Stanford Research Institute Problem Solver) system (Fikes and Nilsson, 1971). The STRIPS representation influenced the design of many subsequent planning representations, many of which continue to share key components and structure. In such representations, the world is represented as states consisting of logical propositions and ground instances of predicates, known as *state variables*, such as *Passenger(Hattie)*, *At(Hattie, Southampton)* and *Train(Southampton, Birmingham)*, in our travel example. Numeric variables may also be represented, such as *Train-fare(Southampton, Birmingham) = 50*, *Money-in-purse(Hattie) = 97*. The goal(s) of the problem are also expressed as predicate instances, e.g. *At(Hattie, Oban)*. The goals are met when a state is reached in which all goal predicates are true. This state is referred to as the *goal state*. Conversely, at the start of the problem the state of the world is referred to as the *initial state*. The automated planning process seeks to find a series of actions that will transform the initial state into the goal state.

Actions are represented as having both *preconditions* and *effects*. Preconditions are instances of state predicates that need to be true in order to perform the action, while the effects define how performing the action will change the current state. Consider the following example action:

> *TRAIN-TRAVEL(passenger, start, destination):*
> *Preconditions: At(passenger, start)* $\wedge$ *Location(start)* $\wedge$ *Location(destination)* $\wedge$ *Train(start, destination)* $\wedge$ *Passenger(passenger)*
> *Effects: At(passenger, destination)* $\wedge\neg$*At(passenger, start)*

For example, if we wished to use the *TRAIN-TRAVEL(Hattie, Southampton, Birmingham)* action to move Hattie from Southampton to Birmingham, the precondition *At(passenger, start)* would only allow the action to be performed if the current state shows that Hattie is in Southampton. After using this action, a new predicate instance *At(Hattie, Birmingham)* is added to the state whilst the predicate instance representing

Hattie's previous location, *At(Hattie, Southampton)*, is now made false and deleted from the state.

2.1.2 Markov Decision Problems

STRIPS was designed to represent 'classical' planning problems, those which are deterministic and have a finite number of fully observable states (i.e. there is no uncertainty about the current state of the world or agent). However, uncertainty is common in realistic planning scenarios: trains may be late and cars may use more fuel than expected, and so these assumptions do not suit all problems. This introduces the concept of '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. A popular representation for such non-classical planning problems is to view them as Markov decision problems (MDPs) (Howard, 1960). In a classical planning problem, the deterministic effects of an action define how the current state is updated when the action is performed. In a stochastic problem where an action may have multiple outcomes, MDPs instead associate each possible resulting state with the probability of each outcome, forming a structure called the *transition function*. The concept of a goal is also adapted. In an MDP, goals are represented as numeric *rewards* associated with state-action transitions. Unlike in classical planning, where the planner seeks to achieve one or more goal states, in an MDP the planning process seeks to find a sequence of actions which earns the maximum reward given the probability of each state transition. Consequently, the use of a Markov decision problem reformulates the planning process as an optimisation problem.

By associating rewards with multiple 'intermediate' states, rather than goal states, the reward function may also be used to express preferences about the resulting solution. For example, returning to the travel domain, suppose we need Hattie to reach her destination before a certain time, as cheaply as possible. Multiple travel options are available, each with a different price and probability of arriving on time. By associating a large negative reward (penalty) with Hattie arriving late, we specify a preference over the trade-off

between price and reliability. Consequently, when we consider the utility of each plan of action, spending more to increase our chances of arriving on time appears an attractive strategy. Reward functions may be 'discounted' to reduce the influence of high rewards associated with distant future states when deciding which action to take in the current state. By varying the scale of the *discount factor*, preference can be given towards short term gains in reward vs long term overall strategy, and vice versa. Such discounts may also be used to represent risks which are outside the control of the planning process, by making future rewards less attractive. For example, in our travel domain, we could specify that there is an equal probability of all travel being cancelled due to snow, representing this by reducing the reward associated with each action by a discount factor γ, where $0 \leq \gamma \leq 1$. This is equivalent to assuming a $1 - \gamma$ probability of the process stopping after each action due to snow cancellations. While this would not influence the selection of a method of transport, as the risk of snow is uniform across all actions, it would favour more direct routes; the shorter the plan, the less discounted the eventual reward will be. Risks and uncertainty may also be represented within the transition function, for example performing an action may cause a transition to a 'failure' state with a given probability. However, whilst this would be appropriate for reasoning about the probability of each mode of transport breaking down during the next leg of our journey, it does not allow us to reason about the effect of inherent risks, such as snow cancellations, on the overall length or structure of the plan.

Solving an MDP does not result in a linear sequence of actions as in a conventional plan. Instead, the 'plan' is represented as a mapping known as a *policy*, which links each state with the best action to perform from that state. Standard methods for generating policies include the value iteration (Bellman, 1957) and policy iteration (Howard, 1960) algorithms.

The value iteration algorithm defined by Bellman (1957) (and described in Russell and Norvig (2003)) constructs an optimal policy by first calculating the utility of each state before defining which action to take in every state so as to transition to the neighbouring

state with the highest utility (i.e. the optimal course of action). The utility of a state depends on the utility of all states which may be reached from it. To calculate the utility of all states, a set of Bellman equations (1957) must be solved. However, solving these simultaneous equations is not trivial as the equations are non-linear. To find solutions, the value iteration algorithm starts with arbitrary initial utility values and iteratively updates the utility of each state based on the utility of its neighbours. This process continues until the utility updates are sufficiently small as to consider the solution as having converged. Once the utilities of each state have been calculated, the optimal action to take in every state may be computed and represented as a policy. An alternative approach for producing an optimal policy is the policy iteration algorithm, defined by Howard (1960). Instead of updating utility, as in value iteration, policy iteration starts with an arbitrary policy and iteratively generates new policies to maximise the utility of each state. When no further updates may be made, the result is an optimal policy. Elements from both approaches have also been combined to form the more efficient *modified policy iteration* algorithm (Puterman and Shin, 1978).

As the policy specifies the choice of action for each state in the problem, solving an MDP can be a computationally expensive task when the state-space of a problem is large, such as when uncertain action effects can cause a single action to transition to one of many resulting states. This computational cost is prohibitive for MDPs with continuous state variables, known as *continuous Markov decision problems* (CMDPs). Like MDPs, CMDPs assume that states are fully observable. However, in a CMDP the set of states is not finite. In order to solve a CMDP with a standard MDP algorithm such as policy iteration (Howard, 1960) or value iteration (Bellman, 1957), the continuous state-space must first be discretised. Notable methods for solving CMDPs include those presented by Bresina et al. (2002) and Feng et al. (2004), which are discussed in detail in Chapter 3.

Formally, a classical planning problem is a tuple $\langle O, s_0, g \rangle$ where:

- O is the set of operators (actions are ground instances of operators).
- s_0 is the initial state.

- g is the set of goal literals.

In contrast, an MDP is a tuple $\langle S, A, T, R \rangle$ where:

- S is the set of states the system can be in, $S = \{s_1, s_2, ..., s_N\}$

  e.g.

  $s_1 = \{$*At(Hattie, Southampton), Passenger(Hattie),...*$\}$

  $s_2 = \{$*At(Hattie, Birmingham), Passenger(Hattie),...*$\}$

  $s_3 = \{$*At(Hattie, Abergavenny), Passenger(Hattie),...*$\}$

  ...

- A is the set of discrete actions available to the vehicle, $A = \{a_1, a_2, ..., a_N\}$

  e.g.

  $a_1 =$ *TRAIN-TRAVEL(Hattie, Southampton, Birmingham)*

  $a_2 =$ *TRAIN-TRAVEL(Hattie, Southampton, Abergavenny)*

  $a_3 =$ *BUS-TRAVEL(Hattie, Birmingham, Harborne)*

  ...

- T is the transition function $T(s, a, s')$. This represents the probability that performing action a in state s will leave the system in state s'.

  e.g.

  $T(s_1, a_1, s_1) = 0$

  $T(s_1, a_1, s_2) = 0.97$

  $T(s_1, a_1, s_3) = 0.03$

  ...

- 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'.

  e.g.

  $R(s_1, a_1, s_2) = 10$

  $R(s_1, a_1, s_3) = -20$

  ...

2.1.3 Partially Observable Markov Decision Problems

Standard MDPs assume that the state of the world is fully observable. As this is not always a valid assumption, an extension of the MDP representation, the partially observable Markov decision problem (POMDP) is also widely used. In a POMDP, the current state is uncertain and we instead gain incomplete knowledge about the current state via *observations*. An *observation function* specifies the probability of making each observation having performed an action in a particular state. A single observation may indicate multiple states, causing uncertainty about which state the world is currently in. POMDPs represent this uncertainty as a probability distribution over states, known as a *belief state*.

Formally, a POMDP is a tuple $\langle S, A, T, O, \Omega, R \rangle$ where:

- S is the set of states the system can be in, $S = \{s_1, s_2, ..., s_N\}$
- A is the set of discrete actions available to the vehicle, $A = \{a_1, a_2, ..., a_N\}$
- T is the transition function $T(s, a, s')$. This represents the probability that performing action a in state s will leave the system in state s'.
- O is the set of observations, $O = \{o_1, o_2, ..., o_N\}$
- Ω is the observation function $\Omega(o, s)$, which specifies the probability of making an observation o in state s.
- 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'.

2.1.4 PDDL — Planning Domain Definition Language

A widely-used language for encoding planning problems is PDDL (Planning Domain Definition Language) (McDermott et al., 1998) which provides a standardised format for many features of both classical and non-classical problems. As PDDL is based on the STRIPS representation, actions have preconditions and effects, while states are sets of

ground instances of logical predicates. The PDDL representation consists of two parts: the *domain* file, which specifies the action schema and defines the state variables; and the *problem* file, which instantiates the initial state variables and goals. Many problems may then be constructed for the same domain. For instance, a travel domain which defines *TRAIN-TRAVEL* and *BUS-TRAVEL* actions may be used to solve multiple problems with various start and destination locations. We use this convention, referring to both *domains* and *problems* throughout this thesis. PDDL is widely used and many variations have been published. Later variants allow the representation of additional features such as conditional effects (version 1.2, McDermott et al. (1998)) and temporal constraints (version 2.1, Fox and Long (2003)).

2.2 AUV Problem Scenario

The 'AUV problem' scenario, as defined and presented in this thesis, envisages a vehicle with limited battery power and on-board memory conducting a long-duration mission to collect scientific data from 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 value of the datasets does not need to be absolute, but the relative value of each dataset should reflect any preference the scientist may have for collecting one dataset over another. At each survey area, if the vehicle has sufficient battery power and unused memory, it can perform a data-collection behaviour, such as mapping the extent of a chemical plume. 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 recovered from the vehicle, either by being present on the hard-drive at the point of vehicle collection 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.

All actions consume battery power; however, the exact amount consumed is not known in advance but modelled with a probability distribution specific to each action. For example, there is much less uncertainty in the power used while surfacing than in collecting data from a survey area. Similarly, the amount of memory a dataset will consume once compressed is also uncertain and represented with a probability distribution.

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. Once the vehicle has either collected data from all of the pre-defined survey areas or the remaining battery power is low, the vehicle should attempt to move to the predefined end location and surface ready for recovery by a support vessel. We assume that the vehicle is operating in open ocean (e.g. not under ice) and thus always has the option to surface, abort the mission and await recovery.

2.3 Resource representation

In our AUV domain we define two resources, battery and memory, which are required by the vehicle in order to execute actions. Both are finite quantities which, in a real-world application, would be defined at the start of a mission as the vehicle's total battery and data-storage capacity respectively. During the mission, the current resources available to the vehicle are monitored and these observations used to update the state variables following the execution of each action. We model the resource usage uncertainty of each action as a Gaussian distribution and, as our work is confined to simulation (described in detail in Section 5.2), we update the state variables according to observations sampled from the distribution(s) associated with each action as it is 'executed'. As memory and battery are internal to the vehicle, we assume both to be fully observable. A Gaussian (or 'normal') distribution is a continuous probability distribution with mean μ and standard

deviation σ. The area under the curve (i.e. the integral of the Gaussian distribution, also known as the *cumulative distribution function*) between two limits represents the probability of sampling/observing a value within this range. We model resource uncertainty as a Gaussian distribution due to its tractability and its suitability for approximating many real-world distributions[a].

Battery

A continuous state variable is used to represent the amount of battery power currently available to the vehicle. We assume battery power to be a monotonically decreasing variable which cannot be recharged during a mission. This is a realistic assumption for many long-distance AUV missions because recharging mid-mission would require a vehicle to return to a support vessel or mooring, as in work by Hobson et al. (2012) and Maurelli et al. (2009). In reality, environmental factors such as operating temperature may drastically alter the life of the battery. Remaining battery power is also only partially observable by a real vehicle due to poor sensing of charge. The rate of discharge of a battery depends on many factors including the discharge curve of the specific battery and the amount of current being drawn. However, we do not represent such factors in our model as while important, we consider these to be operational concerns, rather than directly relevant to the behaviour of our planning algorithm. Instead, we treat the battery as a finite repository of charge that can be accessed in arbitrary ways, and assume the battery usage probability distributions for each action to be independent. We assume that collecting larger datasets requires greater battery power than for collecting smaller datasets and so battery usage for the data-collection action is a function of the dataset size.

[a]This is due to the central limit theorem, which Gelb (1974, p. 34) succinctly describes as: "under certain circumstances the distribution of the sum of independent random variables, each having an arbitrary distribution, tends toward the normal distribution as the number of variables in the sum tends towards infinity".

Memory

A second continuous state variable is used to represent the amount of available memory in which the vehicle can store the data it collects during the mission. In reality, the amount of storage-space used when collecting a dataset depends on the type and resolution of the data collected and how well it may be compressed, as well as the duration of the action and the sampling rate of the sensor. Consequently, we assume that the exact amount of memory used for collecting each dataset is not known in advance. We represent this uncertainty as a Gaussian probability distribution and 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. If a dataset is successfully transmitted mid-mission, the vehicle may delete it from memory, increasing the available storage space. 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 renewable resource.

2.4 Over-subscription planning

The AUV domain is an over-subscribed planning problem, which means that the cost of achieving all of the goals is greater than resource availability will allow. This is a realistic assumption to make as AUV deployments are expensive, battery life is finite and there is always a demand for additional scientific data about the oceans. In over-subscribed planning problems, a choice must therefore be made about which goals to achieve. This decision is typically informed by a metric which combines properties such as cost, value or utility. This metric may also reflect any preference the end-user or operator has for prioritising the completion of a particular goal over another.

Preferences are common in real-world planning scenarios. To return to our example travel domain, we might prefer the cheapest journey, the shortest journey or perhaps the one which requires the least changes. In our AUV domain, we seek a plan which balances

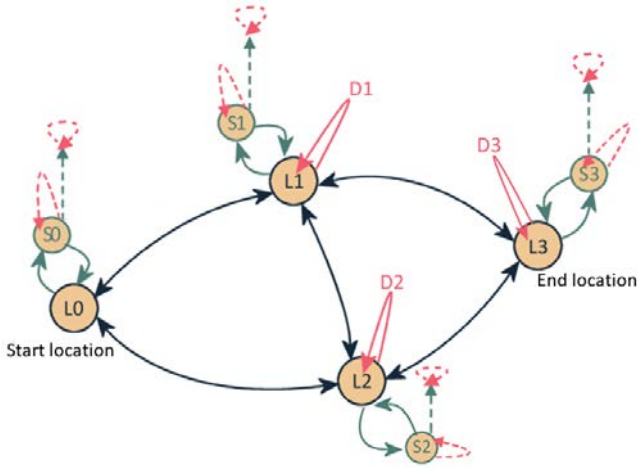

Figure 2.1: Schematic illustrating a problem in the AUV domain. Circles represent locations that the vehicle can visit, including those on the surface (prefixed *S*). The specified end location is *L3*. Black arrows represent possible *Move* actions. Solid green lines represent the *Surface* and *Dive* actions. Solid pink arrows represent *CollectData* actions. Dashed pink arrows represent *TransmitData* and *DeliverData* actions. The dashed green arrows represent the *EndMission* action.

the trade-off between obtaining additional rewards and placing the vehicle at increased risk. Preference-based planning is an extension to the classical planning paradigm and is concerned with finding not only a valid plan, but one which also satisfies the user's preferences regarding the structure or other properties of the solution. Preference-based planning (Baier and McIlraith, 2008) involves defining a criterion by which to determine how well a plan meets a set of preferences and thus an ordering which represents the value of a particular plan over another. Over-subscription planning is closely related to preference-based planning although the preferences typically relate to the utility or the cost vs benefit trade-off. Without a suitable preference metric, plans within the AUV domain may not support the aims of the domain. A plan may be valid without maximising the use of resources or safeguarding the vehicle from unnecessary risk.

2.5 Formalised problem representation

We now present a formalisation of the AUV domain, illustrated in Figure 2.1, represented as a continuous Markov decision problem (see Section 2.1.2). A full encoding of the

domain in PDDL 2.1 (Fox and Long, 2003) is included in Appendix D.

- A complete system state $s \in S$ is defined by the following set of *state variables*:
  - Location of the vehicle (auv):
    $at\_loc(auv, l)$ 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:
    $on\_surface(auv) = \{\text{true}, \text{false}\}$.
  - 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:
    $data\_size(d) = (positive\ real)$ where $d \in Datasets$ and $Datasets = \{d_1, d_2, \ldots\}$.
    This variable is not represented in the PDDL encoding, but is required when simulating the domain (or performing a mission using a real vehicle). As we use a deterministic version of the problem when generating plans, the size of each dataset is instead assumed to be the mean of the associated distribution.
  - Status of the mission:
    $mission\_ended(auv) = \{\text{true}, \text{false}\}$.
  - 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:
$move\_battery\_usage(l_i, l_j) = (positive\ real)$.
N.B. For our investigation and experiments, the standard deviation of the move action is arbitrarily defined for all pairs of locations as 25% of the associated mean.

– 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
$data\_with\_scientists(auv, d) = true$:

$mean\_data\_reward(d) = (positive\ real)$.

- 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:
  $transmit\_battery\_usage(d) = (positive\ real)$.
  N.B. For our investigation and experiments, the standard deviation of the transmit action is defined for all datasets as 10% of the associated mean.
- The reward achieved for ending the mission at the pre-defined end location, $is\_end\_location(l)$:
  $reward\_end\_location() = (positive\ real)$.

- A is the set of discrete actions available to the vehicle. As the action space is large, we use the following seven parameterised *action schemas* to concisely represent the full action space:
  - *CollectData*(auv, d) — Collect dataset d if vehicle is in the correct location.
  - *Move*(auv, l_1, l_2) — If the vehicle is at location l_1, move to l_2.
  - *Dive*(auv) — Move from the surface to depth at the current location.
  - *Surface*(auv) — Move from depth to the surface at the current location.
  - *TransmitData*(auv, d) — Attempt data transmission provided the vehicle is at the surface and has dataset d.

– *DeliverData*(auv, d) — Deliver dataset d provided the mission has ended and the vehicle has collected the dataset d.

– *EndMission*(auv, 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. By heavily penalising plans which do not finish at the agreed end location, plans which end in a different location should only be generated when strictly necessary to safeguard the vehicle and the success of the mission.

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. All action preconditions require current available resources to exceed the mean resource requirement plus one standard deviation. Note that while we include the vehicle as a parameter in all actions and some state variables, in this thesis we consider a single agent domain. However, by including this parameter, our AUV domain may be used in a multi-agent context, which forms a natural extension to our current work. As the vehicle parameter has only a single possible instantiation in all experiments within this thesis, its inclusion does not affect the complexity of our domain. However, by enabling the use of the domain in a multi-agent context, we hope our AUV domain will be of use to a wider number of researchers within the planning field.

- T is the transition function $T(s, a, s')$. As the set of states S is infinite due to the continuous variables representing resources, the transition function T is also infinite. Consequently, without first discretising the continuous variables, the use

of a standard MDP solver is infeasible. The derivation of a discretised transition function for our AUV domain is discussed in detail in Section 4.1.2.

- 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.

When computing the value of a plan, we use a discount factor γ, to represent the small probability that the vehicle could be lost or suffer damage during each action as a result of environmental factors outside the control of the planner. We refer to this event as 'catastrophic failure'. The discount factor slightly reduces the value of future rewards based on the number of actions the vehicle would need to perform to reach the rewarding state. This means that the sooner a reward can be reached, the higher its value.

The combined use of penalties and rewards 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.

CHAPTER 3

ANALYSIS OF RELATED WORK

In this chapter, we present a review of the relevant planning literature, highlighting key contributions which are related to our approach and solve problems with similar characteristics to our AUV domain.

3.1 MDP based approaches

The Mars rover planning domain discussed by Bresina et al. (2002) concerns a vehicle performing a range of scientific tasks on the surface of Mars, where each task is assigned a priority or scientific value. The domain is 'over-subscribed', meaning that achieving all of the available goals would use more resources (power, time and data-storage) than are available to the rover. Consequently, the aim of the planning problem is to find the goal set and plan which maximise scientific value, represented as utility. Actions in the Mars rover domain have a large amount of associated uncertainty. Bresina et al. describe how factors such as moving across different terrains, the angle of solar panels and temperature all create uncertainty about how much power will be required for each task. Our AUV domain shares many characteristics with the Mars rover domain, including uncertain consumption of continuous resources, stochastic actions and observable states. The AUV domain is also an over-subscribed planning problem, seeking to maximise the return of valuable scientific data, represented using rewards.

Representing their Mars rover domain as a Markov decision process (MDP — see Section 2.1.2) with continuous power usage and time, Bresina et al. (2002) compute the optimal value function for all contingencies within a small branching plan. Bresina et al. (2002, p. 80) describe the optimal value function as representing "the expected utility obtained by following an optimal (Markov) policy of the MDP representing the problem". 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. Plotting the expected utility of each branch as a function of all possible combinations of power and time, 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. (2002) 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. (2004) address this computational complexity using an approximate MDP approach. They use the optimal value function to classify regions of the continuous state space into discrete states according to the value of their utility. The level of discretisation is varied across the search space, giving a much finer discretisation over areas where the expected utility increases or decreases most rapidly (indicated by curves in the value function) whilst grouping states with a similar expected utility (plateaus in the value function) into a single state. By assessing and exploiting the structure of the value function in this way, Feng et al. are able to solve the same problem considered by Bresina et al. (2002) using a smaller number of partitions, thus reducing the size of the state space. By restricting the size of the state space, Feng et al. also reduced the computational complexity of generating a policy. 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 2005 paper, Mausam et al. 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.

They note that although the continuous variables increase the size of the state space, the discrete state space alone is infeasibly large. To address this problem, they developed a variant of the existing AO* search algorithm (described by Pearl (1985)), HAO*, which uses forward heuristic search on an aggregate state space. This state space consists of a graph in which each node represents a different combination of discrete state variables, with only the most probable reachable states included. To represent the continuous state variables they associate a value function with each node. Similar to the value functions presented in Bresina et al. (2002) and Feng et al. (2004), these functions relate current resource levels to expected reward and allow the optimal action to take to be defined for each combination of resources, for all nodes in the search graph. A relaxed version of the planning problem is solved and the resulting search graph used as a heuristic when solving the full problem, restricting the search to reachable parts of the search space. Mausam et al. (2005) evaluate the error of the policy during search, stopping the search early if a sufficiently good solution is found. In order to obtain an *admissible* heuristic to drive the search, Mausam et al. (2005) 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 the AUV domain. As the AUV domain includes replenishment actions, the assumption that resource use is monotonic does not hold for every action and therefore the heuristic is no longer admissible.

3.2 Replanning

Yoon et al. (2007) employed a different approach to probabilistic planning. Instead of generating a policy and a value function, as in MDP-based methods, they present a replanning algorithm, named FF-Replan. FF-Replan capitalises on the speed and efficiency of the deterministic planner FF (Hoffmann and Nebel, 2001), 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. Yoon et al. (2007) present two alternative methods for performing this determinisation:

1. Single-outcome determinisation — produces one deterministic action for each probabilistic action. Yoon et al. discuss how various heuristics may be used for selecting the single outcome. These include selecting the most likely outcome or that with the most 'add' effects (i.e. effects which make a state variable true — in STRIPS terminology, 'delete' effects are those which make state variables false).

2. All-outcomes determinisation — produces a deterministic action for each outcome of every probabilistic action.

The authors explain that a key drawback of single-outcome determinisation is that it relies heavily on choosing the right outcome. By instead considering all possible outcomes, all-outcome determinisation avoids this problem. FF-Replan does not reason about the probability of encountering particular outcomes and so will not intentionally avoid actions which may result in failure. However, the authors state that in many real world problems, the most desired effect is quite likely, which means that single-outcome determinisation will often perform well.

Using single-outcome determinisation, FF-Replan won the 2004 International Probabilistic Planning Competition (IPPC-04) (Younes et al., 2005) and, despite not being an official entry, was the top performer two years later on the IPPC-06 domains (Bonet and Givan, 2006) (using all-outcomes determinisation). When solving problems in domains which suit replanning, such as logistics domains, the performance of FF-Replan is very impressive and is considered to be 'state-of-the-art'. Yoon et al. (2007, p.357) attribute some of this success to the fact that most competition problems did not exercise "the full difficulty that is potential in the probabilistic setting" and caution that it is easy to develop domains which exploit FF-Replan's weaknesses. Such domains include features

such as 'dead-end' states, states which represent a failure from which the planner may not recover. One example of a dead-end is in the Blocksworld-based 'exploding-block' domain (Younes et al., 2005), where placing a block may destroy the object on which it is placed, such as a goal block or the table, consequently preventing the completion of the goals. Little and Thiebaux (2007, p.3) present a discussion on the "probabilistic interestingness" of the IPPC planning domains. They identify structural properties of interesting problems which include "the presence — or lack thereof — of dead-end states" and "the degree to which the probability of reaching a dead-end state can be reduced through the choice of actions".

As our AUV domain features 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. The AUV domain also features 'dead-end' states, which are encountered when the vehicle's resources are exhausted or it experiences random catastrophic failure. For the purposes of this discussion, we will disregard the case of catastrophic failure as it has a uniform probability of occurring after every action and (other than by favouring the construction of shorter plans) may not be actively avoided by a planner. Little and Thiebaux (2007) consider such "unavoidable dead-ends" to be probabilistically uninteresting. However, 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 the vehicle and mission at risk.

In a follow-up paper of 2008, Yoon et al. take an alternative approach based on hind-

sight optimisation (HOP), with the aim of achieving good performance on domains which previously defeated FF-Replan. Hindsight optimisation is a method of online action selection which approximates the value of a given state by solving a set of deterministic planning problems which use the given state as the initial state. The determinisation of each action outcome is specified by a random number sequence, and so the outcome of a particular action may change if its position within the plan (and thus the random number sequence) changes. In this way, hindsight optimisation samples possible outcomes to approximate the value function present in MDP-based methods, using the outcome likelihoods to inform the selection of the next action for execution. Yoon et al. (2008, p.1012) describe FF-Replan as a 'degenerate approximation to HOP', which, instead of reasoning about the likelihood of future outcomes, optimistically selects the most beneficial. Yoon et al.'s new system, FF-Hindsight (2008), outperforms FF-Replan on many IPPC problems. The authors also evaluated FF-Hindsight on Little and Thiebaux's (2007) probabilistically interesting problems and recorded significant improvements in performance over FF-Replan.

Bidot et al. (2009) present a scheduling framework to investigate how best to manage uncertainty. Their framework permits the combination of multiple scheduling approaches within a single flexible conceptual model. An initial partial schedule is generated over a short-time horizon, based on the most probable observations and outcomes. During execution, if the uncertain durations of activities causes the expected value of the 'tardiness' cost function to increase significantly, they generate another schedule online with additional ordering constraints to attempt to reduce the computational cost. If a choice of activity is based on an observation, they delay the decision until the observation has been made, generating a new schedule for the chosen activity. The generation of a schedule must take place before it can be executed, so they trigger the scheduling of additional activities when reaching the end of the short-term horizon.

Bidot et al.'s methodology is similar to ours in that they monitor the execution of the schedule, triggering online updates in response to observations. They are also not

concerned with the generation of schedules (or plans, as in our case), instead incorporating a standard scheduling system within their execution framework. However, Bidot et al. consider scheduling problems where resources and start times must be assigned to fixed sets of activities, rather than over-subscribed planning domains, where changes to the goal set affect the overall structure of the plan and the inclusion of particular actions and constraints.

Also concerned with execution strategies, Cooksey et al. (2013) consider a marine domain in which the user may introduce new goals during execution, i.e. in response to analysing datasets which have been transmitted from the AUV mid-mission. Although not concerned with plan generation or action ordering, they produce a temporal plan execution policy which allows the efficient integration of new objectives by balancing proactive and delayed execution strategies.

3.3 Contingency planning

Solutions based on the construction and execution of branching plans are popular for domains with uncertainty (Dearden et al., 2003; Pryor and Collins, 1996; Draper et al., 1993; Conrad et al., 2009).

Drummond et al. (1994) consider an astronomical telescope domain where observation requests with temporal constraints must be scheduled. Each observation task may only be performed during a limited time-window each night and the exact duration of each task is not known in advance. Just-In-Case (Drummond et al., 1994) models the uncertainty in action duration, computing the points in the schedule where failure is most likely to occur (i.e. when an action attempts to start outside its permitted time-window). Using data from previous runs of actions on the telescope, the mean duration and standard deviation of each action can be calculated and used to define sets of possible start and finish times for each action. Just-In-Case then calculates where the schedule is most likely to break based on the probability that tasks will be reached without the schedule failing and the

probability that this will occur within each task's required time-window. Starting at the point with the highest probability of failure, contingent branches are then added to the schedule in an any-time fashion until the execution of the schedule is due to start. Each branch defines the course of action to be taken from that point in the event of a schedule failure.

Drummond et al. (1994) report great success with their system. By raising the number of contingent branches from zero to ten, the amount of time per night that the schedule ran without encountering a failure increased from approximately 63% to 96%. However, when they introduced greater uncertainty by increasing the standard deviation, they recorded a drop in performance. They attribute this disparity to certain characteristics of their domain which simplify the problem for their algorithm. These include the relatively small duration uncertainty (the standard deviation of an action's duration was set to 2.5% of its mean when achieving 96% completion) and that total failure of the system is typically due to a small number of key break points (for which the Just-In-Case algorithm can then add contingencies). Drummond et al. suggest that the problem of duration uncertainty may be avoided by simply assuming that each task will require the worst-case (longest) amount of time. However, as with over-conservative AUV mission formats, this approach often results in the available resources being under-used, leaving the telescope idle between tasks.

Considering the AUV domain, whilst we shall show in Section 4.2 that changing the plan during execution is beneficial, we feel that adopting Drummond et al.'s (1994) approach, adding contingencies at points in the plan where failure is most likely, may limit the vehicle's options for recovery. For example, if we wait until the battery is close to exhaustion before adopting a contingent strategy, it is very unlikely that the vehicle will have sufficient resource left to perform a useful recovery behaviour, such as continuing to the end point of the mission. Instead, we feel that branches should be added at the points where changing the plan is likely to be most beneficial, e.g. where the expected reward or probability of success of a branch exceeds that of the straight-line plan. By

removing a task which is expected to fail (due to low resource availability) long before the point of failure, we can also remove any actions which were only required to facilitate the execution of the task. For example, if we remove a data-collection task in the AUV domain, the vehicle is no longer required to drive to the data-collection location, saving battery power. This decrease in the expected battery consumption thus increases the probability of the contingent plan succeeding. If sufficient battery is available, additional lower-cost tasks may also be achievable. If we instead wait until the point at which plan failure is most likely, such savings would not be possible.

Gough et al. (2004, p. 24) reinforce the motivation of this thesis stating, in relation to the planning literature, that "one largely overlooked area is that of uncertainty in resource consumption amounts, for example time, battery level or fuel", a statement supported by our own literature search. Citing the inefficiencies of using conventional over-conservative plans in domains with large resource uncertainty, Gough et al. 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. (2004, p. 25) 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. To ensure the success of the plan, Gough et al. instead conservatively assume that all actions in the main plan consume resources equal to the 95^{th} percentile of their associated resource usage distributions (replenishment actions are not currently represented). This means that each action has a 95% probability of executing successfully.

Gough et al. use an 'opportunity tree' to represent the probability of taking each branch in the plan. They use this to calculate the average expected utility of making each choice in the plan. At run-time, the branch with the highest expected utility is chosen, provided that upon making this choice there remains a path through the branching plan with at

least a 95% probability of completing successfully. If no plans exceed the 95% threshold, the algorithm instead chooses the branch with the highest probability of completion. As opportunities only extend the main plan and do not offer contingencies, the option with the lowest resource requirement is the main plan. After executing an action, the resource usage of that action is now known and is propagated through the opportunity tree to update the expected utilities and reduce uncertainty.

Gough et al. (2004) compare their approach to both a conservative straight-line plan (i.e. without opportunities) and a greedy strategy which takes the opportunities as soon as they are reached. Their approach achieves an 18.31% increase in utility over the conservative plan, with a negligible decrease in success rate. However, their approach requires 424 times more computation time during plan execution than the straight-line plan, where a typical plan contains 25-35 actions. In contrast, the greedy approach achieves an 8.59% increase in utility over the conservative plan, requiring only 18.72 times the online computation time. The decrease in success rate of the greedy strategy is also deemed negligible.

Using a branching plan-based approach similar to that of Gough et al., Coles (2012) 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. Prior to execution, a pessimistic initial plan is generated which has a high probability of success but does not capitalise on the additional reward available for completing 'soft' goals. After each action in the plan, the algorithm attempts to add a branch, assuming that the resource usage for each action executed so far has been the mean of its distribution. During execution, a branch is executed if resource availability in the current state meets its conditions.

Whilst the approaches presented by both Gough et al. (2004) and Coles (2012) may be able to produce valid solutions to AUV domain problems, there are several key differences between our approaches. Both Gough et al. and Coles consider branches solely as points

to capitalise on opportunities to achieve additional goals. Branches are therefore not also used to avert failure, by reducing the number of goals achieved by the current plan, as in our approach. Both approaches instead guard against failure by ensuring initial plan and all branch conditions adhere to a high confidence level. In the evaluation presented in her 2012 paper, Coles defines this confidence level as 0.95 and 0.99. In the average case, this very high confidence threshold means it is unlikely that goals would need to be removed to avert failure caused by the exhaustion of resources. However, it is still possible that the success probability may fall, potentially leading to mission failure. Without the ability to reduce the resource cost of a plan, e.g. by removing goals, it is impossible to react to a decrease in success probability. Success probability may increase again if actions use less of the resources than expected, but this is out of the agent's control.

Whilst increasing the efficiency of a plan above that of conservative straight-line plans, a very high confidence threshold will, on average, inevitably continue to result in over-conservative plans. While 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 0.99 theoretically almost guarantees the safety of the vehicle, it also means many achievable opportunities are likely to be missed.

Our approach instead favours a lower confidence threshold coupled with the ability to avert failure by removing goals should the probability of success fall. Opportunities may still be taken if resources allow, but unlike in the approach taken by Coles (2012), the size of the current goal set may both increase and decrease during execution. This leads to many more possible combinations of goals. When combined with a large number of branch points (such as after every action, as used by Coles) this would require the construction of an infeasibly large branching plan. If our approach only considered *either* the addition of new goals *or* the removal of existing goals, building a contingency plan, as in work by Coles

(2012) and Gough et al. (2004), would be feasible. A complete contingency plan would allow us to compare the expected value of choosing to take a particular branch against that of taking all other branches in the plan, and make an optimal choice. However, as we permit the option of both adding and removing goals from the current goal set, and thus the current plan, we cannot build a complete contingency plan which represents all possible goal additions and removals. This is because our algorithm may add a goal at one branch point, before potentially removing it at the next. If resource availability dictates, such modifications to the goal set may theoretically then be repeated throughout the execution of the plan. Consequently, the number of possible contingencies which may be considered by our system is infeasibly large for representation as a complete contingency plan. For a discussion regarding the calculation of the number of contingencies which may be considered by our approach, see Appendix C. It is for this reason that, while we define branch points in advance, the construction of the contingent branch itself takes place at run-time. This means that we only invest computation to find branches that are relevant in the states encountered during execution.

In their 2013 paper, Wang et al. combine execution monitoring with deterministic contingency plans and MDP policies for use in domains with partial observability. The authors describe a class of problem in which an agent completes tasks, aided by observation-making actions which provide incomplete information about the world. They use the 'RockSample' domain as an example. In this domain, a rover is tasked with taking a sample from a 'scientifically interesting' rock. While the sampling action is deterministic, i.e. sampling an interesting rock will always result in a 'good' result and vice versa for an uninteresting or 'bad' rock, the rover does not know which rocks are interesting in advance. Consequently, it must perform 'observation-making' actions to assess the value in sampling a particular rock. These observation-making actions are stochastic and so checking a rock only results in the correct observation 80% of the time.

As properties of this class of problem include observational uncertainty and thus only partially observable states, there are clear similarities with POMDPs (defined in Section

2.1.3). However, Wang et al. note that, unlike POMDP representations, their domain does not include stochastic action effects, i.e. where there is uncertainty about how performing an action will alter the world state. While the outcome of performing their observation-making actions is uncertain, they do not class this as the same as having stochastic action effects because their observation-making actions do not change the state, but stochastically observe it. Wang et al. (2013, p. 5285) state that using a POMDP solver to solve the problem they are interested in would be infeasible as "problems with fewer than ten thousand states (equivalent to only 13 binary variables) take hours to solve". Instead, they generate initial plans for a simplified problem, which assume deterministic observations, and explore the use of execution monitoring to increase the overall quality of the plan by improving the use of observation-making actions at run-time.

Wang et al. present and analyse their approach using two different methods for generating the initial plan. Firstly, they employ a contingent planning approach which generates a plan using FF (Hoffmann and Nebel, 2001), assuming the most likely outcome for each observation-making action, before augmenting this plan with branches defining the actions to take in the event of each alternative outcome. The result is a complete contingency plan, with branches for all possible observations. Their second approach uses an MDP planner, SPUDD (Hoey et al., 1999), to produce a policy which replaces the branching plan in the contingent planning approach.

Once either a contingency plan or policy has been successfully generated, plan execution may begin. During execution, when an observation-making action is reached, a 'value-of-information' calculation, based on utility, is used to select the most beneficial observation-making actions to perform at this point. The observation-making action with the highest value-of-information is the one whose updates to the rover's belief state result in the greatest improvement in expected plan quality. Observation-making actions are greedily selected for execution according to their associated value-of-information until all actions which provide a benefit to plan quality have been executed.

Wang et al. (2013) compared both approaches for plan generation, with and without

execution monitoring (and associated run-time updates), to an optimal solution found using a POMDP solver, symbolic Perseus (Poupart, 2005). For each approach, they evaluated the performance by measuring the total reward and computation time on two domains, including RockSample. The results show that while the POMDP solver predictably achieved the best rewards, execution monitoring increased the reward available when used with either the MDP or contingent plan generation approaches, compared to when each approach was used without execution monitoring. The computation time when using execution monitoring was found to be orders of magnitude less than when using the POMDP solver.

The class of problems described by Wang et al. (2013), is similar to that of our AUV problem, but there are several key differences between the two. While both contain significant sources of uncertainty, in the class of problems described by Wang et al., this uncertainty is associated with observations, whilst in the AUV domain it primarily relates to resource usage. This means that problems described by Wang et al. are best represented within the POMDP model, whereas the AUV domain, in which the state is fully observable, is an MDP. As the resource usage within the AUV domain is continuous and the uncertainty represented using Gaussian probability distributions, AUV domain problems are specifically continuous MDPs (see Section 2.5). Uncertainty over continuous variables is not described by Wang et al. as a property of the class of problems they are interested in.

As the plan generation approach presented by Wang et al. (2013) relies on either: defining a branch for each possible outcome of every observation-making action in the plan, or a full policy covering every eventuality, it would not be possible to use this approach, in its current form, when the number of outcomes is continuous. To be as effective in the AUV domain as in the RockSample domain, their approach would require branches to be computed for all values that both resources may take following all actions in the plan. The number of branches required would therefore be infinite.

Plans in the AUV domain are subject to resource constraints, which makes the cost of

execution an important factor when determining the quality of a plan. While action costs are present in the RockSample domain, and are used when calculating the utility of each branch, there do not appear to be firm constraints on overall plan cost. When generating plans, Wang et al. assume a deterministic outcome for each observation-making outcome (defined as the most likely). As the effect of each observation outcome on the belief state (and thus on the quality of the plan) is not represented during plan generation, Wang et al. describe how action cost is the only distinguishing factor when the planner is considering which observation-making action to choose. Consequently, the planner will always choose the action with the minimum cost, although at run-time the execution strategy may favour a more expensive alternative. Therefore, Wang et al. state that the cost estimate of a plan, prior to execution, will always be an underestimate of the true cost. As violating a resource constraint in the AUV domain results in mission failure (including potential loss of the vehicle) it is important that the cost of plan execution should not be under-estimated and may be drastically reduced at run-time to safeguard the vehicle if available resources are running low.

3.4 Planning problem decomposition

Problem decomposition employs a 'divide and conquer' methodology, solving each part of the problem individually before combining the solutions into an overall plan for the full problem. By analysing the interactions between goals, Yang (1997) decomposes a problem into multiple sub-problems, each representing a subset of the goals. Yang presents an example domain where the agent needs to both paint a house and go shopping. In this case, the problem may be divided into two subsets: goals relating to painting and goals relating to shopping. In Yang's (1997) decomposition algorithm, G_{DECOMP}, the interactions between goals are first estimated using a machine learning technique, trained on a library of past plans. A graph of goals is then constructed using the number of interactions between each pair of goals as a weight. Goals are greedily grouped into subsets, first selecting the

goals that, when combined, have the largest number of interactions. The weights in the graph are recomputed accordingly and new goals selected until the branching factor of searching the weighted graph is less than that of the original problem (once the number of ways to resolve potential conflicts during the recombination of the final solution has been taken into account). For each of these sub-problems, a set of solutions is found and a single plan selected. Conflicts between the plans are represented as a variable in a constraint satisfaction problem, with the set of conflict-resolution methods representing the variable's domain. Conflicts can then be resolved using techniques from constraint satisfaction planning. Finally, redundant and duplicated actions are merged into a single action to improve the efficiency of the resulting solution. A group of actions may be merged if there exists another operation which is cheaper and preserves the validity of the plan. Yang (1997, p. 132) presents an algorithm which computes the optimal merge using dynamic programming, although for larger problems this approach becomes computationally infeasible as the time complexity of the algorithm "increases polynomially with the length of plans". Yang instead presents an approximate approach which considers a limited number of actions for merging and greedily commits to merges. Yang states that as each step is now considered only once, the approximation algorithm has a linear time complexity.

As well as improvements in efficiency over conventional plan generation methods, Yang (1997) lists other benefits of plan decomposition:

- Concurrent plan generation — The construction of a plan can be carried out by multiple planners in parallel.

- Identification of actions for re-use — A new problem may share goals with a previous decomposition from the same domain, enabling sub-plans to be re-used.

- Multi-agent planning — In a similar vein to concurrent plan generation, decomposing a problem and analysing the interactions between sub-problems, the divide-and-conquer methodology of plan decomposition is suited to multi-agent applications,

where agents need to both co-operate and function independently.

- Over-subscribed domains — When there are not enough resources to achieve all goals in a problem, the most cost-effective subset of goals should be selected. Breaking a problem down into individual goals and analysing the interactions between them could be used to inform goal selection.

Motivated by a desire to efficiently parallelise the planning process, Sebastia et al. (2006) also use a plan decomposition approach. They present a decomposition algorithm, STeLLa, for use with the STRIPS representation. They focus their efforts and computation into decomposition methods in order to simplify the plan recombination process. To achieve this, they present a technique called 'chronological decomposition', which divides a problem according to the order in which sub-problems must be achieved. They first construct a graph of 'landmarks' — literals which are true at some point in all solution plans. Ordering constraints between landmarks are then found and any positive interactions (those where an action may benefit multiple others) identified. Sets of landmarks are then treated as 'intermediate' goals, which, when solved by an external planner, result in a sequence of conflict-free sub-plans. By decomposing the problem in this way, the eventual plan may be constructed by simply concatenating the sub-plans. Sebastia et al. (2006) evaluate STeLLa against FF (Hoffmann and Nebel, 2001), LPG (Gerevini and Serina, 2002) and VHPOP (Younes and Simmons, 2003), using each planner to solve the original non-decomposed problem and comparing this to the performance when used to solve the sub-problems as part of STeLLa. They found that all three planners were able to solve more problems when used as a part of STeLLa, stating that the simpler sub-problems were easier for the planner to solve. Additional experiments showed that the quality of plans produced using decomposition rivalled those generated when planning for the full problem. However, decomposition using STeLLa was found to be slow, which the authors attribute to the additional parsing required when solving each of the multiple sub-problems. They show that the speed issue may be addressed by solving the sub-problems in parallel using multiple planners.

3.5 Plan repair

The term 'plan repair' is used in the literature to refer to a wide range of algorithms. In this section, we present selected key works relating to plan repair, constructing a picture of the state-of-the-art.

The central concept of plan repair is to adapt an existing solution to solve a new problem. Typically the new problem will differ from the original in one or more elements, including changes to the goal state, initial state or, in the case of online plan repair, the current state. As planning is often an expensive task, it seems intuitive that this cost may potentially be reduced by adapting a previous solution to a similar problem. However, Nebel and Koehler (1995) show that in theory, the costs of modifying an existing solution to suit a new problem are higher than generating an entirely new plan. Their model of plan modification has two parts, first they select a candidate plan from a library of existing plans, and second, they modify this plan to satisfy the new problem. They describe the NP-hard 'matching' problem, to correctly associate objects in the candidate plan with objects in the new problem, as crucial to overall system performance.

3.5.1 Repair by refinement

Despite the conclusions of Nebel and Koehler (1995), many authors have investigated plan repair and reported promising results. A popular approach is to extend the ideas of refinement planning (Kambhampati et al., 1995; Kambhampati, 1997), 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.

In their 1995 paper, Hanks and Weld present a domain independent algorithm, SPA (Systematic Plan Adaptor), which first retrieves a suitable existing plan from a library before adapting it to suit a new problem. SPA is based on partial-order planning algorithms where solutions are found by refining empty plans, adding actions to achieve unsatisfied goals or preconditions as well as ordering and binding constraints to avert threats. SPA

records all refinements using a 'reason' data structure which links the newly added action or constraint to why its addition was necessary (e.g. to resolve a threat).

Investigating the formal properties of SPA, Hanks and Weld show their approach to be *sound* (i.e. all plans found are valid solutions), *complete* (i.e. the algorithm will always return a plan if one exists) and systematic (i.e. the search algorithm does not have redundant behaviour, such as returning to a previously visited plan adaptation). However, despite praising the value of such properties when understanding planning algorithms, Hanks and Weld (1995, p. 356) state that they "are neither necessary nor sufficient to build an effective planner" as "the value of a framework for planning must ultimately be measured in its ability to solve interesting problems". A key part of the effectiveness of such an approach is the suitability of the library plan and, in particular, how closely it resembles the new problem to be solved. Using a simple example domain, Hanks and Weld highlight a relationship between the suitability of the library plan to the problem and the time required to adapt the plan into a solution. SPA does not attempt to address the library plan selection problem, although Hanks and Weld suggest it as a key area for future research.

Boella and Damiano (2002) present a hierarchical refinement planner as part of their reactive agent architecture. During execution, if the expected utility of the currently executing plan deviates significantly from previous values or a goal can no longer be met, the system attempts to replan. In a similar way to Hanks and Weld (1995), Boella and Damiano search for a solution by retracting and refining the current plan. However, instead of retracting planning decisions according to a 'reason' data structure, they search the abstraction hierarchies for a more abstract, partial version of the current plan. They make the assumption that "a plan failure can often be resolved locally, within the sub-tree..." (Boella and Damiano, 2002, p. 190) and only expand the search beyond this when all local partial plans have been considered. Promising partial plans are selected for refinement according to their expected utility, while unpromising partial plans are pruned from the search space. In the worst case, the complexity of the replanning algorithm is

the same as planning from scratch because the search may eventually be expanded to the full problem. However, they suggest that if their assumption is true then the size of the search space will be reduced.

Van der Krogt and de Weerdt (2005) also use concepts from refinement planning. However, they propose an additional strategy, *unrefinement* planning, which allows the removal of constraints or actions which prevent the goals from being met by the current plan. If the current plan is no longer a valid solution to the problem, (e.g. due to a change in current state or goals), their system creates candidates for refinement by first unrefining the current plan, removing actions which depend on either the initial or goal state. As well as removing each action, they also remove all actions which are causally linked to the removed action, up to a specified depth. Each resulting partial plan is then grouped into 'cuts' such that no actions in a group were previously causally linked via a removed action. The combined preconditions and effects of all actions in each group are then represented as a macro action and used by the refinement planner to generate a solution to the new problem. To repair plans efficiently, they favour solutions which require fewer refinement/unrefinement steps to satisfy the problem over those which may be closer to optimal.

In their 1996 paper, Stephan and Biundo also use refinement planning within their deductive planning system to find a solution given an abstract plan as an input. They argue that starting with a prefabricated plan increases the efficiency of their deductive planning approach by reducing the number of necessary proofs, which are time-consuming to compute.

3.5.2 Online plan modification

In this section we consider plan repair approaches applied online, during the execution of the plan.

In the context of online plan repair, Fox et al. (2006b, 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 (Gerevini and Serina, 2002) 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. (2009) present a system that uses online plan modification to enable a planetary rover to perform opportunistic science. Their work is part of a wider project (Woods et al., 2009) relating to the development of the European Space Agency's ExoMars rover. As rover missions are very expensive and high-risk, rover operations are typically closely supervised and directed by a science team. However, communication times between Earth and Mars are significant and introduce delays, leading Woods et al. (2009, p. 359) to describe the whole cycle of planning, supervision, execution and assessment as "inherently inefficient". While they state that the technology is not yet mature enough to allow the vehicle to perform extensive planning and science assessment onboard, these inefficiencies motivated work to further evaluate the use of rover-based plan modification for opportunistic science. The system presented by Long et al. (2009) has three components:

1. SARA — science assessment and response agent.
2. AAPI — arm agent and interface.
3. TVCR — time-line validation, control and replanning system.

In this review, due to its relevance to this thesis, we focus on the last component, TVCR, the development of which was also discussed by Fox et al. (2006a). TVCR monitors the execution of the initial plan, generated on Earth. Actions in this plan which relate to a particular task are grouped into 'plan fragments', along with any constraints on their execution. Constraints between fragments, such as ordering and temporal constraints

may also be imposed. If the executive determines that the current plan is no longer valid, the plan is modified by first removing fragments 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 vehicle's resources. Each plan fragment is assigned a science 'priority value' by human mission planners, which are used to guide the greedy removal of fragments from the plan. Long et al. (2009, p. 4) assume that there is "never to be a situation in which a combination of lower priority fragments might be preferred to a higher priority fragment". 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. Each fragment is also assigned a list of its hardware requirements and estimated resource costs which are used to greedily select fragments to add to the plan. Long et al. (2009) highlight the operational challenges of planning for such high-risk domains and echo the views of the AUV community (Brito et al., 2012) when stressing the importance of fostering trust between both the planning community and vehicle operators. With similar motivations to Stephan and Biundo (1996), Fox et al. (2006a) 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.

With similar motivations to Long et al. (2009), Rabideau et al. (1999) describe their automated scheduling and planning environment (ASPEN) which uses 'iterative repair' to modify the plan to avert any conflicts which may occur during execution (such as the violation of a resource constraint). Within the system representation, each constraint and conflict type are linked to ways in which they may be violated and repaired. Given a schedule and associated conflicts, ASPEN then selects a conflict and linked repair method before searching the options for implementing this method, such as which activity to move or parameter to update. The selected repair is then performed and the constraints updated. This process continues until all conflicts are resolved or the permitted time limit

is reached. As the search contains many decision points, ASPEN uses multiple heuristics to reduce this complexity. These heuristics allow the user to specify preferences about the mission, such as an order of preference for both resolving constraints and choosing repair methods.

Sharing motivations with both ourselves and others in the literature, Bresina and Washington (2001, p. 24) label current rover mission formats as conservative, "sacrificing performance for safety". They instead 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. If execution is completed as expected, the rover will have only performed the actions in the straight-line plan. Contingent branches (which may themselves contain other 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. These alternative plans may either replace the remainder of the currently executing plan or be inserted within it. The former case may be used in the event of action failure, for instance, where a back-up alternative plan may return the rover to a safe state, awaiting the next instructions from Earth. Examples given by Bresina and Washington (2001) of when an alternative plan might be inserted into the current plan include: to re-establish a maintenance condition, preventing the failure of a task; and to continue the execution of commands which may have been suspended by an anomalous power state. In Washington et al. (2000) the authors also suggest alternative plans may be used to facilitate additional science, should an opportunity be detected during a long traverse. 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. (2000) state that they do not currently include uncertainty in their resource models, leaving this as further work. At the current time-step, if the expected utility of executing the rest of the plan without the next action exceeds the utility of the plan including it, the action is skipped. Any further actions which depend on the removed action will also be skipped and the utility estimates updated. When inserting an alternative plan, the same approach of skipping actions in the current plan is employed to remove any potentially redundant or incompatible actions. Bresina and Washington (2001) describe how the insertion of an alternative plan may cause harmful interactions with the current plan. To avert this, both alternative plans and those sent to the rover from Earth are annotated with constraints required for plan validity and flight safety. During alternative plan insertion, conflicting constraints may change the applicability of planned actions, which may result in these actions being skipped. Before committing to the resulting plan, the utility estimates are used to compare it to both the current plan and any other available courses of action. The plan with the highest expected utility is executed.

Bresina and Washington (2001) use a reactive utility-driven architecture to attempt to increase Mars rover productivity, given uncertainty in the rovers' interaction with the Martian environment. While this work has much in common with both our AUV domain and solution approach, there are key differences which set them apart. While Mars Rover missions involve huge costs and risks to the safety of the vehicle, there is always the option to halt execution and await new instructions from Earth. If an important task is skipped because it requires more resources than are currently available, the mission planners can later send a plan to re-do this task, once the resource has been replenished. However, an AUV has finite battery-life and is unable to recharge mid-mission. Typically, AUV deployments and support-ship time may be allocated months in advance, allowing very little time for reattempting tasks — especially as some tasks may require the vehicle, and possibly the support ship, to revisit an earlier location possibly tens or hundreds of kilometres away. It is for this reason that we prefer a more deliberative, goal-driven

approach. Instead of skipping the next task, should its inclusion lower the expected utility of the mission, if we observe a decrease in the probability of mission success, we reason over the set of all goals, removing those which are expected to require the most resources for the smallest gains in reward. Equally, should resources allow, we consider the inclusion of additional data collection goals to maximise the expected reward of the mission. Unlike Bresina and Washington (2001), who can only insert an alternative plan or take a contingent branch at the current state, we reason about where the new tasks should be added. Even if the inclusion of a particularly valuable goal appears attractive in the current state (when compared to the current plan), considering the option of delaying its addition until a later state may reduce the resources required to achieve the same increase in reward. For example, later in a plan the vehicle may be closer to an additional data collection location than it is at present. By adding this goal at the later point, the vehicle uses less battery to achieve the additional reward. When merging the plan fragment associated with a new goal into the current plan, we examine the causal structure of both plans. This allows actions to be interleaved and preconditions common to multiple goals to be achieved by a shared action. Instead of skipping actions in the current plan if maintenance conditions are broken during the merge, we employ a generative planner to insert additional actions, where necessary, to stitch the two plans together and maintain the causal structure. Our approach is motivated by a requirement for a fully autonomous remote vehicle, unable to contact human operators or to recharge mid-mission. Instead, we reason about the cost/benefit of each goal in the over-subscribed goal set, selecting at run-time those which maximise reward given a finite battery resource. We argue that a deliberative approach is more effective than a reactive one for use in the AUV domain.

3.6 Summary

In this section, we summarise the key properties of the related work reviewed in this chapter in two tables, shown below and on the following page. We include our approach, as presented in this thesis, for ease of comparison. If an approach reasons about and/or represents a property, it is marked ✓. If not, it is marked ×. When it is unclear, or not mentioned, the cell is marked ?. For brevity, we include time as a resource. Note, we discuss temporal planning representation as future work in Section 7.2.1.

Property	Our approach	Mausam et al. (2005)	Drummond et al. (1994)	Bresina et al. (2002), Feng et al. (2004)	Bidot et al. (2009)	Gough et al. (2004)	Coles (2012)	Wang et al. (2013)	Yoon et al. (2007), Yoon et al. (2008)	Cooksey et al. (2013)
Resource constraints	✓	✓	✓	✓	✓	✓	✓	×	✓	✓
Uncertainty over continuous variables	✓	✓	✓	✓	✓	✓	✓	×	×	?
Uncertainty over discrete variables	✓	✓	✓	✓	✓	✓	✓	×	✓	?
Non-monotonic resource usage	✓	×	?	✓	?	×	?	×	✓	?
Stochastic action effects	✓	✓	×	✓	×	×	×	✓	✓	×
Goal arbitration	✓	✓	×	✓	×	✓	✓	×	×	×
Numeric goals	✓	✓	✓	✓	×	✓	✓	✓	×	×
Contingency planning	✓[a]	✓	✓	✓	✓[b]	✓	✓	✓	×	×
Replanning	×	×	×	×	✓	×	×	×	✓	×
Plan repair	×	×	×	×	×	×	×	×	×	×
Online plan repair	✓	×	×	×	×	×	×	×	×	×
Execution monitoring	✓	✓	✓	✓	✓	✓	✓	✓	×	✓
MDP/POMDP	×	✓	×	✓	×	×	×	✓	×	×
Plan decomposition	×	×	×	×	×	×	×	×	×	×

[a] Offline generation of plan fragments at pre-defined branch points.
[b] Online generation of contingencies, shortly before choice points are reached.

Property	Yang (1997)	Sebastia et al. (2006)	Hanks and Weld (1995)	Boella and Damiano (2002)	Van der Krogt and de Weerdt (2005)	Stephan and Biundo (1996)	Fox et al. (2006b)	Long et al. (2009)	Rabideau et al. (1999)	Bresina and Washington (2001), Washington et al. (2000)
Resource constraints	×	×	×	✓	×	×	✓	✓	✓	✓
Uncertainty over continuous variables	×	×	×	×	×	×	?	?	×	?
Uncertainty over discrete variables	×	×	×	✓	×	×	?	?	×	✓[c]
Non-monotonic resource usage	×	×	×	?	×	×	?	?	✓	?
Stochastic actions	×	×	×	✓	✓	×	✓	?	×	✓
Goal arbitration	×	×	×	✓	×	×	×	✓	✓	✓
Numeric goals	×	×	×	✓	×	×	✓	✓	?	✓
Contingency planning	×	×	×	×	×	×	×	×	×	✓
Replanning	×	×	×	✓	×	×	×	×	×	×
Plan repair	×	×	✓	✓	✓	✓	✓	✓	✓	✓
Online plan repair	×	×	×	✓	✓	×	✓	✓	✓	✓
Execution monitoring	×	×	×	✓	✓	×	?	✓	✓	✓
MDP/POMDP	×	×	×	×	×	×	×	×	×	×
Plan decomposition	✓	✓	×	×	×	×	×	×	×	×

[c] Actions have uncertain durations, uncertainty associated with vehicle resources is not currently modelled.

CHAPTER 4

APPLICATION OF EXISTING TECHNIQUES TO THE AUV DOMAIN

Following our review of the planning literature, presented in Chapter 3, we applied the most relevant existing approaches to our AUV domain to further investigate their suitability. In this chapter, we present the results of this preliminary research, which inspired and informed the development of our eventual approach, described in Chapter 5.

4.1 Applying existing MDP approaches to the AUV domain

As discussed in Section 2.5, our AUV problem can be represented as a continuous MDP. A natural progression was to investigate the suitability of using an MDP solver to produce a policy for the AUV domain.

4.1.1 Calculating the size of the AUV domain state space

A formula for calculating the number of discrete states in an instance of the AUV problem can be derived by considering the state variables and their parameters, listed in Section 2.5. Some of the state variables are first parametrised by a reference to the vehicle (e.g. *(auv)*) with most also referring to either the set of locations which the vehicle may visit or the set of datasets which may be collected. The size of the state space is therefore a

function of the size of these parameter sets, where $L = |Locations|$ and $D = |Datasets|$. As our AUV domain currently considers only a single agent, we can ignore the vehicle reference parameter when calculating the size of the state space. Whilst at any of the L locations, the vehicle may be either on the surface (*on_surface(auv)*) or at depth ($\neg$ *on_surface(auv)*) and the mission may be marked as ended (*mission_ended(auv)*) or still in progress ($\neg$*mission_ended(auv)*). However, the vehicle may only finish the mission whilst on the surface, as dictated by the preconditions of the *EndMission* action, which causes (*mission_ended(auv)*$\wedge\neg$*on_surface(auv)*) to be mutually exclusive. This leaves us with three possible combinations of the *mission_ended* and *surface* state variables at any given location:

- *mission_ended(auv)*$\wedge$*on_surface(auv)*
- $\neg$*mission_ended(auv)*$\wedge$*on_surface(auv)*
- $\neg$*mission_ended(auv)*$\wedge\neg$*on_surface(auv)*

We also need to consider the number of datasets *D*. Whilst at any location, the vehicle may have collected any combination of datasets (*data_collected(auv, d)*). Of these collected datasets, any combination may also have been transmitted (*data_transmitted(auv, d)*). After the mission has ended, *mission_ended(auv)*, any datasets which have been collected but not transmitted may then be delivered, *data_delivered(auv, d)*. The resulting equation for the size of the state space in the AUV domain is as follows:

$$|states| = \left(2L\sum_{n=0}^{D}\left[{}^{D}C_n \times \sum_{i=0}^{n} {}^{n}C_i\right] + L\sum_{n=0}^{D}\left[{}^{D}C_n \times \sum_{i=0}^{n}\left[{}^{n}C_i \times \sum_{j=0}^{n-i} {}^{n-i}C_j\right]\right]\right) \quad (4.1)$$

where ${}^{D}C_n$ is the number of combinations of length n from a total set D, n represents the number of datasets collected, i the number transmitted and j the number delivered. For a very small problem with five locations and five datasets, the number of states in the AUV domain is 7550, plus two continuous resources. Whilst this appears reasonably small and manageable for an MDP solver, if we increase the size of the problem to 50

locations and 50 datasets, the number of states greatly increases to 6.34×10^{31} plus two continuous resources. Although we have not included the continuous resources when calculating the number of discrete states, the presence of continuous variables creates an infinite number of states. We can reduce the scale of the state space by discretising the continuous variables, as in existing work by Bresina et al. (2002) and Feng et al. (2004). As described in detail in the previous chapter (see Section 3.1), Feng et al. (2004) discretise the continuous variables in their Mars rover domain in an attempt to reduce the computational complexity of solving continuous MDPs. For a problem with two continuous variables and 11 discrete states, Feng et al. (2004, p. 5) evaluate the performance of their algorithm, varying the level of discretisation between 50 and 400. Given the two continuous variables in the AUV domain and an equal number of discrete partitions q to represent each variable, we can extend Equation 4.1 to include continuous variables:

$$|states| = \left(2L \sum_{n=0}^{D} \left[{}^{D}C_n \times \sum_{i=0}^{n} {}^{n}C_i \right] + L \sum_{n=0}^{D} \left[{}^{D}C_n \times \sum_{i=0}^{n} \left[{}^{n}C_i \times \sum_{j=0}^{n-i} {}^{n-i}C_j \right] \right] \right) \times q^2 \quad (4.2)$$

Using the range employed by Feng et al. as a guide when considering an appropriate discretisation for the AUV domain, we can see that with 50 locations, 50 datasets and 50 partitions per continuous variable, the number of states increases to 1.58×10^{35}. When the number of partitions q is increased to 400 the number of states is significantly larger, 1.01×10^{37}.

Bellman (1961, 1957) refers to this as the 'curse of dimensionality', preventing the use of MDP algorithms for solving realistically-sized planning problems. Feng et al. (2004, p. 1) state that "classical dynamic programming algorithms solve MDPs in time polynomial in the size of the state space". Consequently, we conclude that the use of an MDP solver for solving anything more than very small AUV domain problems would be computationally infeasible.

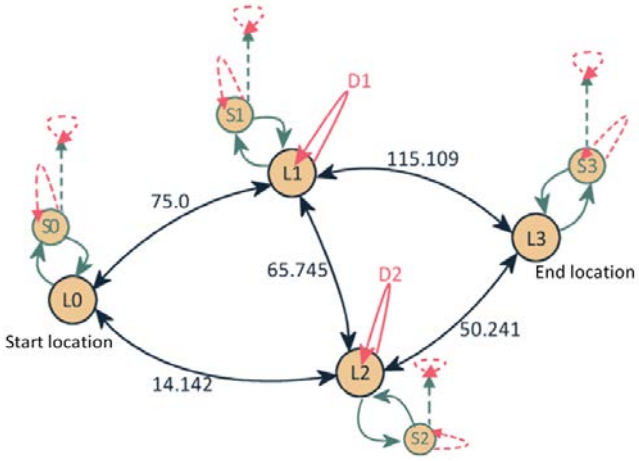

Figure 4.1: Schematic illustrating the small AUV problem. Circles represent locations that the vehicle can visit, including those on the surface (prefixed S). The specified end location is *L3*. Black arrows represent possible *Move* actions and are labelled with the distance between locations. Solid green lines represent the *Surface* and *Dive* actions. Solid pink arrows represent *CollectData* actions. Dashed pink arrows represent *TransmitData* and *DeliverData* actions. The dashed green arrows represent the *EndMission* action.

4.1.2 Computing a policy for a small-scale AUV problem

Although not feasible on realistically-sized AUV domain problems, solving the MDP for a toy-sized problem would result in an optimal solution. Optimal solutions are hard to compute for large problems, such as our AUV domain, which contain significant amounts of uncertainty and continuous resources. We decided to calculate the optimal solution to a toy-sized problem to further investigate the characteristics of our AUV domain and produce an example of what a good solution/plan looks like for this domain.

We considered a very small problem (shown in Figure 4.1) and simplified the domain by only considering battery usage. The problem comprised four locations with two datasets for the vehicle to collect, $D1$ and $D2$. We naively discretised battery availability by classifying it as 'very low', 'low', 'medium', 'high' or 'very high', where each category represents an equal division of the continuous space bounded by zero and the full/initial capacity of the battery. Although this approach is known to be overly simplistic, it is intuitive to understand and relatively straightforward to implement. As a traditional MDP method is unlikely to form the eventual solution to the AUV planning problem, due to factors such as the size of the discrete state space, this naive approach was deemed

sufficient for this investigation.

For this investigation, the states, actions and reward function of the MDP representation used were as described in Section 2.5. As defined in Section 2.5, the transition function of an MDP represents the probability of transitioning to a new state s' having performed action a in state s. Deriving an appropriate transition function to represent the probability that, given a state s with a discrete resource level (e.g. 'high' availability), applying an action would result in transitioning to a state with any other discrete resource level (e.g. 'low' availability), required careful consideration. For example, if we subdivide battery availability into three equal discrete levels (e.g. A, B and C, where A represents the highest availability), we need to represent the fact that the true value of battery, as observed by the vehicle, may fall anywhere within a given level. If, for instance, the true value for battery is at the low end of level B, close to the boundary with C, there is a high probability that performing the next action will result in a transition to C. However, if the true value is actually at the top end of B, close to the boundary with A, the probability of transitioning to C is much reduced; there is a much higher probability of staying within level B. While our discretisation considered five resource levels, for brevity we will discuss our method using an example with three levels, A, B and C, where A represents the highest availability and C the lowest. The method is easily scalable to allow for a greater number of discrete intervals. As the resource usage of each action is modelled using a Gaussian distribution, we can integrate this distribution to produce the cumulative distribution function (CDF) which represents the probability of sampling a resource value below a given point:

$$CDF = \int \left(\frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sigma\sqrt{2\pi}} \right) .dx = \frac{1}{2} \left[\text{erfc} \left(\frac{\mu - x}{\sqrt{2}\sigma} \right) \right] \tag{4.3}$$

We can use this property to calculate the probability that the resource availability will transition to within the bounds of a given resource level (e.g. 'high'/A). To calculate the probability of one level transitioning to any other, we can integrate the CDF a second

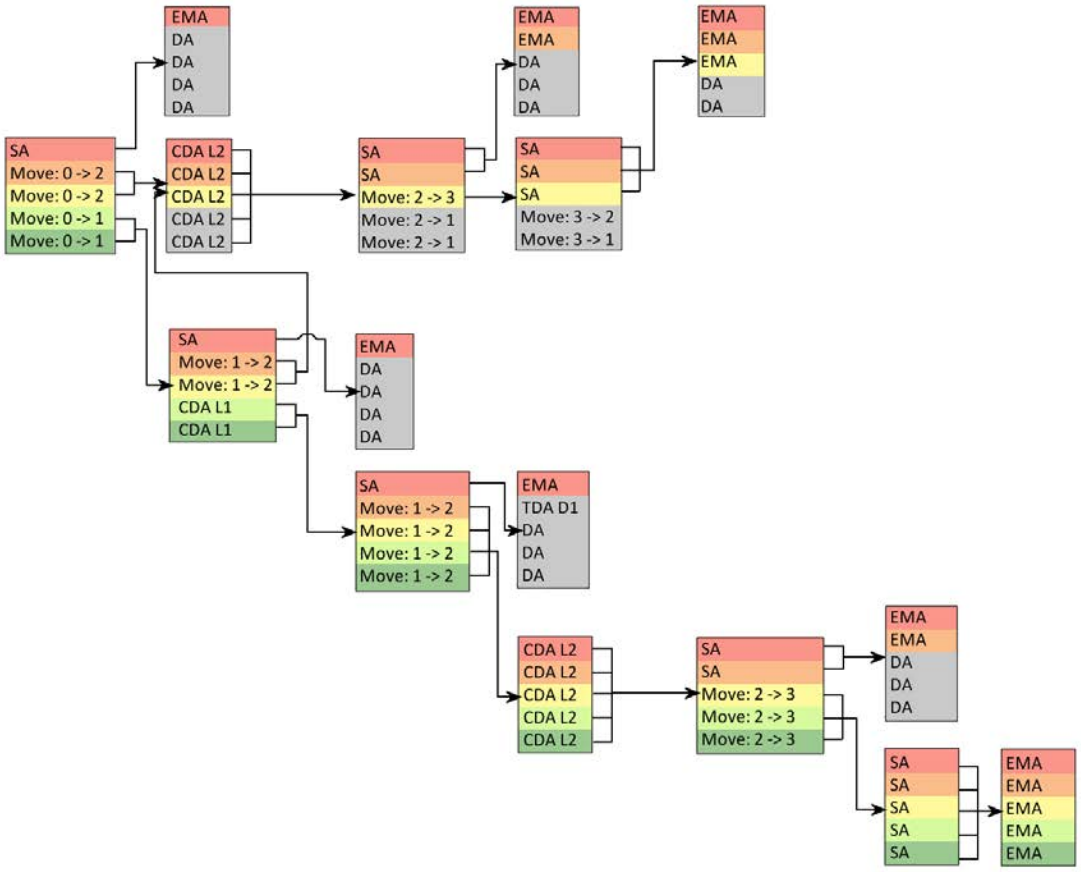

Figure 4.2: Policy tree for the small version of the AUV problem shown in Figure 4.1 with five discrete levels of battery. 'Very high' battery is shown as dark green, 'high' as green, 'medium' as yellow, 'low' as orange and 'very low' as red. States which are unreachable due to monotonic battery usage are shown in grey. SA represents a Surface action, DA represents a Dive action, CDA represents a CollectData action, TDA represents a TransmitData action and EMA represents an EndMission action. The policy tree reads from the initial state on the left to the EndMission leaf nodes on the right.

time using limits which correspond to the resource levels in question:

$$\mathrm{P(A|A)} = \left(\int_{2\mu}^{2\mu+|\mathrm{A}|} CDF.dx \right) - \left(\int_{2\mu-|\mathrm{A}|}^{2\mu} CDF.dx \right) \tag{4.4}$$

$$\mathrm{P(B|A)} = \left(\int_{2\mu-|\mathrm{A}|}^{2\mu} CDF.dx \right) - \left(\int_{2\mu-|\mathrm{A}|-|\mathrm{B}|}^{2\mu-|\mathrm{B}|} CDF.dx \right) \tag{4.5}$$

$$\mathrm{P(C|A)} = \left(\int_{2\mu-|\mathrm{A}|-|\mathrm{B}|}^{2\mu-|\mathrm{B}|} CDF.dx \right) - \left(\int_{2\mu-|\mathrm{A}|-|\mathrm{B}|-|\mathrm{C}|}^{2\mu-|\mathrm{B}|-|\mathrm{C}|} CDF.dx \right) \tag{4.6}$$

This represents the probability of transitioning to a level, e.g. C, given a current resource within the range of a second level, e.g. A.

By solving the symbolic integration in advance, we construct the transition function by simply solving the resulting equation for the desired limit. A full explanation of the derivation of the transition function is found in Appendix B.

The resulting policy tree for a problem with five resource levels can be found in Figure 4.2. When battery is very low, in all but one case the policy specifies that the vehicle should surface (SA) and end the mission (EMA) at its current location. This shows that our reward function represents the need to prioritise the safety of the vehicle. The collection of dataset $D2$ (CDA L2) is prioritised over that of dataset $D1$, as this dataset has a higher associated reward. If battery availability is sufficiently high (either 'high' or 'very high') throughout the mission, the policy specifies that the vehicle should first collect $D1$, followed by $D2$, before ending the mission at the predefined location, location 3. By performing this plan, the vehicle does not incur any penalties.

4.2 Assessing the value of changing a plan during execution

As the actions in the AUV problem use an uncertain amount of resources, committing to a single straight-line plan is likely to be sub-optimal. Following on from our review of existing contingency planning techniques (see Section 3.3), we considered the option of adopting a contingency planning approach within our AUV domain. We hypothesised that by allowing the vehicle to change its planned behaviour during execution in response to unexpected situations, the expected reward will increase above and beyond that of any single straight-line plan. Following on from work by Bresina et al. (2002) (discussed in Section 3.1), in which the authors computed the optimal value function associated with the Mars rover planning problem as a function of continuous resource variables, we designed and performed an investigation to test this hypothesis within a simplified AUV domain. We compared the optimal value functions of a small set of closely related straight-line plans to that of the same plans composed as a contingent branching plan (in the same way as later illustrated in Figure 4.4). For any combination of initial resources the optimal value function represents the expected reward obtained by following a particular plan. If the value function of the branching plan exceeds that of the individual straight-line plans, our hypothesis would be shown to be correct.

To perform this investigation, we hand-generated a varied subset of viable plans from a small AUV problem, illustrated in Figure 4.1. To calculate the value function for each straight-line plan, we considered each of the plans in the subset independently. To compute the equivalent value function for switching between plans during execution, the plans were examined and points common to multiple plans identified. At each of these common points, referred to as 'branch points', we determined the branch condition, i.e. the point at which a change in resources causes one branch to become better than another and thus should be taken, by performing Monte Carlo simulation for every possible combination of resource values. As the resource usage of each action is uncertain, Monte Carlo simulation was performed by running each straight-line plan and the branching

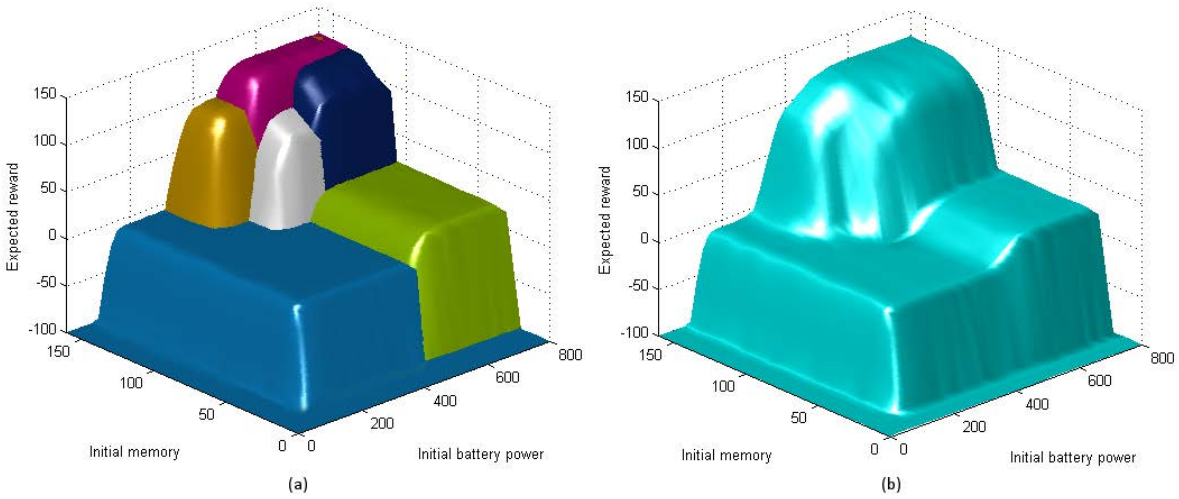

Figure 4.3: (a) The optimal value function produced by performing Monte Carlo simulation on the set of straight-line plans from the small AUV problem. Each colour represents a different plan. (b) The optimal value function produced by simulating a switching planner, showing the expected reward of switching between plan fragments at run-time. When combined with the equivalent function for straight-line plans (shown in (a)), the former is completely dominated by this function.

plan 5000 times for every combination of initial resource values. The resulting reward values from each resource combination were then averaged and combined with the data from every other straight-line plan in the subset to produce the optimal value function, shown in Figure 4.3 (a). Each colour represents a different straight-line plan, with higher expected rewards indicating the most successful plans. Negative reward indicates that the plan received more penalties than rewards. At the start of each mission, the reward is zero. As shown in Figure 4.3 (b), the optimal value function of the branching plan completely dominated that of the straight-line plans, exceeding the expected reward for all resource combinations. This shows that branching is at least as good as the best straight-line plan. In fact, at multiple locations the branching plan significantly outperforms the best straight-line plan. To illustrate this finding, we will discuss the value functions of multiple plans in more detail. Figure 4.4 shows the three plans chosen for this example. In Figure 4.5 (a) we can see that plan three is optimal given an initial battery value greater than 480 units (marked with a dashed green plane) and memory greater than 115 (marked with a dashed blue plane). Between 480 and 310 battery (marked with a dashed yellow

plane), plan two becomes optimal. Below 310 battery, plan one is optimal given any amount of memory. Plan three has an additional move action which causes the average resource requirement for this plan to be higher than that of plan two. However, as plan two does not result in the vehicle ending the mission at the specified recovery location ($L3$), the total reward for this plan is subjected to a 30 point penalty, reducing the maximum reward achievable by following this plan. Likewise, plan one does not collect dataset 2 (or end the mission at the recovery location) and thus the reward for following this plan is lower than for the other two plans. However, plan one has lower resource availability requirements than plans two and three. The points in the value function where the resource requirements exceed the amount of resources available to the vehicle are not sharp thresholds but curves. This is due to the uncertainty associated with resource usage. In Figure 4.5 (a), at the higher end of the pink curve representing plan 1, the plateau in the function indicates that an increasing majority of runs had sufficient battery to complete the plan and earn the full reward. Conversely, as the available battery decreases, the gradient of the function becomes much steeper. This indicates that as initial battery decreased, an increasing majority of runs failed to complete the plan before exhausting the battery and suffering a 100 point penalty. Within this interval (400-530 battery, highlighted in red in the enlarged area in Figure 4.5 (a) and corresponding to the same region of Figure 4.5 (b)), the value function for the branching plan outperforms both plan two and plan three. This is because for some runs of plan three, where battery usage was higher than expected, the branching plan was able to switch to a simpler plan, namely plan two, and complete the plan successfully. Although the reward achievable by following plan two is lower than that of plan three, it is significantly higher than the penalised value received for failure to complete plan three. In addition to this, the branching plan is able to capitalise on additional rewards when the resource usage of a run is lower than expected. For example, the initial resources might suggest that following plan two is optimal. However, if during plan execution the resource usage during the first few actions is lower than expected, the branching plan is able to switch to plan three

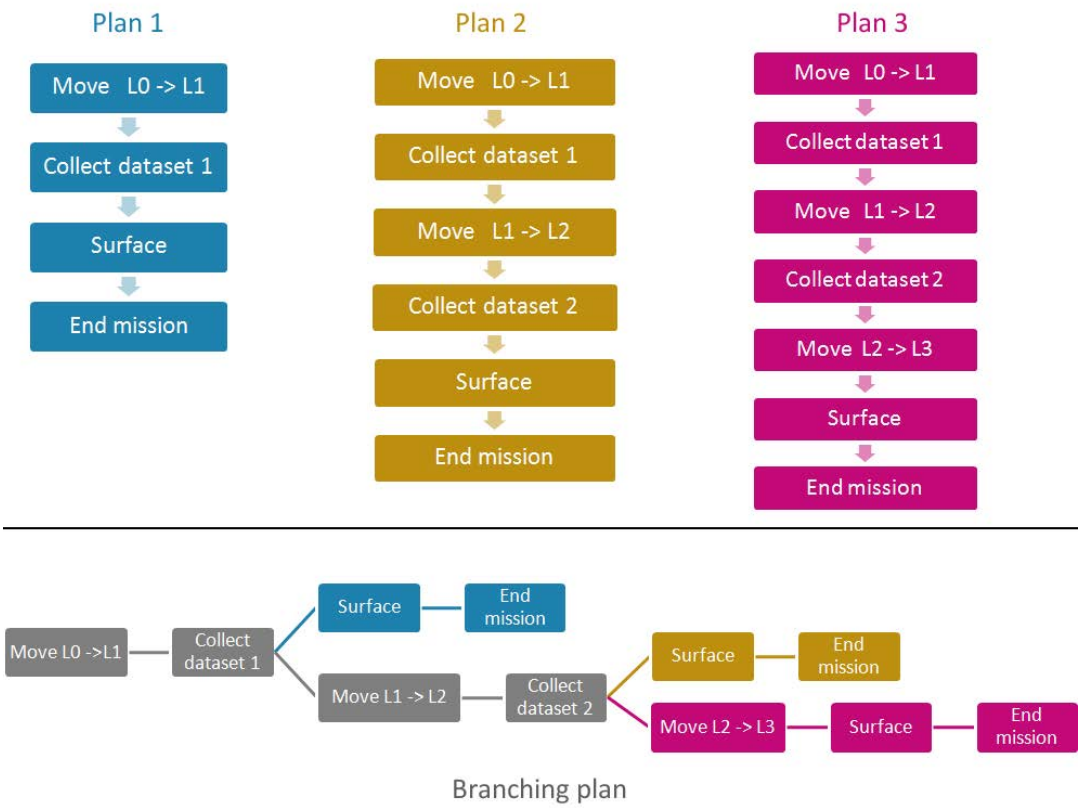

Figure 4.4: Block diagram of three plans from the small AUV problem showing how they can be combined as a branching plan. In the branching plan, the initial actions (starting from the left) are common to multiple plans and shown in grey, while action sequences unique to each plan are coloured separately. The branching plan can switch between plans at the branch points.

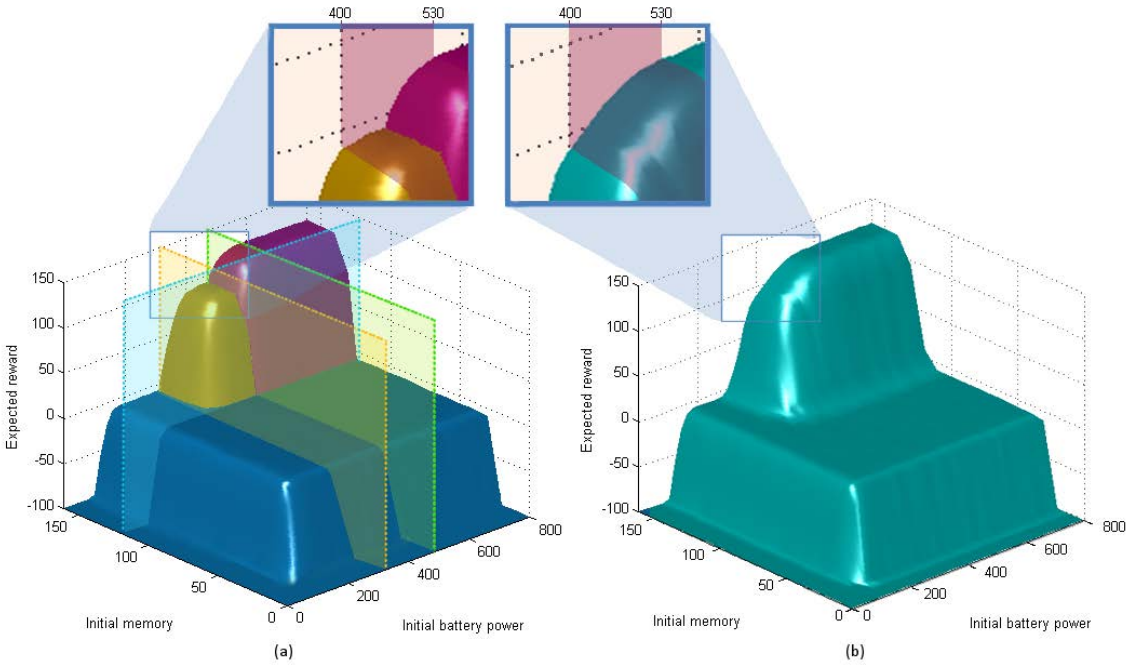

Figure 4.5: Diagram comparing (a) the optimal value function of the three straight-line plans outlined in Figure 4.4 (blue for plan one, yellow for plan two and pink for plan three), with (b), that produced when switching between the three plans when combined as a branching plan. The branching plan is shown to be at least as good as the straight-line plans, significantly outperforming all plans between 400 and 530 battery, as highlighted. The dashed green plane indicates the battery level at which plan three outperforms plan two; the dashed yellow plane indicates the battery level at which plan two outperforms plan one; and the dashed blue plane indicates the memory level below which plan one outperforms both other plans.

instead, avoiding the 30 point penalty suffered by plan two and increasing the expected reward at the end of the mission.

From this investigation we conclude that our hypothesis was correct. We observed that allowing the vehicle to change its behaviour mid-execution did indeed increase the expected reward, with the branching plan value function completely dominating that of the straight-line plans.

4.3 Extracting causal structure from a planning graph

Changing the plan during execution allows an agent to react to unexpected states. However, generating a new plan from scratch, as in the approaches discussed in Section 3.2, is computationally expensive, requiring the investment of time and battery power often without any guarantee of improving on the existing solution. In our AUV domain, where battery power used for online computation is no longer available for performing actions in the plan, it is important to keep such computation to a minimum. Offline computation, prior to the deployment of the vehicle and start of the mission, is not subject to such constraints. Inspired by the concept of plan re-use as described in Section 3.5, we considered the use of an online plan repair approach in which pre-computed plan fragments may be used to modify the existing plan in response to unexpected states.

Our initial research investigated the possibility of extracting graph fragments from a large planning graph representing the overall planning problem. Each fragment, or subgraph, comprises the actions and predicate instances required to complete a single goal. If resource availability is sufficiently high, these may be searched at run-time to extract a sub-plan which achieves the goal from the current state. Actions from this sub-plan may then be combined into the existing plan.

Using a predicate-based representation such as GraphPlan's planning graph (Blum and Furst, 1997) has advantages over state-based representations, such as MDPs. Whilst we established in Section 4.1 that the size of the state space is very large in our AUV

domain, we found the number of ground predicate instances to be significantly lower, thus reducing the complexity of the problem. The worst-case number of ground predicate instances in the AUV domain (assuming that all locations are connected to every other and including negated predicates) can therefore be calculated as follows:

$$|predicate\_instances| = 9D + 2L + (\frac{L}{2}(L-1)) + 4 \tag{4.7}$$

where D is the number of datasets and L the number of visitable locations. There are nine predicates (including negations) which are a function of the number of datasets, D; two which are a function of the number of locations, L; plus four which represent $mission\_ended(auv)$, $on\_surface(auv)$ and their negations. The remaining term calculates the number of $is\_neighbour(l_i, l_j)$ instances; assuming the worst-case in which all locations are connected to every other.

The worst-case number of predicate instances for a problem with 50 locations and 50 datasets is 1779, a tiny fraction of the number of distinct states, 6.34×10^{31} (see Section 4.1). As in the MDP case, this equation ignores the presence of continuous variables (e.g. battery and memory) which would need to be considered if this representation were to be used as part of a solution.

In order to combine graph fragments at run-time, we would first need to be able to extract the causal structure, ordering and binding constraints from GraphPlan's planning graph. To investigate this, we solved a set of simple planning problems by hand using both the UCPOP (Penberthy and Weld, 1992) and GraphPlan (Blum and Furst, 1997) algorithms. We then compared the causal links from UCPOP with the planning graph and found that each causal link was represented within the graph structure. A converged hand-constructed planning graph to achieve two goals: *data_transmitted(D1)* and *data_transmitted(D2)*, is shown in Figure 4.6. N.B. This graph does not represent stochastic action effects or resource costs.

As a proof-of-concept, we designed and implemented an algorithm, closely related to

GraphPlan, to extract fragments of the graph prior to execution. These sub-graphs could then be searched and re-used within the current plan at run-time. Our implementation modified an existing Java implementation of GraphPlan by Saigol (no date) and was used with permission.

Starting with an initial layer comprising only the selected goals, our algorithm adds all actions from the final level of the complete graph that have an effect equal to the selected goals. The preconditions of these actions are then added to the next level of the graph, along with all literals and actions from the first level. This process continues to build the graph backwards from the goals until the graph converges, i.e. the actions and literals in the current level of the graph are identical to those in the previous level. An example sub-graph, to achieve the goal $data\_collected(auv, d2)$, is shown in Figure 4.7. The construction of the sub-graph is very close to that of the original GraphPlan algorithm, although our algorithm builds the sub-graph backwards from the goals in its entirety, rather than iteratively as part of the search for a solution. As we already have the final level of the complete graph, we use the action mutexes present in this final level to define the action mutexes at each level of the sub-graph. In the same way as GraphPlan, we then mark two literals as mutex if either: one literal is a negation of the other or, all actions which create the two literals are themselves mutex.

At run-time, the graph is searched (backwards from the goals, as in the original GraphPlan algorithm) to find a plan which will achieve the new goal given the current state (if the graph has converged, all of the reachable states in the original problem definition are represented). The search algorithm required significant modification from that used by GraphPlan as our algorithm searches the whole sub-graph, rather than iteratively increasing the number of levels until a solution is found, as in GraphPlan. Our initial evaluation found sub-graph extraction and search to be slow and memory intensive in practice, requiring a comparable amount of time to replanning from scratch with GraphPlan. Consequently, we decided to move away from the idea of extracting and re-using parts of a planning graph at run-time.

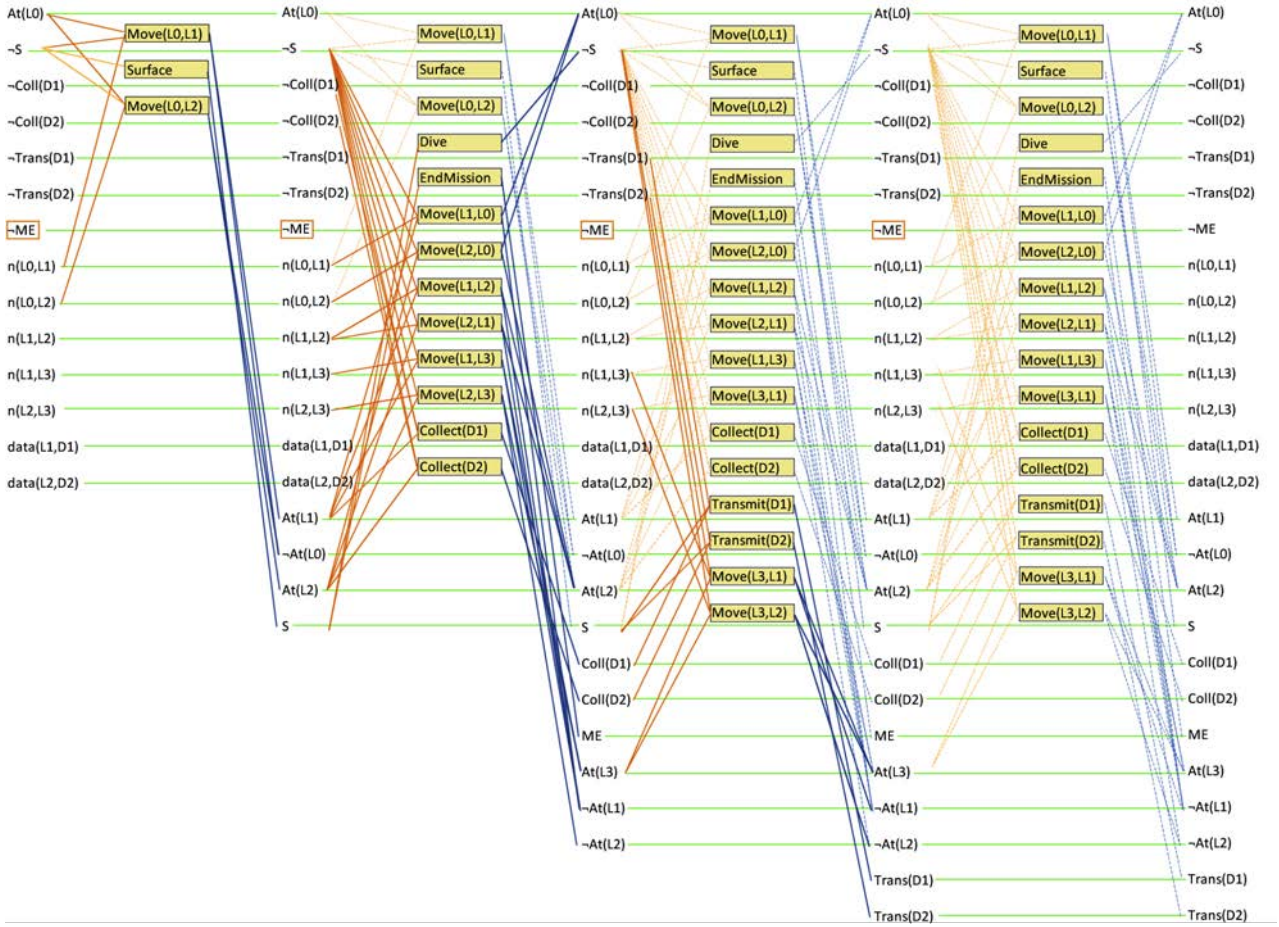

Figure 4.6: A hand-constructed planning graph for the small version of the AUV problem shown in Figure 4.1. Maintenance conditions are shown in green, preconditions in orange and effects in blue. Those which are repeated from a previous level of the planning graph are shown as dashed lines. The $not(mission\_ended)$ predicate (not(ME)) is a precondition of every action and has not been individually represented in the graph. All actions are mutually exclusive due to conflicting pre-conditions and/or effects. As this is true for all actions, mutexes have also not been explicitly represented in the graph.

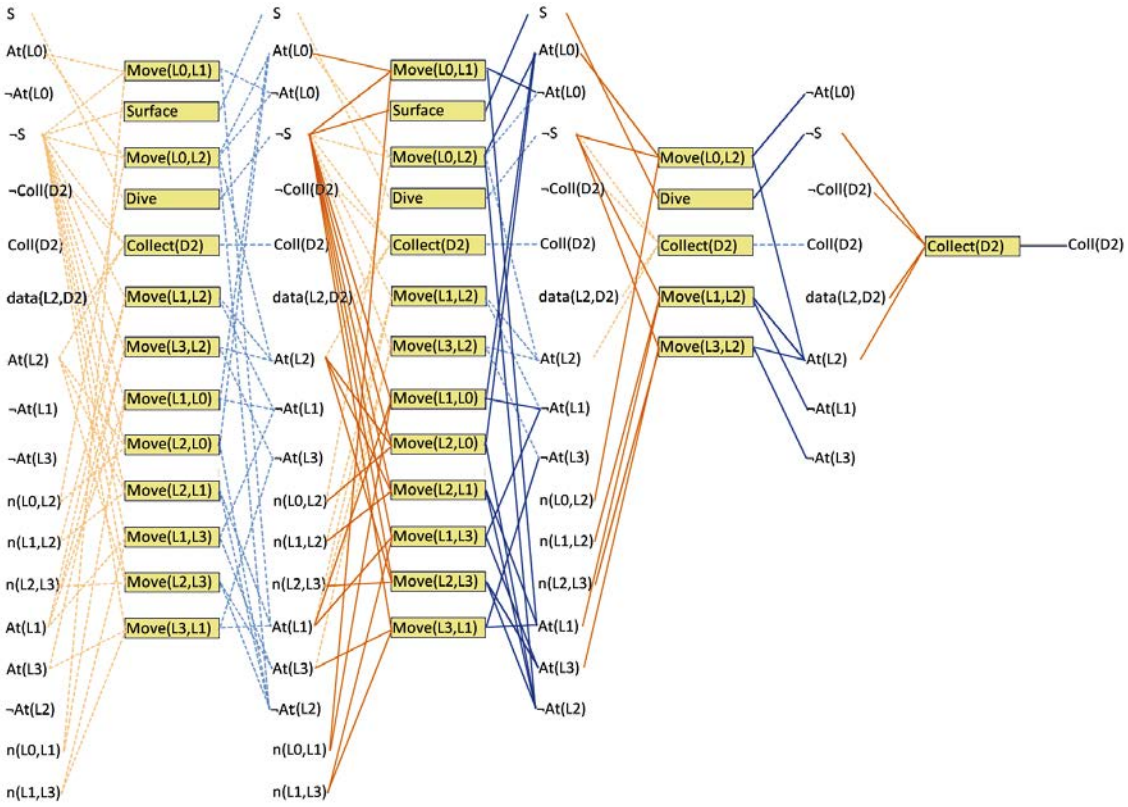

Figure 4.7: Converged sub-graph which achieves the goal $data\_collected(auv, d2)$ (Coll(D2)). This graph was produced by applying our algorithm to the final level of the graph in Figure 4.6. Preconditions are shown in orange and effects in blue. Mutexes and maintenance conditions are not shown.

Prior to starting this investigation, we hypothesised that successfully combining actions from multiple plan fragments to produce valid plans may require large amounts of domain knowledge. However, upon investigating this hypothesis and experimenting with both hand-drawn planning graphs and the modified GraphPlan implementation, we found that all of the causal structure needed to combine plan fragments could be computed from the preconditions and effects as defined in the domain action schema. Whilst our planning graph extraction and search algorithm was found to be computationally expensive, an online plan modification approach which extracts causal structure from the action schema remains an attractive solution within the AUV domain.

CHAPTER 5

DEVELOPMENT OF AN ONLINE PLAN MODIFICATION ALGORITHM

As discussed in Section 4.1.1, the large state space associated with problems in our AUV domain means that using an MDP solver to produce a policy is computationally infeasible. In Section 4.3, we also discussed the possibility of using a modified classical planning algorithm such as GraphPlan (Blum and Furst, 1997) to extract planning graph fragments, each achieving a single goal, for potential inclusion at run-time. However, the performance of our prototype extraction and search algorithm was slow, comparable to replanning with GraphPlan from scratch. As planning is often an expensive task where the benefits obtained by investing time and computation are difficult to quantify in advance, we devised an alternative approach which seeks to maximise reward whilst minimising online plan generation and computation.

5.1 High-level overview

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 branch points, the vehicle 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 scientific data re-

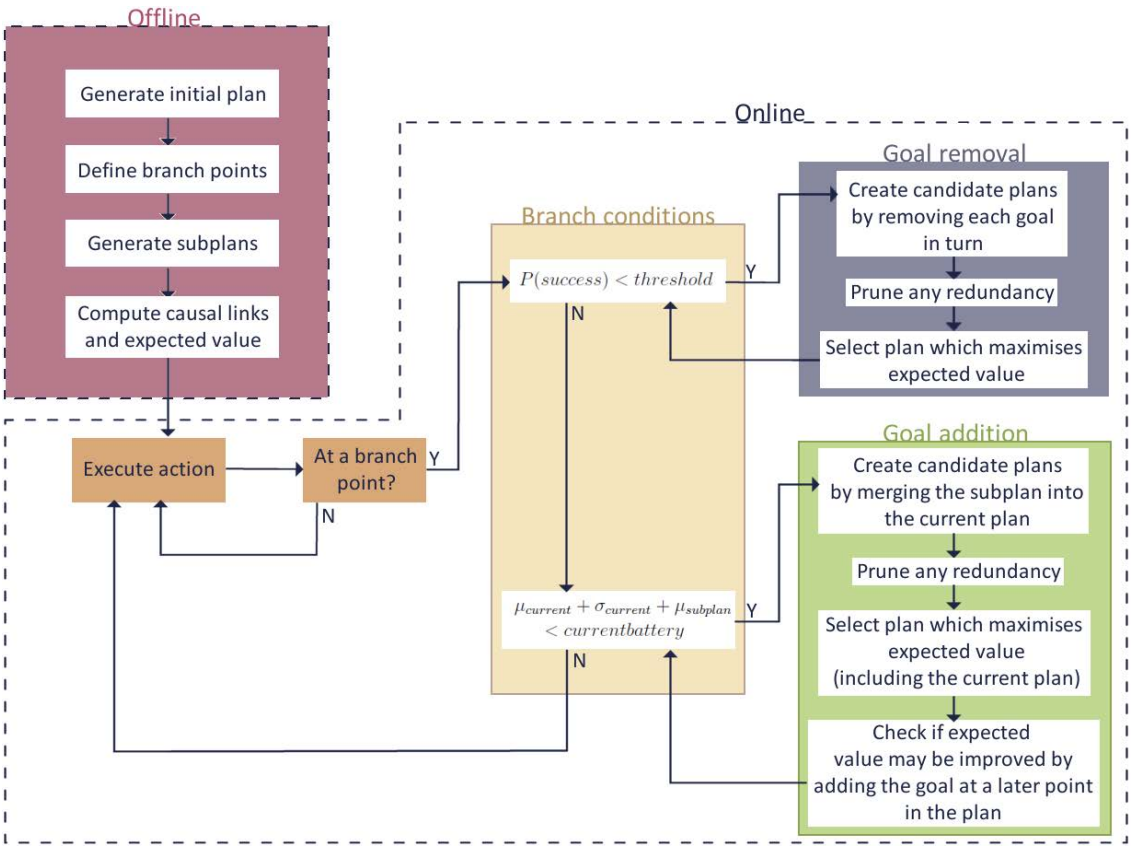

Figure 5.1: A high-level overview of our full approach, illustrating the ordering of key components within the system. The box labelled 'Offline' represents tasks which are performed prior to execution. These are discussed in Section 5.3. All other tasks occur during execution. Parts of the system relating to goal removal (see Section 5.6) and goal addition (see Section 5.7) are also labelled. The central box represents the branch conditions for both goal removal (see Section 5.6.1) and goal addition (see Section 5.7.1).

turn 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 an acceptable level, removing existing goals could potentially avert plan (and thus mission) failure, reducing risk to the vehicle.

A schematic of our full approach is presented in Figure 5.1 and the key elements are discussed in detail in this chapter. Instead of adopting an online planning approach, such as those discussed in Section 3.2, our approach pre-computes individual plan fragments for each goal in the over-subscribed problem to assist with updating the plan during execution. To combine these fragments at run-time, we have developed novel algorithms, presented in detail in Sections 5.6.2 and 5.7.3, which examine and exploit the causal relationships between actions in the plan and the goal set, a method inspired by concepts from classical planning. 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. The derivation of this calculation and the formula itself are given in Section 5.4.2.

By combining 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.

A note on terminology

In this chapter we refer to three different problems:

1. *The overall problem* — The complete over-subscribed problem, which includes all possible goals.

2. *The initial problem* — The subset of the overall problem goal set which was used to construct the initial plan.

3. *Multiple sub-problems* — Used for generating plan fragments, each achieving a single goal within the overall problem.

Throughout the remainder of this thesis we use terms relating to 'branches' and 'branch points' in relation to the sub-plans used in our approach and the points where these may be included in the plan. However, it should be noted that while there are similarities between our 'branches' and those used in other areas of planning, especially contingency planning, our definitions of these terms are not the same as when these terms are used in other areas of planning.

5.2 Simulation environment

Whilst the ultimate motivation of this work is to improve the mission planning capabilities of operational AUVs, the implementation and testing of our system on a physical vehicle is outside the scope of this thesis. To this end, we have implemented a stochastic simulation alongside our system which allows current resource availability to be 'observed'. The simulation comprises a representation of the current state. When an action in the plan is 'executed', the simulation checks the preconditions and updates the state according to the action's effects. The majority of effects, such as moving to a new location, are deterministic. However, the exact amount of resources required to perform each action is not known prior to action execution. Consequently, when executing each action, the simulation updates the state variables representing the resource by an amount sampled from the resource usage distributions of that action. The updated state is then fully observable by our plan modification and execution monitoring system.

The simulation does not model the time taken to execute each action as this is not represented in our AUV domain. Consequently, we model action execution as instantaneous. The simulation also models the chance of total catastrophic failure during each

action (as described in Section 2.5). By sampling a user specified distribution representing the probability of such a failure occurring, the simulation may mark a mission as having failed, preventing the execution of any further actions. The failure is then observable by our plan modification and execution monitoring system which halts the run.

It is important to note that we are not performing execution monitoring in the conventional sense as used by many in the planning and scheduling community. Whilst our system monitors the current state and resource availability, the execution monitoring component does not react to discrepancies between the planned state and observed state. Instead, we use the observed resource availability to estimate the likelihood of completing the current plan and the possibility of including additional goals at the next branch point.

5.3 Pre-computation

In this section, the offline components of the system are presented and the methodology is discussed. Prior to execution, an initial plan is generated and branch points are defined. At each branch point the initial plan is then augmented with plan fragments, each achieving an individual goal from the overall problem.

5.3.1 Initial plan generation

Whilst we do not directly address the issue of initial plan generation in this thesis, as it occurs independently of our system, plan repair-based approaches, such as ours, alter an existing plan, thus making the quality of the eventual solution inevitably linked to the quality of the initial plan. It is for this reason that we briefly discuss some of the considerations relating to initial goal selection and plan generation in this section.

When considering over-subscribed domains, it is necessary to select the initial goal set before initial plan generation may occur. If well-suited to the domain, an over-subscription planner such as OPTIC (Benton et al., 2012) may be used for both goal selection and plan generation. However, due to factors such as the size and complexity of the domain,

this may not always be feasible. In the field of AUV science, AUV mission scripts are traditionally hand-constructed by a skilled operator (Pebody, 2007). This is reflected in the Mars Rover literature (Bresina et al., 2002) where, whilst plan generation may be automated, each script is rigorously checked by a human operator before it is sent to the rover for execution.

As our work is motivated by such applications, we assume that the initial plan generation (and goal set selection) occurs prior to the start of the mission, when resources are plentiful and computation is not constrained. As the optimal choice of initial goals is highly dependent on the specifics of the domain (such as the requirements from scientists to collect data at a particular survey area or the geographic proximity of parcel delivery locations) we assume that the definition of both the initial goal set (along with those which may be added during execution) and the rewards associated with each goal reflect the preferences of a skilled operator.

Our approach uses a deterministic representation for the generation of a straight-line plan, treating resource usage as discrete. In this thesis, 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. For example, if a conservative initial plan is preferred, plan generation may instead assume usage equal to the mean plus one standard deviation of each distribution. Discretising the resource usage distributions allows the use of existing classical planners (such as Metric-FF (Hoffmann, 2003) and LPG (Gerevini and Serina, 2002)), avoiding the large continuous state space and consequent computational complexity which makes the use of MDP solvers infeasible for domains such as our AUV problem (see Section 4.1). After plan generation, we evaluate the resulting plan using the probabilistic model of resource usage, calculating the probability of the plan's successful execution and its expected value. This may be done using either the fast but approximate method described in this chapter (see Section 5.4) and employed by our system for online plan evaluation, or Monte Carlo simulation which is more accurate but requires significantly more computation time. By utilising external deterministic planners

for all plan generation, including the initial plan, our system allows the user to select a planner best-suited to the specifics of their domain. It is also possible that a user may specify different planners for each plan generation task. For instance, a slower, deliberative planner may be employed for initial plan generation, whilst a faster approximate approach may be preferred for the online generation of stitching plans. Indeed it is also possible that a user may decide to construct the initial plan by hand. Our system does not differentiate between methods of plan generation, instead requiring only that the plan is valid and that its probability of success exceeds the minimum threshold as defined by the user. Whilst we use Metric-FF (Hoffmann, 2003) for all plan generation within the experiments presented in this thesis, if the user wishes to use an alternative planner with minimal modifications to both our system and AUV domain, they should consider the following key requirements:

- Action costs must be supported. However, the planner does not need to support probabilistic costs as we use a deterministic version of the AUV domain when generating plans.

- Ideally, negative preconditions, metrics and conditional effects should be supported, although it is possible to modify the domain description to remove this requirement. For example, negative preconditions may be replaced with new predicates, e.g. $\neg$*on_surface(auv)* becomes *not_on_surface(auv)*.

5.3.2 Definition of branch points

Branch points should be placed at points in the plan where modifications to the goal set are likely to be the most beneficial. We quantify this 'benefit' in terms of success probability and expected reward (defined in Sections 5.4.2 and 5.4.3 respectively); the higher the expected reward of an alternative plan, the more appealing adopting this new plan appears. 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 plan/ goal set modifications. Consequently, we define branch 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. As the placement and quantity of such branch points is expected to form a key consideration in the success of our approach, we evaluated the effect of varying the number of branch points n on the achieved reward and computation required, presenting the results in Section 6.2. We hypothesise that while higher values of n are expected to increase the sensitivity and reactivity of the system, greater computation will be required both offline (to generate sub-plans at each branch point) and online (to choose the best option to take given the current resource availability).

An important point to note is that branch points 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 branch point, regardless of the size of their associated standard deviations or the value of n. To add branch points mid-execution would require increased computation as new sub-plans 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. We consider this meta-planning problem outside the scope of this thesis and instead monitor and evaluate the computational cost of our approach, with a view to minimising online computation.

5.3.3 Sub-plan generation

At each branch 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 sub-plans and the initial plan to form a plan that satisfies a set of goals. The sub-problem used to construct each fragment is defined as a PDDL file, which is generated by the system and passed to an external planner to solve. 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 branch point. This is calculated by sequentially applying the effects of each action in the plan to the state resulting from the effects of the previous action. For example, the first action in the plan is applied to the initial state before the effects of the second action are applied to the resulting state etc. This sequential simulation of the actions and states in the plan continues until the required branch point is reached. 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 in the PDDL file as simply a single goal from the overall problem. All other goals (such as those in the initial problem) are not included.

5.4 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. The plan must be valid, i.e. it must be executable and achieve all goals in the current goal set. To determine plan validity, we first check if the logical preconditions of each action are met and whether the sequence of actions satisfies the current goals. We consider resource usage separately, calculating the probability of successfully completing the plan given resource uncertainty. We then calculate the expected value of the plan as a measure of plan and goal set quality.

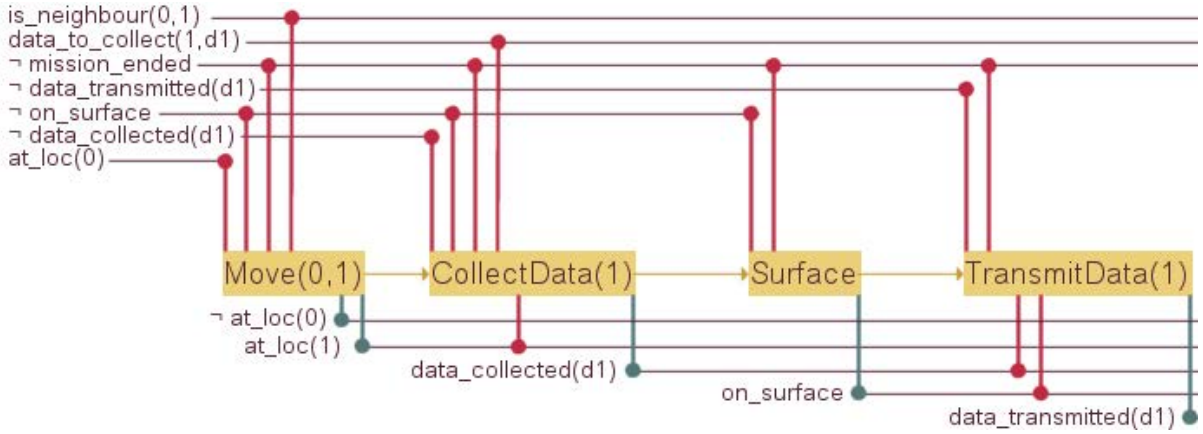

Figure 5.2: Diagram illustrating how causal relationships may be inferred using action preconditions and effects. The predicate instances in the top-left represent the initial state. These are maintained (indicated by the horizontal lines) until an action changes their value, e.g. from true to false. Red lines represent action preconditions, whilst blue lines represent effects.

5.4.1 Causal link computation

In partial-order planning, a causal link between two actions represents that a precondition of one is satisfied by another (Ghallab et al., 2004). We construct 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.

Through the use of preconditions and effects, action definitions tell us a lot about the potential causal relationships between actions. Effects represent the possible provision of a causal link while preconditions indicate where other links may be consumed. By sequentially examining the preconditions and effects of actions in a plan, we can connect these partial links together until the full causal structure of the plan is represented. This concept is illustrated in Figure 5.2.

Given the plan, the current goal set and the current state, the algorithm first creates partial causal links for all literals 'provided' by the current state, adding them to a set C. Partial causal links are represented using the structure shown below:

$$< (\textit{producer action} \vee \textit{initial state}), \textit{literal}, x > \tag{5.1}$$

where x is a number representing the index of the providing action within the plan. In the case of the initial state, $x = 0$. The algorithm then iterates over all preconditions of the first action a_1 in the plan, at index 1. Iterating backwards over the set of partial causal links C, if a precondition requires a literal l that is provided by a causal link $c_i \in C$, a_1 is added as a 'consumer' of c_i at index 1 and the search moves onto the next precondition of a_1. If an action requires the opposite of l, e.g. $\neg l$, we also break and consider the next precondition. While this may seem unnecessary when extracting the causal structure from a complete valid plan (as we would not expect to encounter such a case) it becomes important when merging plan fragments, as the inclusion of a new action may introduce threats which prevent the completion of a partial causal link. To maintain the ordering of actions in the plan, and the impact it has on causal structure, we iterate backwards through C, breaking when the most recent consumer of a literal l is found. Once all preconditions of a_1 have been considered, the algorithm adds a partial causal link for a_1 and each of its effects at index 1. This iteration continues for a_2 and all subsequent actions in the plan. For example, in Figure 5.2, *CollectData(1)* provides the literal *data_collected(d1)* at index 2. This is then consumed by *TransmitData(1)* at index 4. It is important to note that each provider may have multiple consumers, e.g. many actions may rely on the literal $\neg$*on_surface* provided by the initial state (as shown in Figure 5.2). Following the representation used in partial order planning (Ghallab et al., 2004) we consider all goal literals as the preconditions of a dummy action, a_∞, which represents the goal state. Consequently, for each goal 'precondition' we also iterate over the causal links in C, completing provider-consumer pairs as before. Finally, we remove any causal link c_i from C if it remains a partial causal link once all actions have been examined, i.e. refers to a literal which is produced but never consumed. The set of causal links C is then returned.

5.4.2 Calculation of success probability

In order to calculate the expected value of a plan, we need to first calculate the likelihood of the vehicle achieving the goals and earning the associated rewards. We refer to this likelihood as the probability of success or $P(success)$.

As all actions consume or renew the vehicle's available resources, the probability of success is closely related to resource usage. If an action either causes the vehicle's resources to be exhausted or to fall below that required by the next action, execution will halt and the mission will fail. Resource usage of each action is modelled using Gaussian probability distributions (μ, σ), as discussed in Section 2.3. If executing an action affects both battery and memory, e.g. collecting, transmitting or delivering data, the usage of each resource is represented using separate distributions, as illustrated in Figure 5.3. For some actions, such as when transmitting a dataset, battery usage is dependent on memory usage (i.e. larger datasets require more battery to transmit). We represent such dependencies by defining the mean battery usage as a function of the mean memory required for storing the dataset. However, it should be noted that for ease of calculating the success probability and expected value of a plan, we consider the battery usage of an action to be otherwise independent from its memory usage. Consequently, the probability of success is calculated separately for both battery and memory.

As battery availability monotonically decreases, calculating $P(success)$ for battery is more straightforward than when considering renewable memory. We first discuss the calculation of success probability considering only battery, $P(success|battery)$, before extending the method to memory, calculating $P(success|memory)$, highlighting where the two approaches differ and why.

Battery

As illustrated in Figure 5.3, each action in the plan is associated with a Gaussian distribution, representing its battery consumption. The battery consumption of an individual action is assumed to be independent from that of all other actions in the plan. This is a

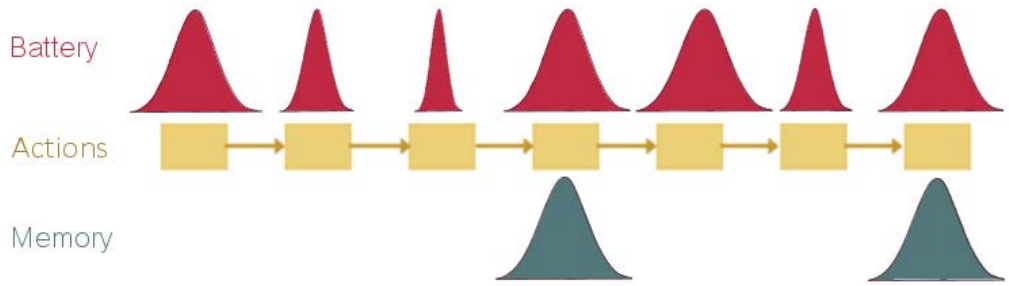

Figure 5.3: Diagram illustrating the Gaussian distributions associated with each action in a plan. Every action has a distribution representing expected battery usage, but some also have a second distribution representing expected memory usage.

valid assumption to make as while external factors such as ocean currents may possibly affect the battery consumption of multiple actions, the effect of such factors would be very difficult to accurately predict. As usage is independent, we sum the distributions for all actions into a single distribution representing the battery usage of the plan, by summing their associated means and variances. This method is as defined in the quote below, from Caswell (1989, p. 255):

> "It can be proved that if X and Y are two independent random variables with means μ_X, μ_Y, and standard deviations σ_X, σ_Y respectively, then the random variable $T = X + Y$ has mean equal to $\mu_X + \mu_Y$ and standard deviation $\sqrt{\sigma_X^2 + \sigma_Y^2}$."

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:

$$CDF = \int \left(\frac{e^{-\frac{(x-\mu)^2}{2\sigma^2}}}{\sigma\sqrt{2\pi}} \right) .dx = \frac{1}{2} \left[\text{erfc} \left(\frac{\mu - x}{\sqrt{2}\sigma} \right) \right] \tag{5.2}$$

While solving the equation of a Gaussian distribution results in the probability of sampling x usage, the cumulative distribution function represents the probability of usage being less than x. Consequently, if we solve the cumulative distribution function where x equals current battery availability and μ and σ 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, $P(success|battery)$:

$$P(success|battery) = \frac{1}{2}\left[\text{erfc}\left(\frac{\mu_T - currentBattery}{\sqrt{2}\sigma_T}\right)\right] \tag{5.3}$$

Memory

As memory is a renewable resource, the calculation of $P(success|memory)$ is more complicated. Some actions consume memory, some renew it and some do not affect memory at all (e.g. when moving between locations). As in the case of battery, the Gaussian distributions associated with actions which consume memory have a positive mean. Actions which renew memory have a negative mean and actions which do not affect memory have both a mean and a standard deviation of 0. 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, as illustrated in Figure 5.4. If we then sum the mean memory usage over a typical plan, which both collects and transmits/delivers 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 5.3, results in $P(success|memory) = 1$. This is not an accurate reflection of the true probability of plan success. Even if the mean memory usage of a collect action exceeds the amount available, $P(success|memory)$ would still equal 1 provided the dataset was transmitted and all memory restored before the end of the plan. To avoid this over-estimate of $P(success|memory)$, we instead subdivide the plan at the points where memory is renewed and calculate the probability of successfully reaching each of these points. The

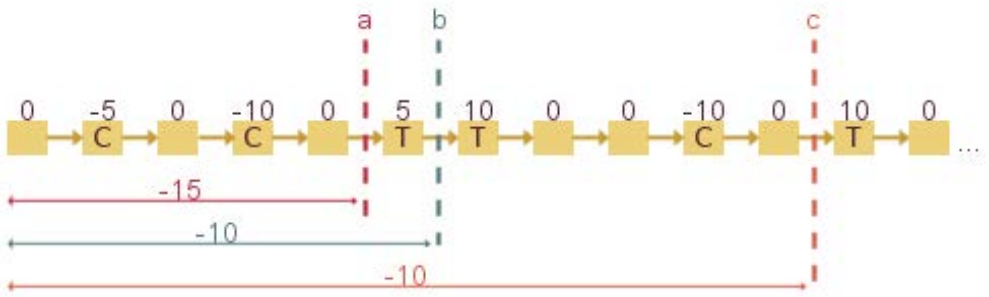

Figure 5.4: Schematic showing an example of subdividing the plan during the calculation of success probability for memory usage. The numbers above actions represent the mean of Gaussian distribution associated with each action; negative numbers represent consumption of memory when collecting datasets (C), while positive numbers represent replenishment during transmission actions (T). The plan is subdivided prior to the transmit actions, labelled a, b and c, and the cumulative mean at these points is shown.

point immediately prior to a replenishment action (e.g. *TransmitData*) was chosen because at these points, memory usage is at a local maxima. We can then treat each of these subdivisions as we treat battery, integrating the combined distribution and solving for current memory.

An example of this subdivision is shown in Figure 5.4. At the point marked a, before the first transmit action, the combined memory *usage* of the plan so far is 15. We can then sum the memory usage distributions of all actions prior to a, integrating this combined distribution to calculate the probability of having enough memory to successfully reach point a, $P(success|memory_a)$. We then consider all actions prior to point b. At this point, the summed mean memory usage is 10 and we again integrate the new combined distribution to calculate a second value, $P(success|memory_b)$, this time representing the probability of reaching point b without exceeding the available memory. This continues until $P(success|memory)$ has been calculated for all subdivisions. To then 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.

5.4.3 Calculation of expected value

A plan with a high success probability is attractive as it suggests the risk to the vehicle and mission are minimal. However, it is also important to use the available resources to maximise the collection of data and consequently the associated reward. To reason about the balance between risk and reward, we define a metric called *expected value*.

To calculate the expected value of a plan, the success probability is weighted by the reward:

$$E[v] = \sum_{x=1}^{n} \left[\left(P(success|battery_x) \times \prod_{q=1}^{x} P(success|memory_q) \right)^2 \times reward(x) \right] \quad (5.4)$$

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 5.4 as the product term. Considering only the most recent subdivision would give us the probability of success over only the actions within this subdivision. To instead calculate an estimate of success probability over the whole plan, we must also consider the probability of successfully reaching the current subdivision. 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 expected value, which is then used as a metric when comparing plans.

5.5 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 expected value of the plan are recalculated. When at a branch 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 expected value 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 vehicle, 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 by the vehicle but this is potentially at the cost of mission success. To balance the trade-off between risk and reward, for experiments and analysis

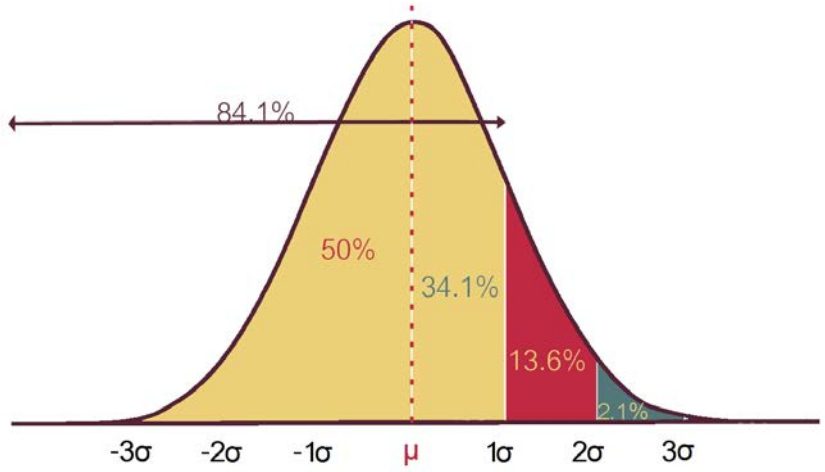

Figure 5.5: Diagram illustrating the probability of sampling values which fall within each standard deviation (σ) of a Gaussian distribution. In a two-tailed distribution, there is a 68.2% chance of the value falling within 1σ of the mean (μ). However, we are only interested in the single-tailed case 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.

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 (see Figure 5.5), 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.

5.6 Goal Removal

5.6.1 When to consider removing a goal

At each branch point, if the success probability of the current plan (either for battery or any subdivision of memory) falls below the minimum threshold, goal removal is considered. However, following on from considerations presented during the calculation of $P(success|memory)$ in Section 5.4.2, as memory is renewable we have to again consider the success probability of each memory subdivision individually. If we were instead to compare the product of all subdivisions to a minimum *P(success)* threshold of 0.841, as

Algorithm 1 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

defined for our AUV problem, a product equal to 0.841 does not necessarily follow the mean plus one standard deviation assumption (shown in Figure 5.5). For example, if three subdivisions of memory each have a success probability of 0.841, this implies the current memory availability is equal to the mean plus one standard deviation requirement. However, the product of the three sections (i.e. 0.841^3) is 0.595 which vastly underestimates the true value of $P(success|memory)$ for the plan.

By reducing the number of goals, and thus the number of actions and the resource consumption of a plan, the algorithm increases the probability of successfully completing the plan. Each goal in the current goal set is then individually passed to the goal removal algorithm which removes the goal and produces an alternative plan.

5.6.2 Goal removal algorithm

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. To detect and remove such redundant actions, we examine the causal structure of the plan (see Section

5.4.1).

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 examines the causal structure of the plan until no more actions may be removed — see Algorithm 1.

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.

5.6.3 Removing redundant actions

Following the removal, or addition, of a goal and associated actions, the resulting plan may contain newly redundant actions. For example, the following plan achieves the goals: *data_collected(auv1, d1)*, *data_collected(auv1, d2)* and *at_loc(auv1, l1)*.

o_1: `move(auv1, l0, l1)`

o_2: `collect_data(auv1, l1, d1)`

o_3: `move(auv1, l1, l2)`

o_4: `collect_data(auv1, l2, d2)`

o_5: `move(auv1, l2, l1)`

o_6: `surface(auv1)`

o_7: `end_mission(auv1, l1)`

After the execution of the first action, *move(auv1, l0, l1)*, the success probability of the plan falls below the required threshold. In response, the system removes the goal *data_collected(auv1, d2)*, resulting in the following plan:

o_1: `move(auv1, l0, l1)`

`- - - current state - - -`

o_2: `collect_data(auv1, l1, d1)`

o_3: `move(auv1, l1, l2)`

o_5: `move(auv1, l2, l1)`

o_6: `surface(auv1)`

o_7: `end_mission(auv1, l1)`

In this new plan, actions o_3 and o_5 (*move(auv1, l1, l2)* and *move(auv1, l2, l1)*, respectively) are now redundant, consuming resources without furthering the completion of any goals. While it may seem obvious to the reader that neither o_3 nor o_5 are required within the resulting plan, the action o_5, *move(auv1, l2, l1)*, is unrelated to the removed goal and instead provides for the goal of ending the mission at location one, *at_loc(auv1, l1)*. For this reason, it is not considered for removal. Action o_3 does produce an effect (*at_loc(auv1, l2)*) that is required by the removed action o_4 and is thus considered for removal. However, this literal is also required by action o_5, which prevents the removal of o_3 from the plan.

To avoid such redundant behaviour and the waste of vehicle resources in such scenarios, following the removal of a goal any redundant actions are detected and removed from the plan. The algorithm detects redundant actions 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 and are therefore redundant. Following the removal of the redundant actions, the resultant plan used in this example, and shown below, is now shorter, requires less resources to execute and thus has a higher success probability and expected value.

o_1: `move(auv1, l0, l1)`

`- - - current state - - -`

o_2: `collect_data(auv1, l1, d1)`

o_6: `surface(auv1)`

o_7: `end_mission(auv1, l1)`

5.6.4 Goal selection

Once a list of alternative plans has been produced, the algorithm analyses each one, selecting the best plan and thus the goal to remove.

To perform this analysis, any plans that violate any of the resource constraints are first declared invalid and discarded. The success probability of each remaining candidate is then compared to the minimum threshold and any plans which do not exceed it are also discarded. Of the plans which exceed the threshold, the plan with the highest expected value after goal removal is selected. This will inevitably be less than that of the current plan. However, whilst the system removes a goal to increase $P(success)$ at the cost of reward, it is important that we still seek the maximum reward (and thus expected value) once $P(success)$ has been considered. If none of the candidate plans exceed the $P(success)$ threshold, the system removes the goal which results in the plan with the highest expected value. 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 vehicle in resource constrained domains and ensures it operates within the user-defined constraint of risk, as defined by the minimum $P(success)$ threshold. In the context of an AUV deployment, such a scenario would initiate emergency routines, for example by dropping an abort weight and floating to the surface.

5.7 Goal addition

5.7.1 When to consider adding a goal

At each branch point, following consideration of the goal removal criteria (see Section 5.6.1), the algorithm iterates over all candidates for goal addition (excluding any goals which have just been removed and may not be re-added at the same branch point) to

re-allocate any surplus resources thus maximising reward. These are goals which are part of the overall problem but are not in the current goal set (and thus not achieved by the current plan). For each candidate goal g at the current branch point, the algorithm checks whether the mean battery requirement of the sub-plan associated with g, pf, is less than the total amount available when combined with the expected battery usage of the current plan $(\mu_{current}, \sigma_{current})$. The algorithm considers adding a goal if:

$$\mu_{current} + \sigma_{current} + \mu_{pf} < currentBattery \tag{5.5}$$

The same check is not performed for memory. As memory is renewable, the current memory available to the vehicle will differ depending on where the new plan fragment is included within the current plan. Instead, we check the validity of the plan following the merge, ensuring that all constraints, including those on memory, are satisfied. Once the set of candidate goals G ($g \in G$) has been pruned to remove those which do not meet the resource criteria (see Equation 5.5), the states of the sub-plan are updated to include any additional literals from the current state (at the branch point), such as datasets which have already been collected. The states and causal links of the current plan are also passed to the merge algorithm, along with the plan fragment pf and the remainder of the current plan p.

5.7.2 Development of a plan merging algorithm

In its simplest form, merging two plans is a process which combines the actions from both plans into one, creating a new plan capable of satisfying the goals of both. We proposed and investigated two possible methods for merging plans in this way by:

1. Inserting one plan into another as a contiguous block or *macro-action*.

2. Interleaving actions from both plans.

Inserting plan fragments as a macro-action

The first option, inspired by van der Krogt and de Weerdt (2005) and He et al. (2010), was to insert the sub-plan as a single contiguous block or *macro-action*. This involves finding a point in the current plan at which the sub-plan may be inserted without introducing new threats to any causal links in either plan. Using the notion of a threat as defined in partial order planning (Ghallab et al., 2004, p. 92), we consider the inclusion of a new action a at index i to threaten an existing causal link c_i if an effect of a is the exact negation of the literal represented by c and i is within the range of the causal link, i.e. between the actions/states which produce and consume the literal. A threatened causal link may prevent existing preconditions from being met, resulting in an invalid plan. To avoid introducing threats, the state at which the sub-plan is inserted must satisfy the preconditions of the first action in the sub-plan, whilst the preconditions of the first action following the sub-plan must continue to be met. For example, if we take the original plan shown below with the goal *data_transmitted(auv1, d1)*:

o_1: `move(auv1, l0, l1)`

o_2: `collect_data(auv1, l1, d1)`

o_3: `move(auv1, l1, l2)`

o_4: `surface(auv1)`

o_5: `transmit_data(auv1, d1)`

o_6: `end_mission(auv1, l2)`

and execute the first three actions before attempting to insert the following sub-plan, with the goal *data_collected(auv1, d2)*, as a macro-action:

pf_1: `collect_data(auv1, l2, d2)`

We can merge two plans successfully into the valid plan ordering shown below:

o_1: `move(auv1, l0, l1)`

o_2: `collect_data(auv1, l1, d1)`

o_3: `move(auv1, l1, l2)`

`- - - current state - - -`

pf_1: `collect_data(auv1, l2, d2)`

o_4: `surface(auv1)`

o_5: `transmit_data(auv1, d1)`

o_6: `end_mission(auv1, l2)`

As the current state meets the preconditions of action pf_1, we can simply insert the new action at the current state. Although actions o_4, o_5 and o_6 are from the original plan, inserting the new action prior to this point does not threaten any existing causal links. For example, action o_5, to *transmit_data(auv1, d1)*, requires ($\neg$*mission_ended*, *on_surface(auv1)*, *data_collected(auv1, d1)*) as preconditions of its execution. None of the effects of the newly inserted action (pf_1) change the value of these state variables. In the style of 'no-op' or 'frame' actions from GraphPlan (Blum and Furst, 1997) (which exist purely to maintain the truth of a predicate instance between multiple levels of a planning graph), we can think of these required predicates as maintained until at least action o_5. However, such a simplistic merge strategy is not suitable in every situation, given different combinations of actions. For example if we attempt to combine the same original plan as in the example above, using the same current state, with a new sub-plan:

pf_1: `collect_data(auv1, l2, d2)`

pf_2: `surface(auv1)`

pf_3: `transmit_data(auv1, d2)`

which satisfies the goal *data_transmitted(auv1, d2)*, our simplistic strategy is no longer capable of creating a valid plan. This is because the *surface* action is present in both the original plan and the new plan-fragment. If the new plan was inserted as a whole, the vehicle would be on the surface upon finishing the execution of the new plan fragment, causing a threat to the causal link consumed by the *surface* action from the original plan, o_4. More specifically, the $\neg$*on_surface(auv1)* precondition of action o_4 is no longer met, resulting in a plan which is not valid and thus is unexecutable. There are two ways to resolve this problem:

1. either we recognise that one of the two *surface* actions is redundant when the plans

are merged and remove it from the plan, essentially combining both transmit actions into a single trip to the surface, for example:

o_1: `move(auv1, l0, l1)`

o_2: `collect_data(auv1, l1, d1)`

o_3: `move(auv1, l1, l2)`

`- - - current state - - -`

pf_1: `collect_data(auv1, l2, d2)`

pf_2: `surface(auv1)`

pf_3: `transmit_data(auv1, d2)`

o_5: `transmit_data(auv1, d1)`

o_6: `end_mission(auv1, l2)`

where the action o_4 has been skipped;

2. or we add in an extra dive action to 'stitch' the two plans together.

Adding in a dive action resolves the threat to the existing causal link by making $\neg$*on_surface(auv1)* true again, thus satisfying the preconditions of action o_4, as shown below (actions added as part of a *stitching plan* between the two plans are labelled s_x):

o_1: `move(auv1, l0, l1)`

o_2: `collect_data(auv1, l1, d1)`

o_3: `move(auv1, l1, l2)`

`- - - current state - - -`

pf_1: `collect_data(auv1, l2, d2)`

pf_2: `surface(auv1)`

pf_3: `transmit_data(auv1, d2)`

s_1: `dive(auv1)`

o_4: `surface(auv1)`

o_5: `transmit_data(auv1, d1)`

o_6: `end_mission(auv1, l2)`

The stitching plan was constructed using the goal state of the sub-plan as the initial state and the unsatisfied preconditions of the rest of the plan as its goal. We initially used the state prior to the insertion of the sub-plan as the goal state (i.e. the current state). By ensuring the goal state of the inserted plan is the same as the current state, the stitching plan returns the state of the vehicle to that prior to the insertion of the sub-plan, thus implicitly resolving any threats introduced by actions within the sub-plan. However, whilst this definition of the goal state produces valid plans, in many situations it can lead to sub-optimal results. This is because some predicates in the current state may not be required by the remaining actions in the plan (e.g. the location of the vehicle may be irrelevant). This can cause stitching plans to contain redundant actions. By defining the goal state of the stitching plan as the unsatisfied preconditions of the remainder of the plan, only actions which are required in order to perform the rest of the plan are included in the stitching plan.

However, by treating the plan fragment as a macro-action it is still possible for redundant actions to remain. In the example above, s_1 and o_4 are redundant — the vehicle dives only to immediately return to the surface. We can remove such redundancy using the method presented in Section 5.6.3 but, by recognising that one of the two *surface* actions is redundant and skipping its inclusion, we save computation and arrive at the same resulting plan.

There are situations where neither the plan fragment nor the existing plan contain the actions required to produce a valid combined plan. For instance, in the previous example, if we were to instead introduce a goal to transmit dataset $d3$ from location $l3$, new actions will be required to move the vehicle from $l3$ back to $l2$. This is because the $end\_mission(auv1, l2)$ action, o_6, has a precondition requiring the vehicle to be at the end location $l2$. None of the actions in either the plan fragment or the existing plan would have such an effect and so a stitching plan must be generated.

Interleaving actions from both the sub-plan and the current plan

As an alternative to the 'macro-action' approach of inserting the sub-plan as a single contiguous block, we investigated an approach which allowed actions from both plans to be interwoven. For example, if we take the plan shown below:

o_1: `move(auv1, l0, l1)`

o_2: `collect_data(auv1, l1, d1)`

o_3: `move(auv1, l1, l2)`

o_4: `surface(auv1)`

o_5: `transmit_data(auv1, d1)`

o_6: `end_mission(auv1, l2)`

and introduce the goal *data_transmitted(auv1,d2)* after the fourth action, the new sub-plan is:

pf_1: `dive(auv1)`

pf_2: `collect_data(auv1, l2, d2)`

pf_3: `surface(auv1)`

pf_4: `transmit_data(auv1, d2)`

Combining the two plans results in three possible merges:

Ordering 1/3:

o_1: `move(auv1, l0, l1)`

o_2: `collect_data(auv1, l1, d1)`

o_3: `move(auv1, l1, l2)`

o_4: `surface(auv1)`

`- - - current state - - -`

pf_1: `dive(auv1)`

pf_2: `collect_data(auv1, l2, d2)`

pf_3: `surface(auv1)`

pf_4: `transmit_data(auv1, d2)`

o_5: `transmit_data(auv1, d1)`

o_6: `end_mission(auv1, l2)`

Ordering 2/3:

o_1: `move(auv1, l0, l1)`

o_2: `collect_data(auv1, l1, d1)`

o_3: `move(auv1, l1, l2)`

o_4: `surface(auv1)`

`- - - current state - - -`

pf_1: `dive(auv1)`

pf_2: `collect_data(auv1, l2, d2)`

pf_3: `surface(auv1)`

o_5: `transmit_data(auv1, d1)`

pf_4: `transmit_data(auv1, d2)`

o_6: `end_mission(auv1, l2)`

Ordering 3/3:

o_1: `move(auv1, l0, l1)`

o_2: `collect_data(auv1, l1, d1)`

o_3: `move(auv1, l1, l2)`

o_4: `surface(auv1)`

`- - - current state - - -`

o_5: `transmit_data(auv1, d1)`

pf_1: `dive(auv1)`

pf_2: `collect_data(auv1, l2, d2)`

pf_3: `surface(auv1)`

pf_4: `transmit_data(auv1, d2)`

o_6: `end_mission(auv1, l2)`

In the first ordering, the sub-plan is inserted as a 'macro-action' at the current state, but this is not the case in orderings 2/3 and 3/3, with actions from both plans inserted

where causal links and ordering constraints allow. Interleaving actions does not affect the ordering constraints of either plan provided that the actions of each plan remain in the same order. Also of note in ordering 3/3 is that the plan transmits the existing dataset ($d1$) before diving again and collecting dataset $d2$. Orderings 1/3 and 2/3 instead group the two transmissions together at the end of the plan. This results in redundancy between actions o_4 and pf_1. However, as the vehicle has already executed o_4, it is impossible to remove the redundancy from the plan using the method presented in Section 5.6.3. As the AUV domain includes a continual risk of vehicle loss (see Section 2.5), ordering 3/3 is likely to be the plan with the highest expected reward because the reward for transmitting $d1$ is achieved much earlier in the plan.

5.7.3 Goal addition algorithm

The methods for plan merging presented in the previous section were formalised into an algorithm that is capable of achieving new goals by combining existing plans.

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, shown in Algorithm 2, takes the first action of the plan fragment pf and compares its preconditions with each state in the original plan, 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

Algorithm 2 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 S 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

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, we skip the action as it may not be required when the two plans are combined (see example in Section 5.7.2).

If no valid plan orderings are found using Algorithm 2, a *stitching plan* is generated which uses the goal state of the sub-plan 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 sub-plan. After generation, the stitching plan is concatenated onto the end of the plan fragment pf and the newly extended pf is passed to Algorithm 2 to be merged into the current plan.

5.7.4 Goal selection

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

Firstly, the success probability is calculated for each merged plan. If $P(success|battery)$ or $P(success|memory)$ for any subdivision falls below the acceptable threshold (defined as 0.841 for the AUV domain — see Section 5.5), the plan is discarded and pruned from the set. Calculating the success probability evaluates the plan given the continuous variables and resource uncertainty. Of the plans which exceed the $P(success)$ threshold, the plan with the highest expected value is chosen. If none of the new plans has a success probability greater than the *P(success)* threshold or an expected value which exceeds that of the current plan, the new plans are discarded and execution continues with the current plan.

5.7.5 Delaying goal addition

Once the goal to add has been selected, the algorithm considers the possibility that adding the goal at a later branch point, instead of at the current state, may increase the expected value of the 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 higher expected reward, as illustrated in Figure 5.6. Gough et al. (2004, 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". However, as Gough et al. consider a full branching plan with a relatively small number of contingencies, their algorithm can calculate and compare the utility of taking each opportunity within the entire plan. As we have a far larger number of possible contingencies and thus cannot represent all contingencies as a branching plan (see Section 3.3 and Appendix C), we

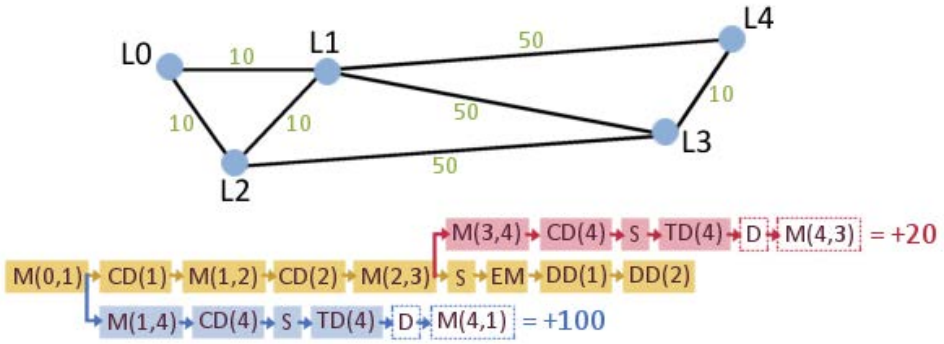

Figure 5.6: Schematic illustrating how delaying the addition of a goal may be beneficial. The graph shows the connectivity of locations in a small problem and the distances between them. The current plan is shown in yellow. By including a new goal to transmit dataset 4 (TD4) at the current state, the move actions (M) of the resulting plan (shown in blue) 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 (shown in red) 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*.

instead employ a greedy approach for delaying the addition of a goal.

When considering whether to delay the addition of the selected goal, our algorithm selects any subsequent branch points, within the current plan, where there exists a plan fragment for the goal. A list of valid merged plans is constructed for each branch point and the list of merges is then pruned according to the $P(success)$ threshold, as before. This time, the expected value of merging the new goal at a later branch 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 branch points, selecting a goal to add at a particular branch point from the full set. However, as battery power used for computation is 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 branch point would instead increase the expected value 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 (see Section 5.7.1) or the expected value of the merged plans fails to exceed that of the current plan.

CHAPTER 6

NUMERICAL EVALUATION

In this chapter, we present the methodology and results of four experiments that evaluate key parts of our approach.

6.1 Methodology and test problems

Four problems were defined, each relating to the same topology of 20 survey locations shown in Figure 6.1, with 20 associated datasets (i.e. each of the four problems is a different subset of the same *overall problem*). Problems one and two both have an initial goal set of five goals, where each goal represents a specific dataset to be returned (e.g. *data_with_scientists(d)*). 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. Three levels of resource availability were defined, *low* (L), *medium* (M) and *high* (H). Defining absolute 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. For example, while

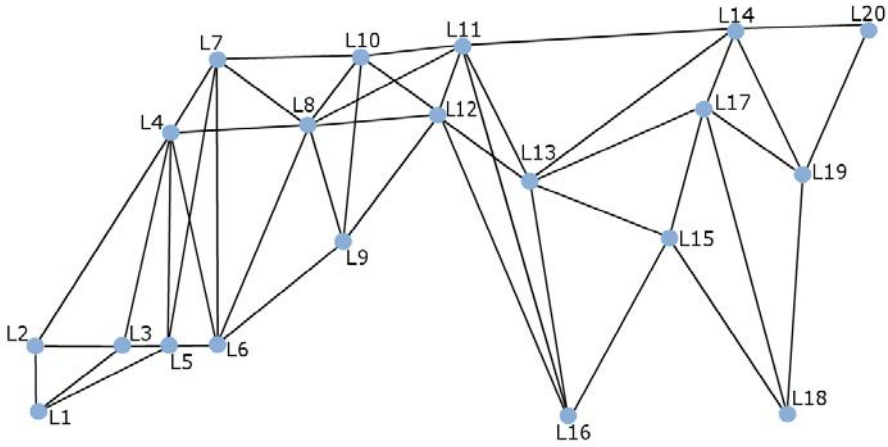

Figure 6.1: Connectivity of locations and datasets in the overall problem. Circles represent data-collection opportunities.

a value of x battery may be low for a problem a, it may provide a surplus for a second problem, b. 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 largest memory requirement (mean plus standard deviation) of all actions in the initial plan.

  As memory is renewable, summing the mean and standard deviations of expected memory usage of all actions (as in the case of battery) would result in a large over-estimation of the memory required to execute the plan (as discussed in Section 5.4.2). In the *low* case, as resource availability represents the bare minimum required to execute the plan, initial memory is defined as sufficient to enable collection of the largest dataset. Therefore, the initial memory is defined as the mean plus the associated standard deviation of the action with the highest expected usage.

Medium resource availability was then defined as 110% of *low* and *high* was defined as 120% of *low*.

As the uncertainty in the domain is significant and the resource usage of each action is sampled, it is likely that a direct comparison of the results would not be fair. For example, it would not be fair to compare the results of a run in which the sampled resource usage was very high, to one in which it was very low. Equally, as reward values are also sampled, it would not be a fair test to compare vastly different rewards for collecting the same dataset. To remedy this disparity and facilitate a fair comparison, both between tests 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 B, the second run will use a second set, and so on. These sets of fixed values were then used by all experiments. Using the same sampled values also allows the comparison of individual results, for example evaluating statistical significance using a test for paired samples.

While these considerations greatly improved the fairness of the experiments, it was difficult to design a perfect methodology due to the significant variability and uncertainty inherent in both the AUV domain and our approach. 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.

6.2 Determining the quantity of branch points

The first experiment was designed and performed to evaluate the effect of varying the number of branch points on the reward achieved and on the probability of successfully completing the mission. The experiment compared five different levels of branch points, defined as percentages of the total number of actions in the initial plan. The number of branch points was varied from 0%, representing a fixed straight-line plan, to 100%

which permits branching after any action in the plan. The effect of defining branch points after 20%, 50% and 80% of actions in the initial plan was also evaluated. For each problem/resource combination, the experiment was repeated 50 times. The sampled values for resource usage and reward were fixed for each run of 0%, 20%, 50%, 80% and 100% branching, allowing us to directly compare performance.

6.2.1 Analysis of results

The experiment was performed and the results compiled. These are shown in Figures 6.2 - 6.12. 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. The analysis of reward was therefore treated separately to that of success rate. As reward is heavily penalised for failed runs, the average reward would be significantly affected by the presence of failed runs. For example, a group of runs may encounter an adverse scenario, such as low resource availability (caused by the sampled usage being higher than expected). This may cause 0% branching runs to fail, whilst those with branching opportunities may be able to recover. If only the failed 0% run was removed, this would unfairly disadvantage those runs which removed a goal (sacrificing reward) to increase the chance of mission success. Consequently, when evaluating average reward, if a run failed, the corresponding group of runs for all levels of branching was removed, thus performing case-wise deletion in the presence of missing runs. For example, if the second repeat of problem 3L 0% failed, all results for the second repeat of 3L were ignored. While this means that some results are removed from the dataset when evaluating average reward, they are still incorporated within the dataset when analysing success rate.

6.2.2 Success rates

The success-rates achieved for each problem are shown in Figure 6.2 and the averages plotted in Figure 6.3. Over all problems, the inclusion of branching (i.e. 20-100%) out-

Problem Number	Resource level	Success rates achieved by varying the percentage of branch points				
		0	20	50	80	100
	L	82	88	98	96	100
1	M	98	96	96	98	98
	H	94	94	94	96	92
	L	88	94	100	100	100
2	M	94	92	92	90	90
	H	100	98	94	98	98
	L	88	98	96	100	96
3	M	98	96	94	98	96
	H	98	96	96	96	98
	L	76	96	96	92	96
4	M	94	92	94	92	92
	H	96	96	94	98	98
	Mean	92.2	94.7	95.3	96.2	96.2
	S.d.	7.3	2.9	2.1	3.2	3.2

Figure 6.2: Table showing the average success rates of all runs and their relation to the mean for each problem. The spectrum of colours correspond to the magnitude of the values, where red indicates high success rates and blue, the lowest. The 50th percentile is the mid-point on the spectrum, coloured yellow.

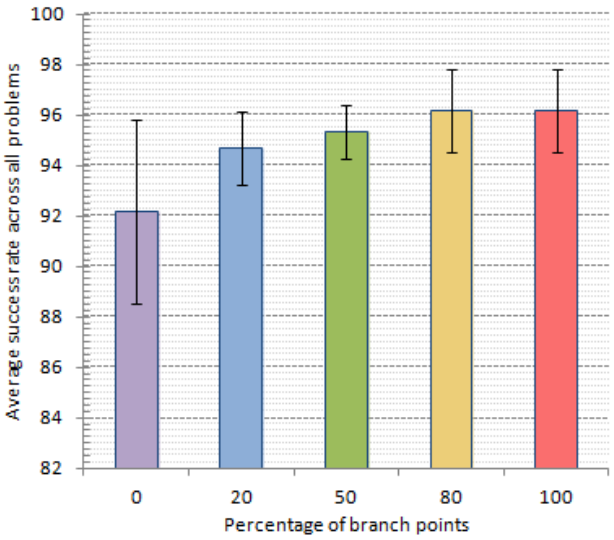

Figure 6.3: Graph showing the average success rate, across all problems, for each level of branching, as tabulated in Figure 6.2.

performed the straight-line, 0% branching case, achieving an average of 95.6% compared to 92.2%. The difference in performance achieved by each level of branching was found to be significantly better than that of 0% branching (within the 5% significance level, using a chi-square test (Caswell, 1989)).

When resources were *low*, 0% branching results in a larger number of failures. This is to be expected as, with no opportunities for branching, the system is unable to remove goals (and associated actions) to increase the chance of mission success. Given the initial resources for the *low* case are defined as the mean plus one standard deviation of the initial plan, we would expect a success rate of approximately 84.1% (see Section 5.5). Considering only the *low* cases, the average success rate of the 0% branching runs is 83.5%. This is lower than 84.1% due to the chance of catastrophic failure during the execution of each action — arbitrarily defined for these experiments as 1/3000. If we assume a plan has 25 actions, the expected success rate including the probability of catastrophic failure is reduced to 83.4%. For a plan with 50 actions, it is reduced to 82.7%. Also only considering *low* runs, 20-100% branching achieved an average success rate of 96.63%, significantly higher than that of 0% branching (found to be within the 5% significance level, using a chi-square test (Caswell, 1989)).

The success rate for 0% branching 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. Considering higher levels of branching, the average success rate remained consistent with that achieved when resource availability was *low* (the difference in average success rate between the *high* and *low* cases was not found to be statistically significant).

The success rates for runs using between 20% and 100% branching are very close, with only 1.5% separating them — a difference which was not found to be significant, using a chi-square test (Caswell, 1989). The standard deviations also show that there is little to distinguish these runs in terms of variation.

In summary, when resources are plentiful, the presence of branch points has little effect on the success rate achieved. However, when resources are constrained, the branching plan is able to avert failure, avoiding the reduction in success rate suffered by the straight-line plan. These results confirm our earlier hypothesis that modifying the plan during execution improves the success rate of a mission, decreasing the chance of potential vehicle loss.

6.2.3 Average reward

The table in Figure 6.4 shows the average reward achieved for different amounts of branching. The results show that branching allows a plan to take advantage of additional opportunities for reward when resources are plentiful (in the *medium* and *high* cases), while sacrificing reward to prioritise successful mission completion when resources are constrained (in the *low* case).

When resource availability was *medium* or *high*, branching performed, on average, 7.9% better than when 0% branching was used. This difference was found to be statistically significant at the 5% level, using the Wilcoxon signed rank test for paired samples (Wilcoxon, 1945). This was to be expected as 0% branching does not allow for any modifications to be made to the initial plan and so these runs are unable to use surplus resource to achieve additional goals. However, when resource availability was *low*, the trend was reversed; branching achieved, on average, 7.7% lower rewards than when 0% branching was used (also found to be statistically significant at the 5% level, using the Wilcoxon signed rank test for paired samples). In the *low* case, resource availability is heavily constrained. In such situations, branching allows goals to be removed, sacrificing reward to increase the probability of mission success. A straight-line plan (0% branching) is unable to make such updates and so the higher average in the *low* case indicates that during some runs, the branching strategies removed goals, reducing the available reward. It is important to remember that we are only considering the successful runs (see Section 6.2.1). If we were also to include failed (which suffer a large penalty) when calculating

		Average reward (and lower and upper quartile) achieved when varying the percentage of branch points														
Problem Number	Resource level	0			20			50			80			100		
		1st quartile	mu	3rd quartile	1st quartile	mu	3rd quartile	1st quartile	mu	3rd quartile	1st quartile	mu	3rd quartile	1st quartile	mu	3rd quartile
1	L	1956.59	1962.73	1968.00	1949.41	1982.67	2020.74	1970.45	1960.33	2007.07	2003.22	2035.85	2069.76	1868.78	1912.81	2021.35
	M	1957.85	1964.65	1970.61	2072.85	2151.55	2232.58	2121.28	2180.27	2256.36	2121.28	2182.17	2260.90	2121.28	2182.17	2260.90
	H	1960.53	1964.85	1968.72	2240.57	2252.05	2297.76	2252.34	2271.46	2302.02	2240.76	2275.42	2320.78	2238.52	2273.04	2322.65
2	L	1454.06	1458.45	1462.00	1364.34	1375.29	1382.67	1411.24	1410.84	1430.18	1414.76	1423.21	1430.18	1414.76	1423.21	1430.18
	M	1453.94	1458.27	1462.65	1516.94	1548.49	1576.93	1520.79	1560.48	1623.89	1521.03	1560.76	1624.30	1520.79	1559.44	1624.59
	H	1451.76	1457.10	1461.05	1618.71	1644.81	1680.00	1605.32	1622.24	1639.65	1609.25	1615.77	1636.99	1603.54	1612.22	1633.67
3	L	2626.56	2634.04	2639.10	2404.17	2400.48	2506.93	2343.79	2374.62	2399.26	2332.04	2369.29	2386.59	2355.39	2421.57	2498.92
	M	2625.46	2630.59	2635.19	2732.75	2736.35	2831.86	2651.10	2669.14	2814.55	2651.10	2670.31	2814.55	2507.70	2693.96	2832.41
	H	2626.86	2632.39	2636.98	2770.35	2822.29	2853.13	2760.72	2739.25	2824.65	2760.72	2750.08	2824.65	2760.72	2750.08	2824.65
4	L	1897.33	1920.60	1955.07	1540.02	1571.86	1625.13	1510.74	1565.78	1605.17	1542.24	1552.08	1562.72	1543.75	1559.53	1570.31
	M	1929.04	1935.25	1945.79	1934.80	1960.41	2026.92	1949.79	2014.65	2083.39	1923.06	1985.68	2058.31	1923.06	1988.92	2083.40
	H	1938.92	1944.69	1949.73	2050.52	2075.74	2142.23	2052.55	2104.02	2244.59	2053.70	2105.62	2244.59	2052.55	2149.80	2245.09

Figure 6.4: Table listing the average reward *mu* achieved when varying the percentage of branch points from 0% (indicating a straight-line plan) to 100%. The spectrum of colours correspond to the magnitude of the rewards for each problem (and are calculated for each row). Red indicates the greatest reward for that problem, whilst blue highlights the lowest. The 50th percentile is the mid-point on the spectrum, coloured yellow.

Problem Number	Resource level	Value compared to 0% branching for each problem						Percentage value improved on 0% branching				
		0	20	50	80	100		0	20	50	80	100
	L	0.0	19.9	-2.4	73.1	-49.9		0	1.02	-0.12	3.73	-2.54
1	M	0.0	186.9	215.6	217.5	217.5		0	9.51	10.97	11.07	11.07
	H	0.0	287.2	306.6	310.6	308.2		0	14.62	15.61	15.81	15.69
	L	0.0	-83.2	-47.6	-35.2	-35.2		0	-5.70	-3.26	-2.42	-2.42
2	M	0.0	90.2	102.2	102.5	101.2		0	6.19	7.01	7.03	6.94
	H	0.0	187.7	165.1	158.7	155.1		0	12.88	11.33	10.89	10.65
	L	0.0	-233.6	-259.4	-264.7	-212.5		0	-8.87	-9.85	-10.05	-8.07
3	M	0.0	105.8	38.6	39.7	63.4		0	4.02	1.47	1.51	2.41
	H	0.0	189.9	106.9	117.7	117.7		0	7.21	4.06	4.47	4.47
	L	0.0	-348.7	-354.8	-368.5	-361.1		0	-18.16	-18.47	-19.19	-18.80
4	M	0.0	25.2	79.4	50.4	53.7		0	1.30	4.10	2.61	2.77
	H	0.0	131.0	159.3	160.9	205.1		0	6.74	8.19	8.28	10.55
	Average	0.0	46.5	42.5	46.9	46.9	Average	0.00	2.56	2.59	2.81	2.73
	S.d.	0	187	190	193	189	S.d.	0.00	9.47	9.60	9.69	9.61

Figure 6.5: Table showing the relation between the reward achieved (shown in Figure 6.4) and the reward achieved by 0% branching. These values are then presented as a percentage improvement over the 0% case. The spectrum of colours correspond to the magnitude of the values, where red indicates the highest values and blue, the lowest. The 50th percentile is the mid-point on the spectrum, coloured yellow.

average reward, 0% branching would not outperform higher levels of branching in the *low* case.

Based on the absolute values in Figure 6.4, there is not a clear trend in performance to distinguish the cases which include branching (i.e. 20-100%). To investigate the branching cases further, the average reward for each problem was calculated as the percentage improvement over the 0% case, as this provides an appropriate base-line. These relative average rewards are tabled in Figure 6.5 and plotted in Figure 6.6. An overall average may then be calculated for each level of branching, facilitating fair comparison across all problems (see Figure 6.7). Of the branching cases, 80% and 100% slightly outperform 20% and 50% on relative average reward, although the difference is small, compared to the increase over 0% branching, and was not found to be statistically significant.

From these overall averages, we can conclude that any amount of branching increases the overall average achieved reward above that of 0% branching. This supports the results of the preliminary investigation, described in Section 4.2. If we also include failed runs (whose achieved reward is subject to a penalty), shown in Figure 6.8, the trend is the same.

We then considered the size of the interquartile range (see Figure 6.9). As 0% branch-

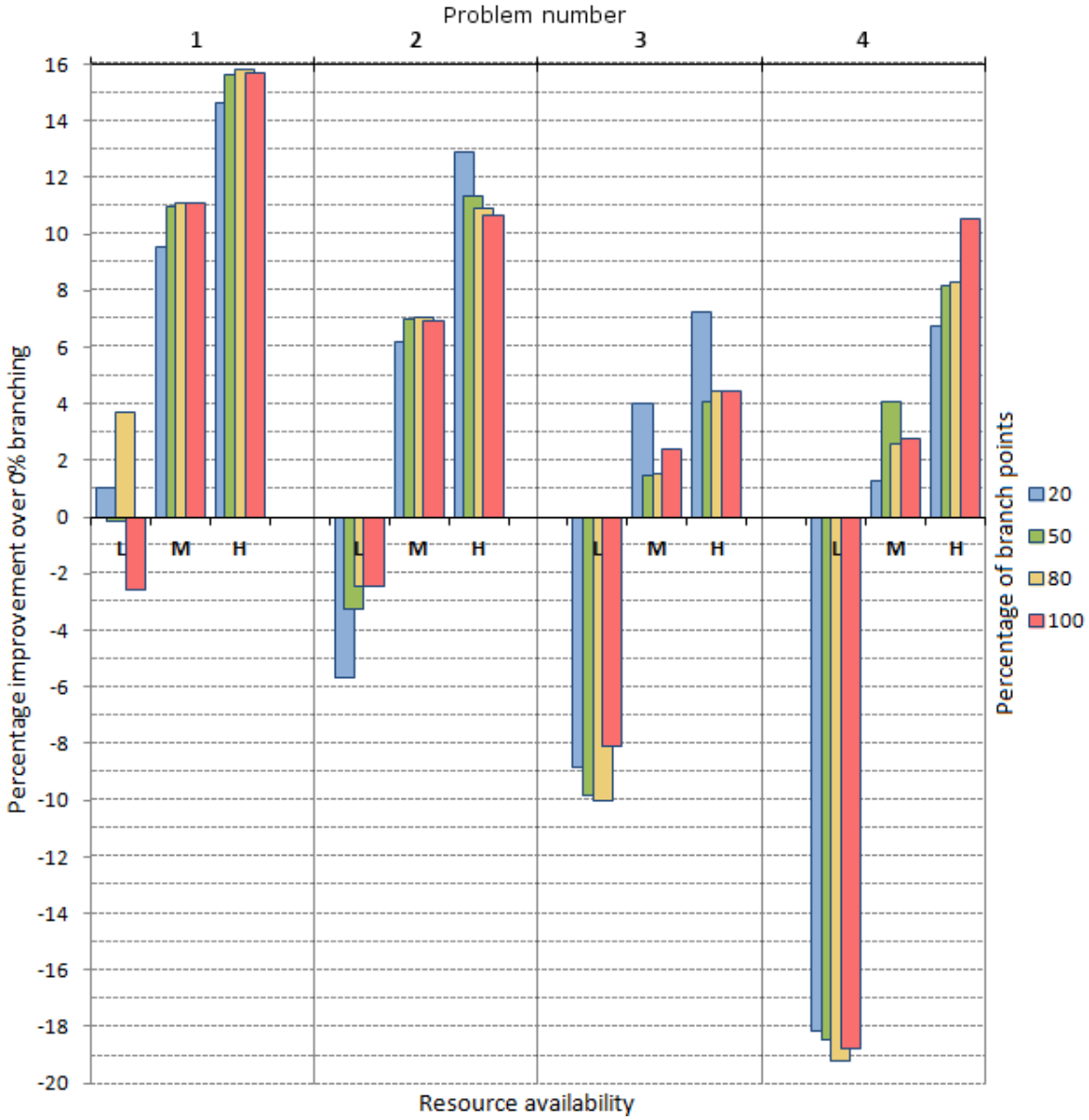

Figure 6.6: Graph showing the percentage improvement in achieved reward over that of 0% branching, for each problem.

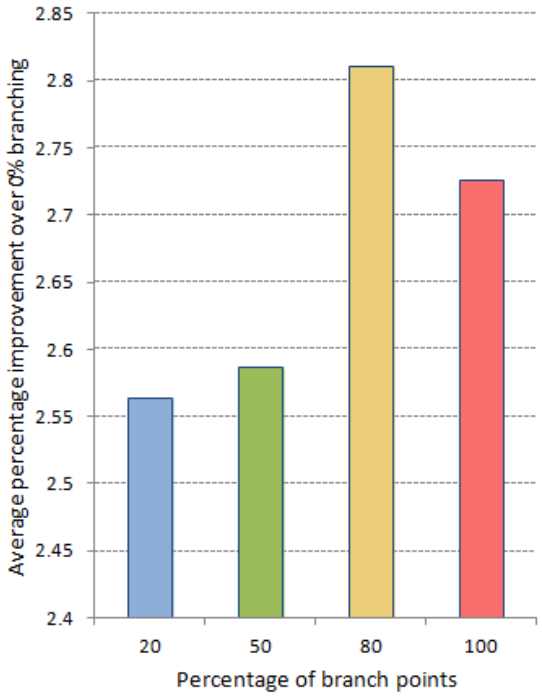

Figure 6.7: Graph showing the average percentage improvement in achieved reward over that of 0% branching.

Problem Number	Resource level	Average reward (including penalty for failed runs) as a percentage of that of only successful runs				
		0	20	50	80	100
	L	75.58	83.77	96.96	96.02	99.81
1	M	97.30	95.48	94.44	97.13	97.13
	H	92.43	93.89	92.58	94.70	89.64
	L	78.00	90.07	98.26	99.54	99.54
2	M	90.73	87.88	88.16	84.77	85.34
	H	99.98	97.10	90.97	94.66	97.29
	L	87.27	97.43	95.37	99.51	94.02
3	M	98.00	95.90	93.93	98.04	95.32
	H	97.43	96.30	98.36	98.38	98.90
	L	60.34	94.41	94.63	88.58	94.12
4	M	91.61	89.89	91.61	89.79	89.97
	H	93.93	94.84	92.33	97.50	97.24
	Average	88.55	93.08	93.97	94.89	94.86

Figure 6.8: Table showing the average reward achieved when including penalties for failed runs, as a percentage of that achieved when considering only successful runs. The spectrum of colours correspond to the magnitude of the values, where red indicates the highest rewards and blue, the lowest. The 50th percentile is the mid-point on the spectrum, coloured yellow.

Problem Number	Resource level	Interquartile range					
		0	20	50	80	100	100
	L	11	71	37	67	153	153
1	M	13	160	135	140	140	140
	H	8	57	50	80	84	84
	L	8	18	19	15	15	15
2	M	9	60	103	103	104	104
	H	9	61	34	28	30	30
	L	13	103	55	55	144	144
3	M	10	99	163	163	325	
	H	10	83	64	64	64	64
	L	58	85	94	20	27	27
4	M	17	92	134	135	160	160
	H	11	92	192	191	193	193
	Average	15	82	90	88	120	101
	S.d.	14	34	56	58	87	61

Figure 6.9: Table showing the interquartile ranges associated with the average rewards presented in Figure 6.4. The additional column for 100% branching shows the effect of removing the anomalous value for problem 3M on the average interquartile range. The spectrum of colours correspond to the magnitude of the values, where red indicates large interquartile ranges and blue, the lowest. The 50th percentile is the mid-point on the spectrum, coloured yellow.

ing results in the execution of a straight-line plan, regardless of resource availability, there is predictably very little variation in the rewards for each problem. The variability of 20, 50 and 80% branching is more comparable, especially between 50 (average interquartile range = 90) and 80% branching (average interquartile range = 88) which are very similar. The variability associated with 100% branching is significantly higher than that found for 20, 50 and 80% branching. Whilst the large average interquartile range for 100% branching (mean 120, standard deviation 87) is somewhat skewed by the outlying interquartile range for problem 3M, if this value is excluded, the average interquartile range for 100% branching still exceeds that of the other levels of branching (mean 101, standard deviation 61, as shown in Figure 6.9). This anomalous result may suggest that, in this case, the system was overly reactive, updating the plan more than was optimal. The first quartile associated with 100% branching problem 3M (see Figure 6.4) is significantly lower (170.62 on average) than for the other levels of branch point definition (i.e. 20, 50 and 80%) for this problem, which would suggest over-conservative behaviour. We analysed the individual runs for problem 3M in an attempt to learn more about this anomaly. When the rewards for 3M 100% branching are arranged in order of size and compared to the rest

of the results for 3M, the only results which are significantly different are the 11th, 12th and 13th runs (when considered in increasing size) which are approximately 100 lower, the rest are very similar to those for problem 3M when using different levels of branching. It is therefore unfortunate that it is these anomalous values which are used to calculate the 1st quartile and thus the interquartile range. We believe that the algorithm selected a slightly different goal-set during these runs.

In summary, where branching was used, the average achieved reward was found to reduce when resources were constrained. This was expected as branching allows a vehicle to sacrifice reward to prioritise success rate. However, when resources were more plentiful, the average rewards achieved using branching were 7.9% higher than achieved by the straight-line plan. This shows that branching allows a vehicle to use surplus resources to capitalise on additional rewards.

6.2.4 Computation time

As battery used by the vehicle 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 the experiments presented in this thesis were performed in simulation, it was not possible to measure the battery usage directly. Instead, computation time was measured to approximate computational cost. The execution time of each action is negligible in the simulation, as the simulation assumes all actions are executed instantaneously.

It was challenging to accurately measure the computation time as the system was written in Java. It is not possible to control the point at which Java performs 'garbage collection' and this consequently may sporadically increase the computation time required to perform a particular run. In an attempt to reduce the effect of this random bias, each data-point is 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 also 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 (Hoffmann,

2003). In our experience, if Metric-FF has to find a solution using best-first search, it 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 sub-plan and stitching plan. Due to the simplicity of sub and stitching plan problems, ten seconds is more than sufficient for Metric-FF to complete enforced hill-climbing.

Figure 6.10 shows the average total computation time for each problem in the test set (averaged across all resource levels) and Figure 6.11 shows the average computation time for each level of branching across all problems. Each bar is then subdivided to show the average proportion of time spent on both online and offline computation. The computation times for 0% branching were not included in Figure 6.10 as both the pre-computation time (including the computation of sub-plans and branching 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 and, in all problems apart from problem two, the initial plan generation time was minimal [a].

The total computation time increased with the number of branch points. This was expected because additional branches must be first generated offline prior to being merged and evaluated during execution. As all offline computation occurs prior to the deployment of the vehicle, offline computation time is not tightly constrained. However, online computation time is restricted and thus requires greater consideration. These times are presented separately for ease of comparison in Figure 6.12. As with total time, online computation time also increased with the number of branch points. If we consider the largest average online computation time, that of 100% branching for problem three, 304.1 seconds, this may initially seem quite large. However, this experiment considered action execution time to be instantaneous. If this experiment had instead been performed in

[a]The longer initial plan generation time for problem two indicates that some property of this problem made it more difficult (and thus time-consuming) for Metric-FF to solve, compared to the other problems in the test set. We could find no obvious reason why this problem should be more difficult to solve, although a full evaluation of Metric-FF's performance on this individual problem was outside the scope of this thesis.

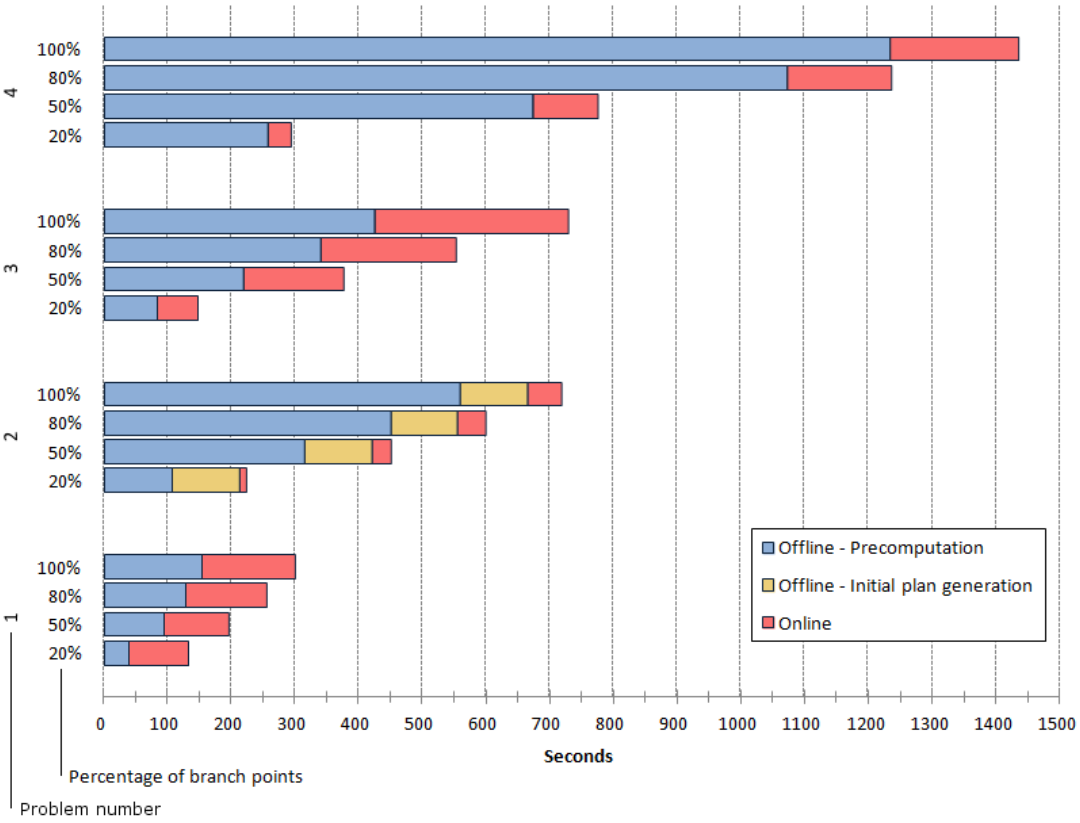

Figure 6.10: Graph showing the average total computation time for all runs.

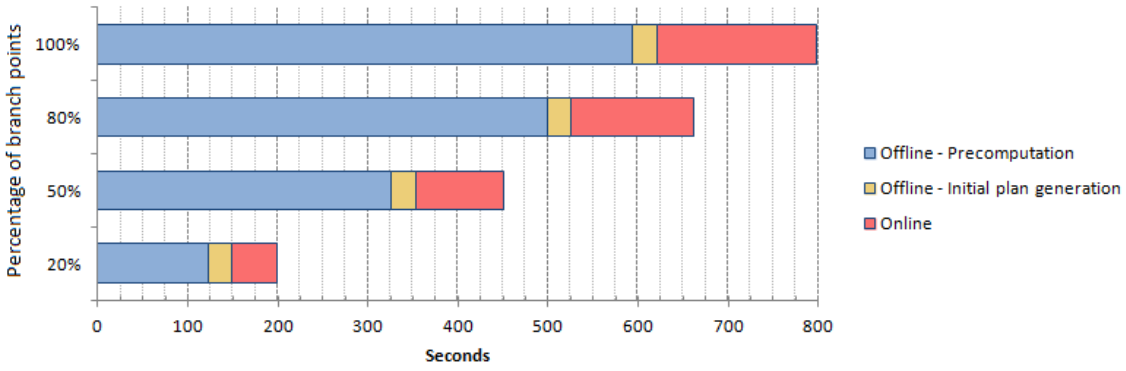

Figure 6.11: Graph showing the average computation time over all problems.

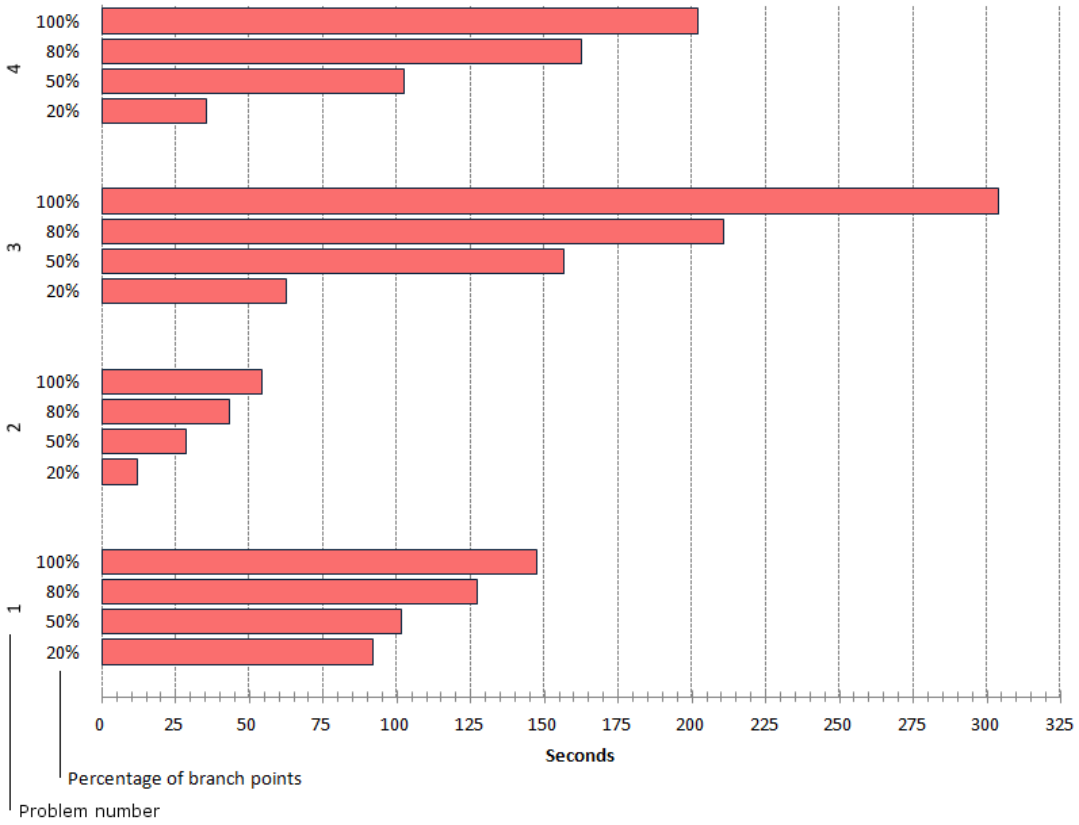

Figure 6.12: Graph showing the average online computation time for all runs.

real-time using a vehicle, a total online computation time of 5 minutes, as in problem three, 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). Further experimentation was required to directly compare these two approaches. The results are presented in Section 6.4.

6.3 Determining the branch criteria

A second experiment was performed to evaluate the branching criteria of the system, i.e. defining when the system should consider the options at a branch point and when it should instead continue the execution of the current plan. We compared our approach, described in Section 5.5, to a more straight-forward, but equally valid approach, which compares the observed resource usage (prior to the branch point) to the expected usage for this same period. We refer to this second approach as *ObservedVsExpected*. *ObservedVsExpected* is an intuitive alternative approach which considers updating the goal set if resource usage has been significantly higher or lower than expected. Conversely, our approach considers what may be possible given our current resource availability. The differences between the two approaches are summarised below:

When to consider removing a goal:

- Our approach — Consider removing a goal if the probability of successfully completing the plan, $P(success)$, for either battery or memory availability, falls below a 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 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 sub-plan (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 $Usage(memory) < \mu_{memory} - \sigma_{memory}$.

Whilst our branching criteria considers adding and removing goals in terms of 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.

In this experiment we recorded the success rate and reward achieved using the same set of four test problems (see Section 6.1) as in experiment one. Following the results of the first experiment, we fixed the number of branch points at 100%, i.e. defining a branch point after every action in the initial plan. This is because 80% and 100% branching performed well considering both success rate and average achieved reward. However, unlike 80% branching, where the distribution of branch points may vary between plans, by defining branch points after every action in the initial plan, we also 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 6.1. 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.

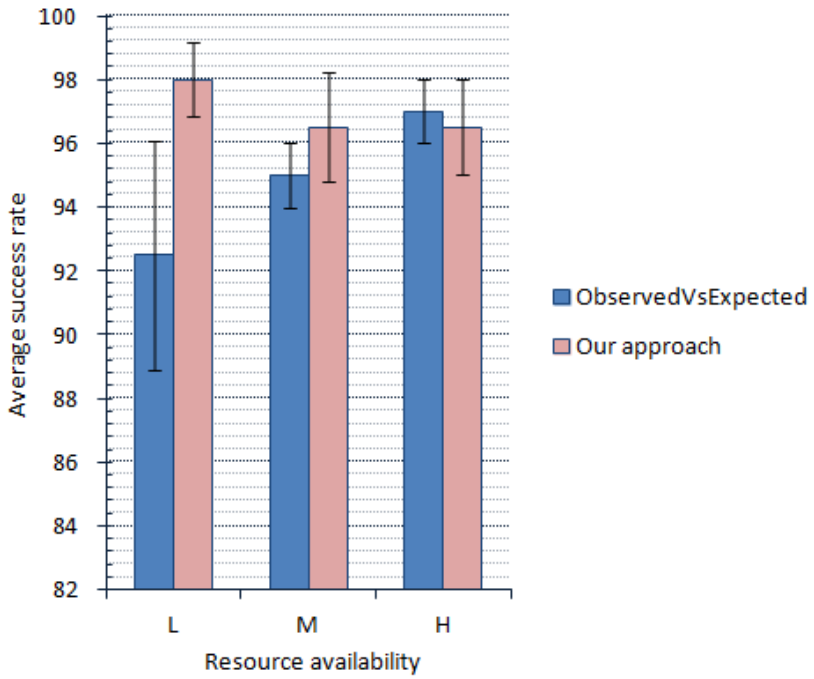

Figure 6.13: Graph showing the average success rates (and associated standard deviations) achieved by both approaches, given *low*, *medium* and *high* resource availability.

6.3.1 Analysis of results

The experiment was performed and the results are presented in Figures 6.13 - 6.15. When evaluating the results of the first experiment (see Section 6.2.1) we evaluated the success rates separately from achieved rewards. Prior to calculating the average achieved reward, any failed runs and their corresponding pairs (i.e. the run which shares the same set of sampled usages as the failed run) were again removed, as explained in Section 6.2.1.

6.3.2 Success rates

The average success rates achieved by both our approach and *ObservedVsExpected* for problems with *low*, *medium* and *high* resource availability are shown in Figure 6.13. When resource availability was *low*, our approach achieved a success rate of 98%, outperforming *ObservedVsExpected* by 5.5% (found to be statistically significant at the 5% level using a chi-square test (Caswell, 1989)). Our approach also achieved a significantly higher average success rate in the *medium* case (96.5%). In the case of *high* resource availability,

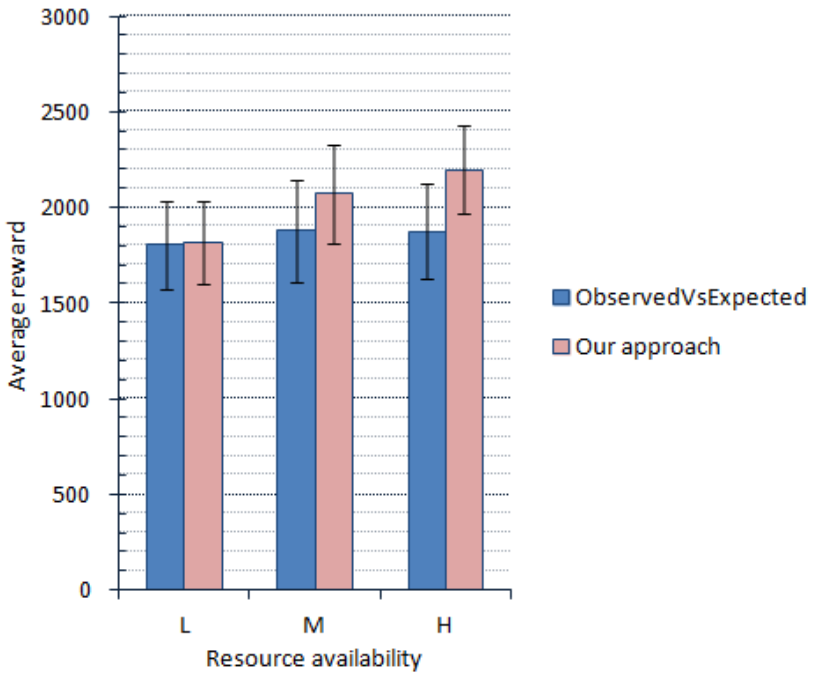

Figure 6.14: Graph showing the average rewards (and associated standard deviations) achieved by both approaches, given *low*, *medium* and *high* resource availability.

ObservedVsExpected appeared to slightly outperform our approach, achieving an average success rate of 97% compared to 96.5% for our approach, but this difference was not found to be statistically significant.

Over all problems, our approach achieved a statistically significant 2.2% increase in success rate over *ObservedVsExpected*.

6.3.3 Average reward

The average rewards achieved by both approaches for varying levels of resource availability are shown in Figure 6.14. 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 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 at the 5% level using the Wilcoxon signed rank

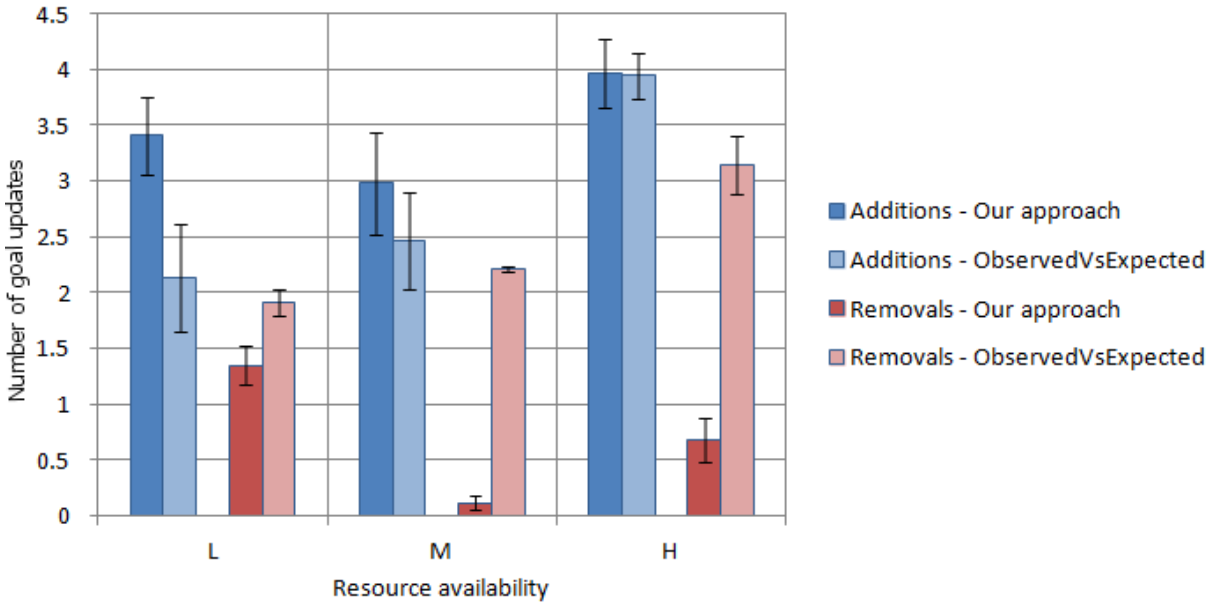

Figure 6.15: Graph showing the average number of goal additions and removals made per run by each approach, given *low*, *medium* and *high* resource availability. Error bars show the associated standard deviations.

test for paired samples (Wilcoxon, 1945).

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

6.3.4 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 approach outperformed *ObservedVsExpected* by 5.5%. This means that when compared to *ObservedVsExpected*, our approach was able to achieve a high success rate without compromising average reward.

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

In Figure 6.15, we show the average number of goal additions and removals made by

each approach given varying levels of resource availability. On average, our approach made more goal additions than *ObservedVsExpected* and less removals per run. This is especially noticeable in the case of *high* resource availability where, despite making a similar number of additions to our 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 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 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 (as described in Section 6.1) and thus experienced the same states at each branch point, this would suggest that the *ObservedVsExpected* branch conditions resulted in poor decisions, 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*.

6.4 Evaluating online plan modification against online replanning

We designed an experiment to compare the cost of the plan modification component of our system (i.e. adding or removing a single goal, without execution monitoring triggering plan modification) to that of total replanning[a]. For replanning, Metric-FF (Hoffmann,

[a]These results were first published in the proceedings of the 7th Starting AI Researcher Symposium (STAIRS) at ECAI 2014. This section appears in Harris and Dearden (2014) and is reproduced here.

2003) was used in the same configuration as when generating the initial plan, sub-plans and stitching plans for the plan modification algorithm, i.e. optimising for plan length. Owing to the size of the AUV domain and the inclusion of the renewable memory resource, optimising for minimal battery usage (which would produce better quality plans for this domain) was not feasible as Metric-FF does not return within a significant amount of time. We measured the time taken to construct the new PDDL problem file (and included the time taken to generate the sub-plan, in the case of our algorithm) and return either a valid solution or report a failure. We divided the experiment into two distinct tasks: firstly, to add an additional goal to the problem; and secondly, to remove an existing goal. In both cases the same initial problem (20 locations with 16 data-collection goals, shown in Figure 6.16) and the same initial plan (with 66 actions) were used. All updates to the goal set were made at the same point, fixed as the start of the initial plan.

As plan modifications are triggered by a change in observed resource usage, we varied the amount of battery and memory available at the point of replanning/plan-repair. When adding a goal, battery was increased from the mean usage of the existing plan, μ_b, to two standard deviations above the mean, $\mu_b + 2\sigma_b$. When removing a goal, battery was decreased from μ_b to $\mu_b - 2\sigma_b$. In both cases, memory ranges from the minimum required to complete the existing plan (501.0 units, the mean size of the largest dataset) to one standard deviation above this (501.0 $+\sigma_m$). Each data-point represents the mean of 15 trials. We specified a timeout of two minutes of CPU time for each trial. A laptop with a 2.40GHz Intel Core 2 Duo CPU and 3GB of RAM was used for all runs.

6.4.1 Adding a goal

Both approaches were tasked with producing a valid solution when the current problem was updated to include a new data-collection goal. The first additional goal consists of collecting and delivering (either via transmission or during vehicle recovery) a dataset ($D19$ from location $L19$, see Figure 6.16) which is already on the route taken by the existing plan, thus requiring minimal plan modification. The second additional goal, to

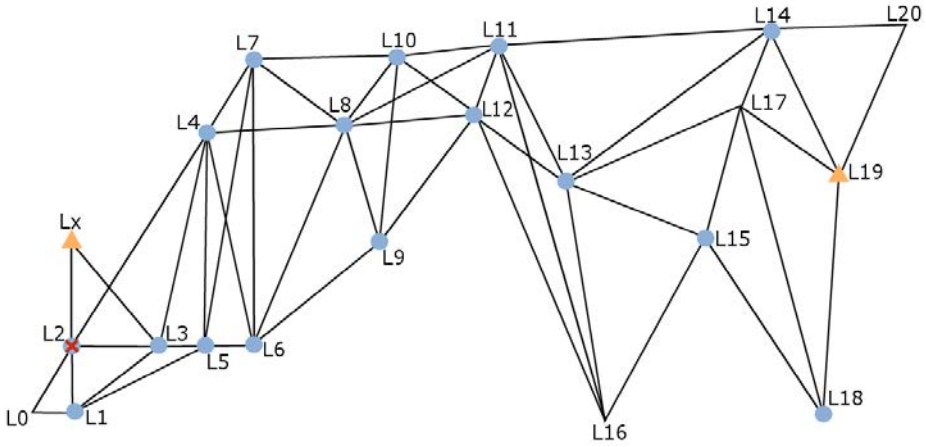

Figure 6.16: Connectivity of locations and datasets. Circles represent data-collection goals in the current problem. Triangles represent goals added during runs of the experiment. The crossed $L2$ indicates the removal of this goal during the experiment.

collect and deliver Dx from location Lx, is off the route of the existing plan, requiring much greater modification. By evaluating the performance using these two extremes, we attempted to ensure a fair representation of both approaches.

When adding the en-route data-collection goal, $D19$, the plan modification algorithm performed well, finding a solution in 84.6% of trials, compared to replanning which found a solution in 62.1%. Replanning from scratch was, on average, 48.8 seconds slower than adding this goal using our plan modification algorithm (a difference found to be statistically significant at the 5% level using the Wilcoxon signed rank test for paired samples (Wilcoxon, 1945)). The plan modification algorithm's performance was very consistent, with a standard deviation of only 0.01 seconds, compared to 56.45 when replanning. This significant difference is caused by replanning failing to return within the two-minute timeout period (as shown by the large plateau area, labelled a and b in Figure 6.17) and would be even greater had the timeout not been imposed. As the plan modification algorithm is not performing a full search, it is quick to report when a solution cannot be found (see areas b and c). However, as it does not consider the wider search space, plan modification may miss solutions which replanning was able to find (see area c). When available resources are plentiful (towards the bottom left corner of area d), replanning is very quick, out-performing the plan modification algorithm by up to 0.63 seconds. However, when

both approaches found a solution, replanning was, on average, 5.98 seconds slower (a difference found to be significant at the 10% significance level, using the Wilcoxon signed rank test for paired samples). We believe this difference in plan generation time is due to whether Metric-FF is able to replan using enforced-hill-climbing (e.g. when resources are plentiful and do not significantly restrict the search) or has to resort to slower best-first search (Hoffmann, 2003).

When adding the goal to deliver Dx, which required the vehicle to deviate from its previous route, the performance of the plan modification algorithm was predictably worse than when adding an en-route goal, as a *stitching* plan was required to join the plan fragment to the existing plan. As shown in Figure 6.18, replanning returned a solution in 98.2% of cases, whereas the plan modification algorithm found a valid plan in 61.5% of cases. Plan modification was, on average, 2.83 seconds faster than replanning (found to be statistically significant at the 5% level using the Wilcoxon signed rank test for paired samples); however, when only considering successful runs, replanning was an average of 0.95 seconds faster (also found to be statistically significant at the 5% level). The extra time required by the plan modification algorithm in this case, compared to when adding an en-route goal is due to the generation of the stitching plan, which requires the construction of a new PDDL problem file and additional actions using Metric-FF. The plan modification algorithm's performance was again highly consistent across all runs, with a standard deviation of only 0.02 seconds, compared to 34.38 for replanning.

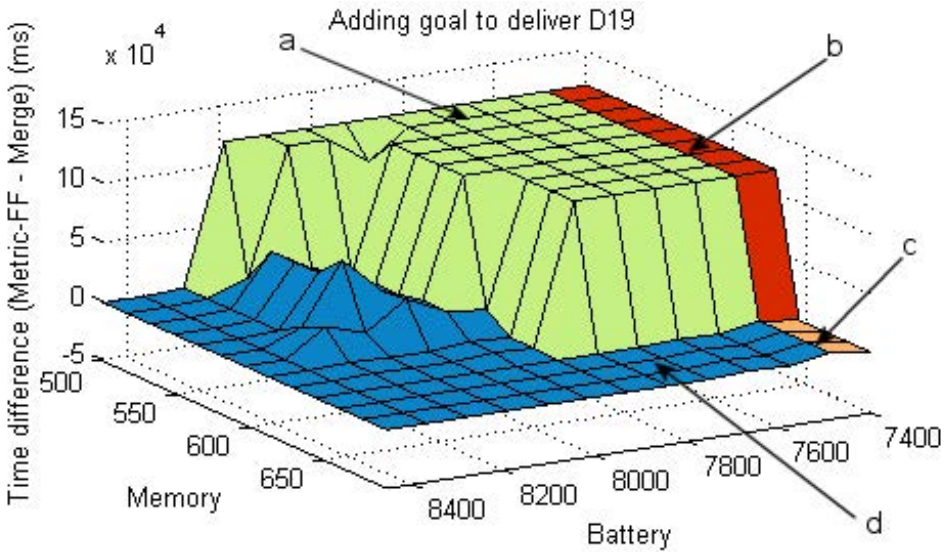

Figure 6.17: Time difference between the two approaches as a function of available resources when adding the en-route data-collection goal $D19$. The area labelled b represents cases where both Metric-FF and the modification algorithm failed to find a valid solution; c, cases where only plan modification failed; a, where only Metric-FF failed; and d, cases where both were successful.

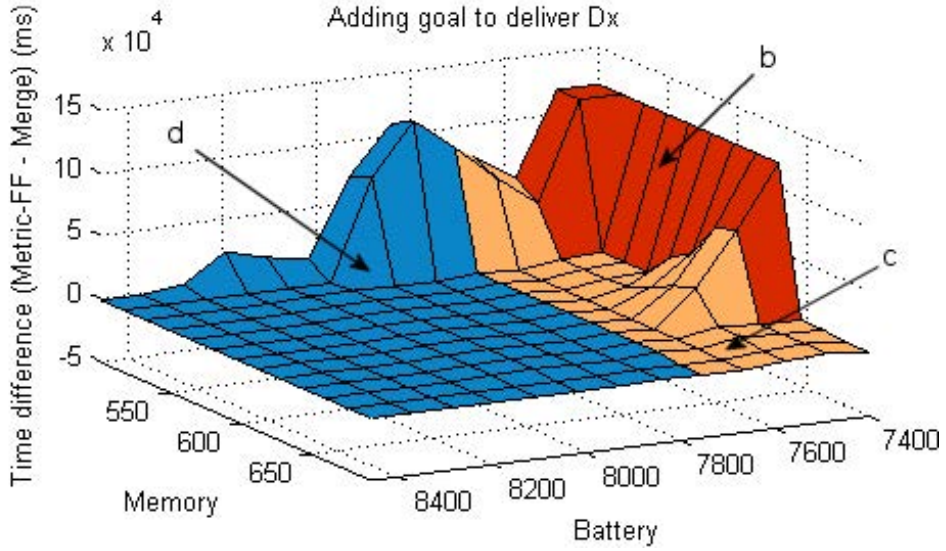

Figure 6.18: Time difference when adding the off-route data-collection goal Dx. The area labelled b represents cases where both Metric-FF and the modification algorithm failed to find a valid solution; c, cases where only plan modification failed; and d, cases where both were successful. There were no cases in which only Metric-FF failed.

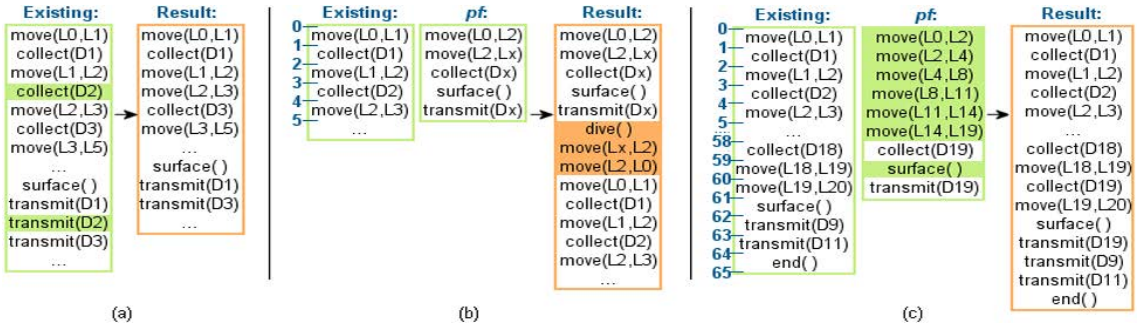

Figure 6.19: Plans used when: (a) removing the goal to deliver $D2$ (removed actions are shaded); (b) adding the goal to deliver Dx (actions from the stitching plan are shaded); (c) adding the goal to deliver $D19$ (skipped actions are shaded).

Discussion

The reason stitching is required when adding the off-route goal is illustrated in Figure 6.19 (b), which shows a subset of the existing plan, the new plan fragment pf and the plan resulting from the merge. The only state which meets the logical preconditions of the first action in pf is state 0. However, merging this action at state 0 threatens the causal links which require the vehicle to be at $L1$ to be able to collect $D1$ and move from $L1$ to $L2$. This threat is not resolved by the remaining actions in pf and so, as $move(L0, L2)$ does not achieve any goals, we skip $move(L0, L2)$ and consider $move(L2, Lx)$. The preconditions of $move(L2, Lx)$ are met by states 3 and 4, but merging the action at either state threatens the causal link requiring the vehicle to be at $L2$ to execute $move(L2, L3)$. The algorithm skips $move(L2, Lx)$ but is then unable to add $collect(Dx)$ as its preconditions are not met by any state. Actions in pf continue to be skipped until $transmit(Dx)$ is reached. This action may not be skipped as it achieves a goal. Instead, the algorithm generates a stitching plan by calling Metric-FF using the state at the end of pf as the initial state and the unsatisfied preconditions of the existing plan as the goal state. Actions added to the resulting plan from the stitching plan are shaded in Figure 6.19 (b). It may seem obvious to the reader that a more efficient plan would result if the vehicle was able to travel directly from $L2$ to $L1$, without having to travel via $L0$, as these are neighbours (see Figure 6.16). This is indeed a short-coming of the current stitching approach; however, correctly updating the variable bindings of existing actions to resolve such inefficiencies would be non-trivial as it is challenging to identify the optimal action to update without resorting to domain-specific criteria. When the full system operates as a whole, it is unlikely that the goal to return Dx would be added at this point during plan execution. The ratio of expected resource cost to expected reward is likely to be better later in the plan when the vehicle is already at $L2$, as in states 3 and 4. Consequently the decision to include this goal would be delayed until then.

When adding the en-route goal to deliver $D19$, a stitching plan is not required. Instead, the plan modification algorithm efficiently adapted the existing plan to meet the new

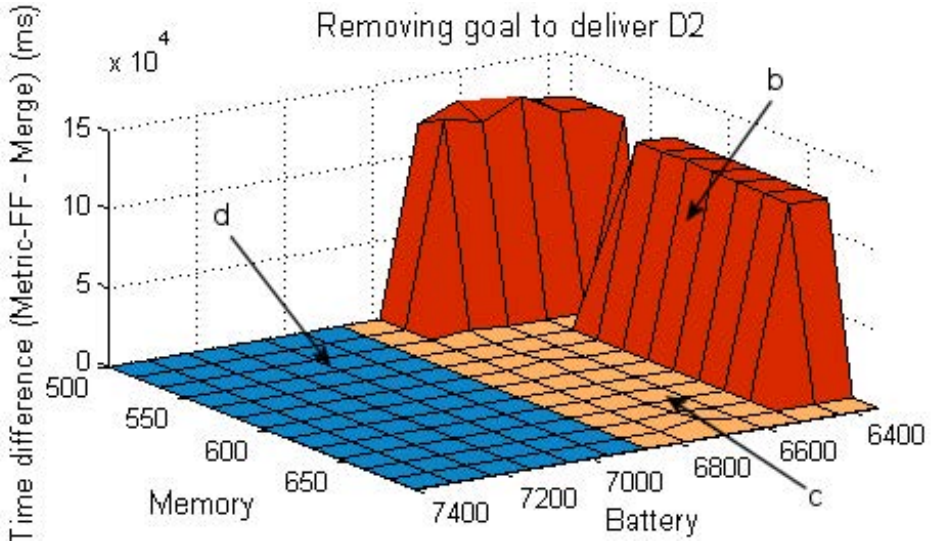

Figure 6.20: Time difference between the two approaches as a function of available resources when removing the goal to deliver $D2$. Area b represents cases where both Metric-FF and the modification algorithm failed to find a valid solution; c, cases where only plan modification failed; and d, cases where both were successful. There were no cases in which only Metric-FF failed.

goal by making minimal changes, as shown in Figure 6.19 (c). There are no states in the existing plan where the first six *move* actions may be included without introducing threats. Since these actions do not achieve any goals, they are skipped by the merging algorithm, Algorithm 2 in Section 5.7.3. The preconditions of $collect(D19)$ are met by state 60 in the existing plan and, as no threats are introduced, $collect(D19)$ is added at this point. The *surface* action causes threats and is skipped, but the goal-achieving $transmit(D19)$ action may be merged at states 62, 63 or 64, each resulting in a valid plan.

6.4.2 Removing a goal

When removing the goal to deliver $D2$ to the scientists, replanning found a valid plan in 85.2% of cases compared to our plan modification algorithm which found a plan in 46.2% of cases (as shown in Figure 6.20). However, replanning from scratch was, on average, 18.31 seconds slower than modifying the plan to remove the goal (owing to replanning reaching the timeout, represented by the peaks in area b of Figure 6.20) and 0.13 seconds slower

when comparing only successful runs. Both differences were found to be statistically significant at the 5% significance level using the Wilcoxon signed rank test for paired samples. The plan modification algorithm was faster than replanning in all cases, when either finding a solution or reporting a failure. Again, the standard deviation of the plan modification algorithm (0.003 seconds) was considerably lower than for replanning (43.59 seconds).

Discussion

The reason why replanning finds more solutions than the plan modification algorithm when removing the goal to deliver $D2$ is illustrated in Figure 6.19 (a), which shows the subset of the existing plan concerning the completion of the goal and the resulting plan following goal removal. When removing the goal, the algorithm first removes the $transmit(D2)$ action as it was only present to complete this goal and is no longer required. The $collect(D2)$ action is also removed in this way. Note the redundancy in the resulting plan between the $move(L1, L2)$ and $move(L2, L3)$ actions, as it is possible for the vehicle to move directly from $L1$ to $L3$. As the vehicle is no longer required to collect $D2$, we theoretically have no reason to visit $L2$. However, the subsequent $move(L2, L3)$ action has a precondition that the vehicle is at $L2$ and so the $move(L1, L2)$ action is still required and may not be removed. Replanning avoids this redundancy as it is not constrained by the causal structure of the initial plan. When battery is low and memory is high, replanning may produce a plan which collects more datasets before needing to surface and transmit. This allows it to produce valid plans requiring less battery than the initial plan, causing it to succeed in a greater number of cases than plan modification. However, when operating under realistic conditions, it is unlikely that this situation would occur to the same degree. This is because the current available memory would never exceed the amount available when generating the initial plan, so replanning would be unable to capitalise by trading excess memory for savings in battery power.

6.5 Evaluating sequential plan modification and goal selection against an over-subscription planner

In this section we compare the performance of our goal selection and plan merging algorithm to a state-of-the-art over-subscription planner, OPTIC (Benton et al., 2012). Whilst the results from the previous experiments in this chapter showed that our online plan modification and execution monitoring approach offers improvements over both straight-line plans and online planning within our AUV domain, it was important to also evaluate the goal selection choices made by our approach and the quality of the resulting plans. 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.

As shown in experiment three, presented in Section 6.4, online replanning is not feasible for use in domains such as our AUV scenario. For this reason, we did not consider the use of OPTIC online, for example as a direct replacement for our own goal selection procedure and goal merging algorithm. 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 amount specified in the PDDL domain encoding. For this experiment, this was defined for each action as the mean plus one standard deviation of their resource usage distributions. By making such assumptions we ensured that plans generated by OPTIC executed in exactly the way the planner anticipated and were not disadvantaged at run-time.

When designing the experiment, another important factor to consider was that our system sequentially modifies an existing plan during execution, arriving at an eventual goal set in response to observations made during execution, rather than generating both the eventual goal set and plan in a single search prior to execution, as OPTIC does. As

discussed in Section 5.3.1, 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 — see Section 5.3.1), 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 branch point prior to the execution of the first action. We recorded the mean battery usage and mean reward of the resulting plan. It was not necessary to record the memory usage of each plan as memory is renewable and thus is not as representative of the quality of a solution as the consumption of non-renewable battery. However, all plans were checked to ensure they met all resource constraints. We also recorded the time taken by both approaches to find a solution. As OPTIC is an anytime planner (i.e. it will return each solution it finds, continuing to search for increasingly better plans until the search is complete or a timeout is reached), 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 (i.e. if we ran the planner for one hour, but it returned a single plan after three seconds, we recorded the plan and a computation time of three

seconds, and not one hour). 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)*.

In summary:

- We assume fixed known resource usage for both approaches to nullify the advantage our online approach would otherwise have (i.e. by being able to react to and capitalise on unexpected situations). Our system is invoked once, on a single branch point prior to the execution of the first action.
- Goals in the initial goal set are defined as 'hard' goals which must be completed and may not be removed.

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 approach against existing methods was essential. Whilst not optimal, as a deliberative over-subscription planner, OPTIC provides the closest available approximation. A knowledge of what 'good' plans and goal sets look like within domains such as ours is very valuable when evaluating the performance of our approach.

6.5.1 Analysis of results

We initially performed the experiment using the set of test problems defined in Section 6.1. While Metric-FF (Hoffmann, 2003) was able to find initial plans for these problems in our previous experiments given varying levels of resource availability (i.e. *high*, *medium* and *low* as defined in Section 6.1), OPTIC was not able to return a plan for any of these problems using the previously defined resource levels. Consequently, we considered a

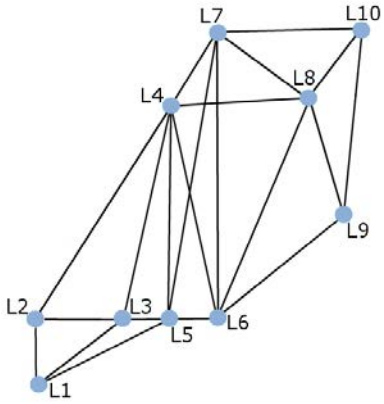

Figure 6.21: Topology of locations and datasets as defined in the new test set of smaller problems. Circles represent data-collection opportunities.

larger range of resource availability, increasing the availability to 130%, 140% and 150% of the *low* case. Given this expanded set, OPTIC was only able to find a plan in a single case, that of problem three with 150% resource availability. In all other cases, system memory or the timeout were exceeded. As a single data-point is insufficient for drawing any conclusions, we instead defined a set of smaller problems for use in this experiment. The problems used the topology of survey locations shown in Figure 6.21. This equates to half the number of locations used in the other experiments in this chapter. 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. As we assumed fixed known resource usage, both approaches are now deterministic and thus it was unnecessary to repeat each run of the experiment.

6.5.2 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%).

We compared the mean (expected) reward achieved by the plans generated by both approaches. The results are presented in Figures 6.23 and 6.24. In Figure 6.23, we can see

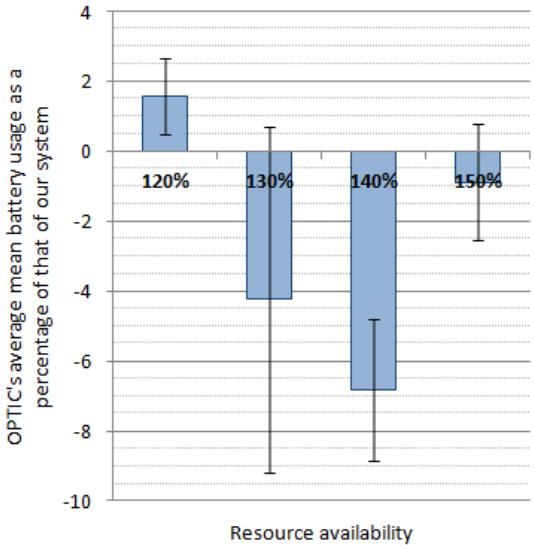

Figure 6.22: Graph showing the average mean battery usage (and associated standard deviations) of plans produced by OPTIC as a percentage of the mean battery usage of plans produced by our approach, given a range of resource availability.

that while OPTIC outperformed our approach in many cases, the difference was relatively small, under 10% in all but one case. In two cases, OPTIC performed over 5% worse than our system and in four cases, we recorded no difference in reward. Figure 6.24 shows the average of all successful runs for each level of surplus resources (and associated standard deviations). Across all problems, OPTIC improved on the reward achieved by our system by 3.26% on average. Using the Wilcoxon signed-rank test for paired samples (Wilcoxon, 1945), this difference in the mean rewards was found to be statistically significant at the 5% level.

6.5.3 Expected resource usage

When attempting to judge the quality of a plan, we also must consider the expected resource usage of the plan. As discussed earlier in this section, the memory usage of a plan is not very informative as it is renewable. Instead, as renewing memory (e.g. by transmitting datasets) requires additional battery power, we solely consider resource usage in terms of battery.

Figure 6.22 shows the percentage by which OPTIC's plan required more battery than

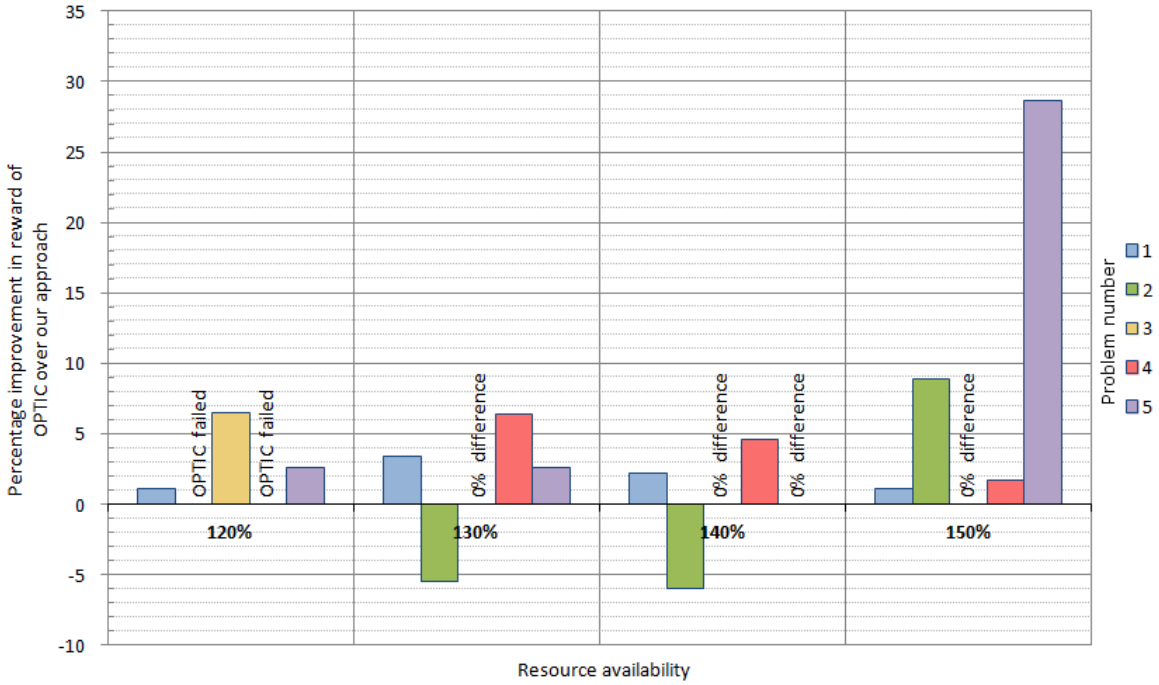

Figure 6.23: Graph showing the percentage improvement in reward of plans produced by OPTIC over those produced by our approach $((reward_{OPTIC} - reward_{our})/reward_{our} * 100)$, for each problem in the new smaller test set.

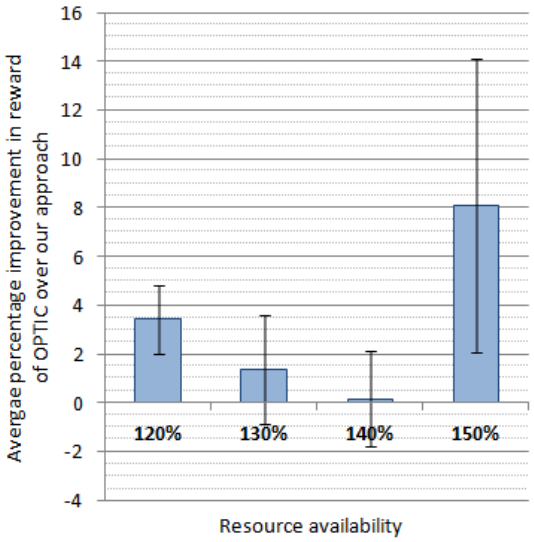

Figure 6.24: Graph showing the average percentage improvement in reward (and associated standard deviations) of plans produced by OPTIC over those produced by our approach, given a range of resource availability.

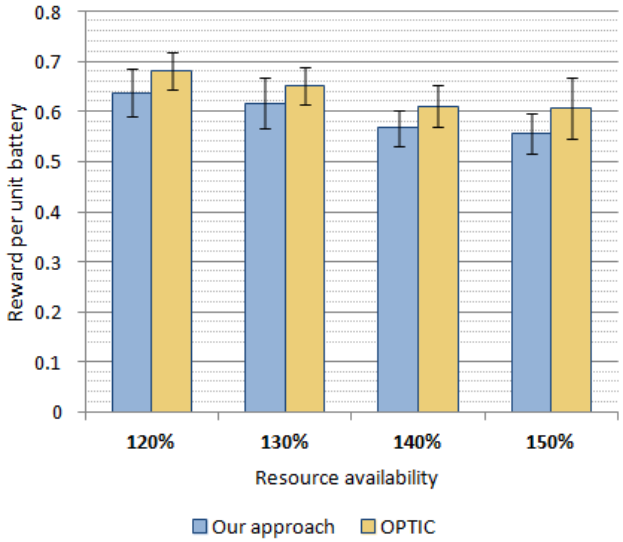

Figure 6.25: Graph showing the average reward per unit battery (and associated standard deviations) of plans produced using both our approach and OPTIC, for varying levels of resource availability.

that produced by our approach (thus negative values represent cases where OPTIC's plan required less battery than that of our approach). Whilst the standard deviations are large across all cases, plans produced by OPTIC used, on average, 2.59% less battery than those produced by our approach. Using the Wilcoxon signed-rank test for paired samples, this difference in the mean battery usage was found to 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.

6.5.4 Plan quality

The results from the previous sections showed that, in comparison to OPTIC, our approach achieved lower rewards and used more battery to do so. This implies that the quality of our plans may be slightly lower than those produced by OPTIC. However, as the differences are small, the results suggest that the quality of our plans is reasonably similar to that of OPTIC. Indeed, 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

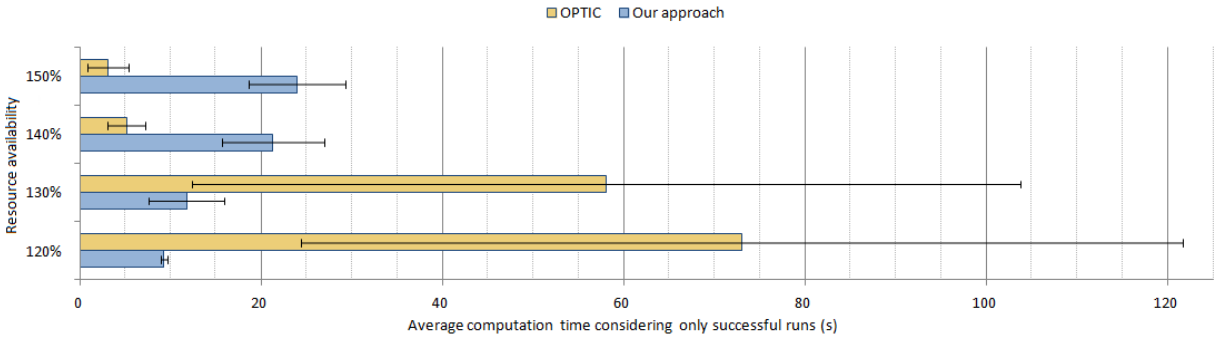

Figure 6.26: Graph showing the average computation time in seconds (and associated standard deviations) required by both our approach and OPTIC, considering only successful runs (i.e. ignoring timeouts).

a trend across all problems, shown in Figure 6.25. 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 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 (statistically significant at the 5% level, using the Wilcoxon signed-rank test for paired samples), 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 over-subscription planner.

6.5.5 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 6.26. 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.

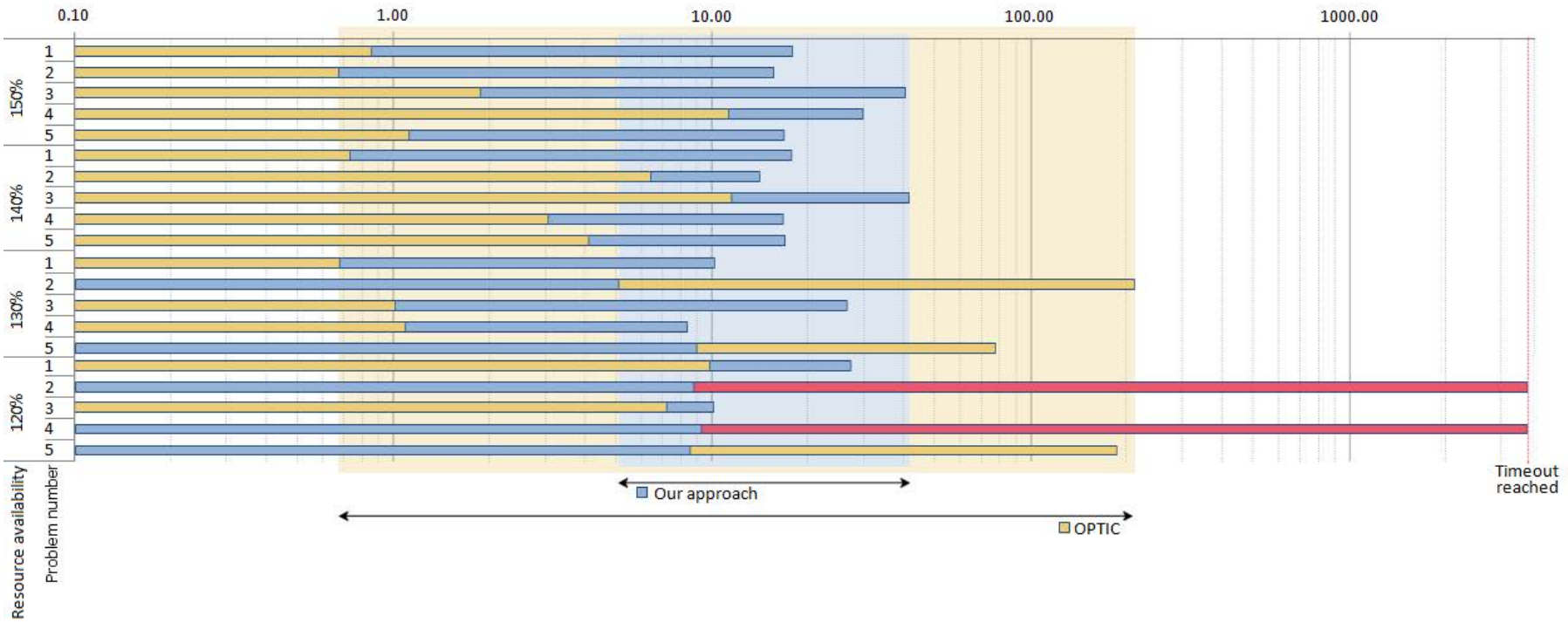

Figure 6.27: Graph showing the computation time required for both OPTIC (in yellow) and our approach (in blue) across all problems. The two OPTIC runs which reached the timeout (3600 seconds) without returning a plan are shown in red. Shaded areas highlight the range of times required by both approaches in successful runs (note the logarithmic axis) and correspond to the arrows below the graph.

This reflects the trend seen in the previous experiment (see Section 6.4) when replanning with Metric-FF (Hoffmann, 2003). When resource availability is high, both planners are very fast. However, this is not the case when resources are more constrained. We believe this disparity is due to whether or not the heuristics used by both planners to improve the efficiency of the search are suited to the problem in question. For example, the heuristic used by OPTIC assumes costs are monotonically worsening (Benton et al., 2012). When resource availability is very high, it is unlikely that actions to replenish memory will need to be considered. However, when memory is much lower and thus replenishment actions are necessary, a heuristic which assumes action costs are monotonic is likely to be much less effective. Figure 6.27 illustrates the variability of both OPTIC and our approach across all problems — note the logarithmic horizontal axis. Excluding the two cases where the timeout was reached (shown in red), the computation time required by OPTIC ranges from 0.6 to over 200 seconds, compared to a range of between 5 and 40 seconds for our system. 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 because goals are added as soon as resources and success probability constraints will allow, without ever allowing large amounts of surplus resources to accumulate. 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.

6.6 Conclusions

In this chapter, we presented an evaluation of the performance of our approach on AUV domain problems, analysing achieved reward and success rate.

We first evaluated the effect of varying the quantity of branch points on success rate

and reward, finding that:

- Defining branch points after 20-100% of actions significantly increased the success rate over that of a straight-line plan when resources were constrained. The definition of branch points allowed updates to the plan which sacrificed reward to safeguard mission success.

- When resources were plentiful, branch points allowed opportunities for additional reward to be exploited without adversely affecting success rate, increasing achieved reward by an average of 7.9% above that of the straight-line plan.

- We did not find a clear trend between improved performance and higher quantities of branch points. While both online and offline computation time were found to increase with the number of branch points, we found that increasing the number of branch points above 20% did not significantly affect either success rate or achieved reward.

We then evaluated our branch criteria, which specifies when the algorithm should consider adding or removing a goal, against an alternative approach, *ObservedVsExpected*. Unlike our branch criteria, that consider current resource availability and the probability of successfully completing the mission, *ObservedVsExpected* chooses whether to consider adding or removing goals by comparing the expected resource usage of the plan to the actual usage observed during execution. Our results showed that:

- As resource availability increased, our approach was able to capitalise on additional opportunities, increasing reward without compromising success rate.

- While the success rate achieved by *ObservedVsExpected* increased with resource availability, this alternative approach was unable to effectively exploit opportunities for additional reward.

- We compared the average number of goal additions and removals made per run by each approach. While our approach made significantly more additions than

removals, *ObservedVsExpected* appeared overly-reactive, removing nearly as many goals as it added.

We compared the computational cost of our online plan modification approach to that of complete replanning with Metric-FF (Hoffmann, 2003), whilst varying resource availability. Our results showed that:

- 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 approach.
- In the average case, our approach required less computation time than replanning from scratch, when either adding or removing goals.
- Our plan modification algorithm 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.
- When removing goals, our system found a plan in 46.2% of cases, compared to 85.2% for replanning.

Finally, we compared the goal selection and plan merging capabilities of our system to plans produced by an over-subscription planner, OPTIC (Benton et al., 2012). As OPTIC was unable to find plans for problems in our existing test set, it was necessary to define a new test set of smaller problems and increase the resource availability. Our results showed that:

- Our system found a plan in 100% of cases, compared to 90% for OPTIC.

- Across all problems, plans produced by OPTIC improved on the rewards achieved by our system by an average of 3.3%, requiring an average of 2.6% less battery. This was to be expected as OPTIC employs a comprehensive search, compared to our greedy approach to goal selection.

- When we analysed the reward per unit battery, our approach achieved 93.2% of that achieved by OPTIC. This shows that the quality of plans and goal sets selected by our approach closely approximates those of a state-of-the-art over-subscription planner.

The reader may question why we considered only a relatively small number of problems and whether our test set provides sufficient 'coverage' of possible problems. The space of problems in the AUV domain is very large due to the possible topologies of survey locations, the resource costs of each action and the variability on both the initial and eventual goal sets. To allow us to analyse our approach and draw clear conclusions, we needed to limit the number of variables. To achieve this, we fixed the location topology for all problems and the expected usage (μ, σ) of each action before hand-picking a set of varied problems. While the expected usages are fixed, there is significant variation in both the means and the standard deviations. The standard deviations range from $< 5\%$ to approximately 25% of the mean. Given the large number of possible eventual goal sets and plans, we feel this is sufficient parameterisation to provide a representative and varied test set. It should also be noted that as we repeated each experiment 50 times for each problem/resource combination, the results were time consuming to produce, which restricted the size of the tests we could perform. In cases where the external planner (e.g. Metric-FF) could not quickly find solutions and thus many time-outs were reached, the time to perform a single set of 50 repeats (equal to one data-point) required up to 24 hours of computation time.

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. Due to

this, in the next chapter we analyse and discuss the merge algorithm when applied to existing International Planning Competition (IPC) domains.

CHAPTER 7

APPLICABILITY TO EXISTING DOMAINS

The approach described in this thesis was developed to solve a class of over-subscribed planning problems which feature continuous resource uncertainty. Whilst our work was motivated by and evaluated using our AUV domain, our system is applicable to a much wider range of problems. In this chapter, we discuss domain characteristics which are applicable to our online plan modification and execution monitoring approach as well as those for which an alternative solution should be considered. This chapter is designed to inform the reader whether the approach presented in this thesis could be considered as a potential solution to their own planning problem.

Whilst we have not performed a full evaluation of our approach on domains other than our AUV planning problem, in this chapter we present examples of outputs resulting from the application of our merge algorithm (presented in Section 5.7.3) to a number of widely published domains. The merge algorithm, which is capable of combining multiple plans into a single plan for execution, is a key component of our system and may be considered as a contribution of this thesis in its own right as we have yet to find existing work in the literature which combines plans in the same way. As the performance of the merge algorithm is a limiting factor in the success of our overall system, we present and discuss examples of its application to other domains.

7.1 Key domain characteristics

To enable the reader to decide whether to consider our approach when attempting to solve their own planning problem, in this section we outline the key characteristics a planning domain should have before our approximate approach may be considered as a potential alternative to existing approaches. The applicability of both our system and alternative approaches in relation to combinations of these key domain characteristics are summarised in Table 7.1 in Section 7.1.4.

7.1.1 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 situations, this choice is informed by a measure of solution quality, such as minimising cost/time/plan length or maximising reward. PDDL 2.1 (Fox and Long, 2003) allows the definition of such an optimisation criteria, or *metric*, via the use of the `(:metric` syntax. 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. For example, Metric-FF (Hoffmann, 2003) allows the user to define a lower-bound on a metric (e.g. total plan cost), only returning solutions which exceed this threshold.

The system presented in this thesis does not strictly require the use of a metric during plan generation (e.g. if the user's chosen planner does not allow it), but care should be taken to ensure that the plans produced are of a suitably high quality. As discussed in Section 5.3.1, in plan-repair based approaches, the quality of the eventual plan is directly influenced by the quality of the initial plan. However, to enable our system to make informed decisions when evaluating plans and goal sets, it is crucial that a suitable metric is defined for use at run-time. The 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 defined in Section 5.4.3. However, alternative

metrics may be defined for use on other domains depending on the characteristics of an optimal solution.

7.1.2 Over-subscribed problems

The approach presented in this thesis was designed for use with over-subscribed domains, i.e. those where the resources available to the vehicle are insufficient to achieve all goals and thus a choice of goals to complete must be made. If the domain is not over-subscribed, i.e. all goals within a fixed goal set must be satisfied, no changes may be made to the goal set during execution and thus our approach offers no benefits over executing a fixed straight-line plan.

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 system is likely to be when applied to a new domain. In our AUV domain, the causal structure is such that the goal literals are relatively independent — while resource availability may prevent the completion of a goal, there are 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 success probability 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[a], which are not over-subscribed, the goal literals each represent components of a single goal state comprising an arrangement of blocks and

[a]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.

thus are highly dependent on each other. While it is possible for our merge algorithm to successfully add goals to Blocksworld problems under very specific circumstances, such as adding a new block to the top of a stack (as we shall see in Section 7.3.3), 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.

7.1.3 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. A fixed straight-line plan will execute exactly as the planner assumed during plan generation and so there are no alternative outcomes to consider. Valid and optimal solutions may be found to such problems using many classical planners. In cases where goal selection is necessary, over-subscription planners may also be used.

The approach presented in this thesis is designed for domains where the outcomes of actions are continuous and uncertain, which we represent as probability distributions. In our AUV domain, we model the uncertain usage of two continuous resources, battery and memory, as Gaussian probability distributions (as described in Section 2.3). However, our system also supports the use of alternative continuous probability distributions, provided a suitable discretisation may be found for use during plan generation (e.g. in our AUV domain, during plan generation we assume action resource usage is equal to the mean of each distribution). Consequently, the uncertainty in a domain does not need to be modelled as a Gaussian for our approach to be applicable.

As discussed throughout this thesis, 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 (as shown by results presented in Sections 4.2 and 6.2.2).

- MDP-based methods require coarse discretisation of the transition function and are limited to those problems with a sufficiently small state space (as discussed in Section 3.1).

- Contingency planners require either a finite number of possible contingencies or suitable outcome discretisation (see Section 3.3).

- Replanning, such as that performed by FF-Replan (Yoon et al., 2007), 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 (see Section 3.2).

While these alternative approaches have limitations when applied to domains featuring uncertainty over continuous state variables (thus providing the motivation for this thesis), 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 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 the specified metric, it does not perform a full-search for an optimal plan/policy. Consequently, whilst our approach supports uncertainty over both discrete and continuous state variables, if a domain does not feature continuous variables, we would instead recommend the use of alternative approaches, such as an

MDP solver or contingency planner as we would expect these to produce solutions which are either optimal or are close approximations.

7.1.4 Summary table

Approach	Action costs	Over-subscribed goal set	Discrete probabilities	Continuous probabilities	Metric	Large state space
Online plan modification (our approach)	✓	✓	✓	✓	✓	✓
MDP-based methods	-	✓	✓	✓[a]	✓[b]	×
Contingency plan	-	✓	✓	×	-	-
Over-subscription planner	✓	✓	×	×	✓	-
Re-planning	-	×	✓	×	-	-
Straight-line plan	-	×	×	×	-	-

Table 7.1: Table summarising the applicability of both our approach and other related planning approaches given the presence of different combinations of key domain characteristics. Dashes (-) indicate cases where the presence (or lack of) a certain characteristic does not have a significant impact on the success of the approach.

[a] Requires a suitable transition function.
[b] Represented as the reward function.

7.2 Future work

In this section, we discuss the updates which would be required to extend our current system to support additional PDDL functionality.

7.2.1 Durative and concurrent actions — PDDL2.1

PDDL2.1 (Fox and Long, 2003) introduced a standardised representation for temporal planning domains — those which explicitly model the passage of time — in the form of durative actions. The standard STRIPS-based PDDL action representation assumes that action execution is instantaneous. Prior to PDDL2.1, the time required by an action could be represented as an action cost, as used to model battery and memory in our AUV domain encoding (see AUV PDDL in Appendix D). Gough et al. (2004, p. 24) represent time in this way, treating it "in exactly the same manner as other physical resources". Consequently, this representation of time is fully supported by our current system. However, this representation is restrictive as it does not allow a planner to fully reason about and exploit opportunities for concurrent action execution.

Concurrent actions

As durative actions permit concurrent action execution, in order to support a richer temporal model within our system we would first need to extend our assumptions about action ordering. Currently, we model the execution of a plan as a sequence of alternating states and actions, assuming one action per step in the plan. However, this is insufficient for representing more complicated action orderings and their associated causal structure, the accurate representation of which is crucial to our plan modification approach. To address this issue would require the implementation of a temporal plan representation, such as a timeline (Ghallab and Laruelle, 1994). Causal links would then apply over intervals in the timeline structure, rather than between points in the state-action sequence, as at present.

Whilst the introduction of a temporal plan representation such as a timeline would require significant updates to our present system, such an extension would be a feasible and logical future direction for our work. We do not foresee any features of a temporal representation which would invalidate the logic of our overall approach. However, careful consideration should be given to the definition of branch points within the new representation as well as the meta-planning question of whether online plan modification should itself be represented as a durative action.

Defining duration

In PDDL2.1, durative actions replace the standard `(:precondition` clause in the action definition with a more expressive `(:condition` syntax. Unlike preconditions, which specify facts which must hold before an action may be executed, Fox and Long (2003) state that each condition must be 'temporally annotated' as applying either at the start (`at start`), at the end (`at end`) or during the execution of an action (`over all`). The effects of durative actions must also be defined using these three annotations. For example, an action effect may switch on a light at the start of its execution, switching it off at the end. In addition, durative actions have a new clause, `(:duration`, which defines the time required to execute the action. This may be fixed, e.g. the flight-time between two locations:

```
(:duration (=?duration (flight-time ?x ?y)))
```

or defined as an inequality, which gives the planner control over how much time to invest in performing an action, e.g. recharging a battery:

```
(:duration (<?duration 10) (>?duration 0))
```

Provided our system was updated with a suitable temporal representation, such as a timeline (Ghallab and Laruelle, 1994) and used in conjunction with an external temporal planner, we would expect fixed action durations to be supported by our approach.

However, dynamic planner-controlled durative actions are unlikely to work well within our current system. In our system, the eventual plan (as executed by the vehicle/agent) is the result of multiple updates to an initial plan and thus is not generated as a whole. Consequently, as the initial plan, sub-plans and stitching plans are generated individually by an external planner, action durations would be defined without the context of either the overall plan or the current state observed during execution. This is very unlikely to result in the execution of a high-quality coherent plan. For example, the optimal duration of an action may depend on factors such as the resource availability in the state at which it is added into the plan — if time is limited, a recharge action may be used for a minimal amount of time; if time is plentiful but energy is low, it may be best to use the action until energy is fully replenished. To produce high quality solutions, the durations of actions would need to be re-assigned following each modification during plan evaluation in order to take the whole plan into account. Such an extension would require significant updates to both our approach and assumptions, and would still be unlikely to compete with approaches that compute the full plan as a whole.

Discretised and continuous durative actions

PDDL2.1 allows the representation of two forms of durative actions: *discretised durative actions* and *continuous durative actions*.

Whilst in reality an action may affect a numeric state variable over time, discretised durative actions model this change as a discrete update which occurs at either the start or end of an action. Fox and Long (2003) define a concept of 'conservative resource updating' in which the consumption of a resource must be modelled as occurring at the start of an action, whilst the production or replenishment of a resource must be modelled as occurring at the end. This results in a step function model of the underlying 'actual' continuous resource usage, as illustrated by Fox and Long (2003, p. 77, Fig. 9) and reproduced here in Figure 7.1 (a). By making such ordering assumptions when modelling both resource consumption and replenishment, Fox and Long ensure that concurrently

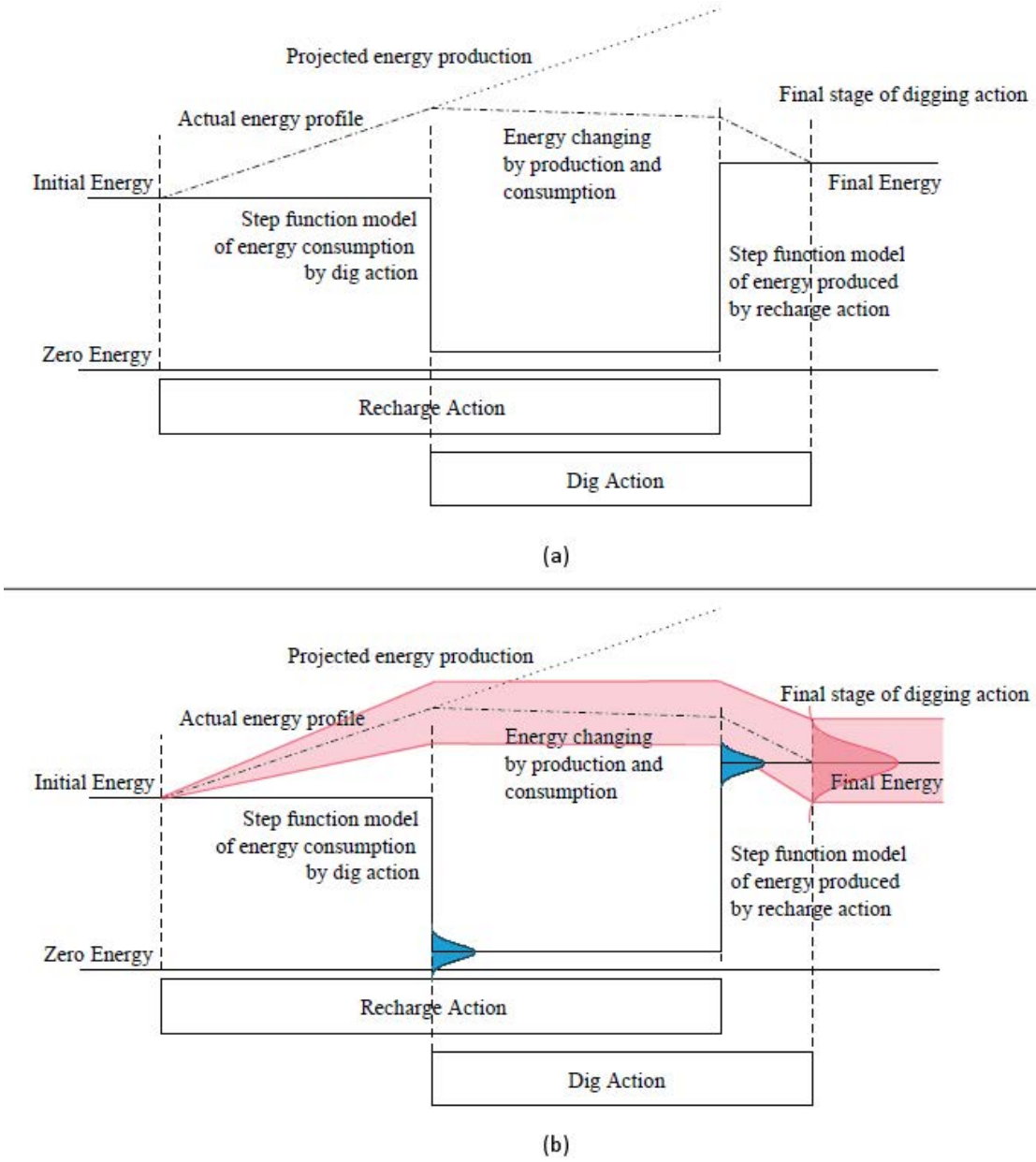

Figure 7.1: (a) Figure reproduced from Fox and Long (2003, p. 77, Fig. 9) illustrating their conservative resource update assumption. Note how the step function model is updated at the start of the resource *consuming* 'Dig' action and again at the end of the resource *replenishing* 'Recharge' action. (b) Updated version of (a) showing how the conservative resource update assumption may be combined with our model of resource usage uncertainty. Blue Gaussian distributions represent the uncertainty associated with individual actions; the pink band and Gaussian distribution represent the uncertainty over the actual energy profile.

executing actions do not overestimate the true availability of a resource. For example, if conservative resource updating was ignored, and the 'Recharge' action in Figure 7.1 (a) instead applied the step-function increase in energy availability at the start of its execution rather than at the end, the 'Dig' action would overestimate the 'actual' energy availability and begin concurrent execution, potentially exhausting the physical energy source, such as a battery. Conservative resource updating prevents this situation from happening whilst ensuring that 'the final energy level is consistent with having used a continuous model' (Fox and Long, 2003, p. 77).

Provided our system was first updated to use a suitable temporal representation, the use of discretised durative actions would be a feasible and valuable extension to our system's capabilities. As shown in Figure 7.1 (b), Fox and Long's conservative resource updating assumption may be combined with our model of resource usage uncertainty to produce a temporal model which adheres to our current assumptions. When discretising resource usage during plan generation, we assume the usage effect of each action is equal to the mean of its distribution. This creates a step function approximation of the actual resource usage profile of the plan. As resource usage is uncertain, the future resource usage profile is therefore represented by the sum of all action distributions.

Continuous durative actions allow both discrete and continuous effects. Unlike discretised durative actions which update state variables at either the start or end of their execution, continuous durative actions allow the correct value of a variable (e.g. the current availability of a resource) to be calculated at any point during action execution. Consequently, continuous effects are not temporally annotated but are instead essentially equivalent to differential equations, where the use of a resource depends on a local clock, $\#t$. Continuous durative actions provide a much richer representation of resource usage over time compared to discretised durative actions, allowing concurrent actions to calculate the true value of a resource, rather than relying on the conservative resource updating assumption.

In order to extend our approach to support continuous durative actions, a much richer

model of resource consumption and availability would first be required. While the actual consumption of a vehicle's resources is continuous over time, we currently model this change as discrete updates and observations made after the execution of each action. Consequently, implementing support for continuous durative actions would require significantly larger updates to both our underlying resource model and assumptions, when compared to supporting discretised durative actions.

7.2.2 Derived predicates — PDDL2.2

In PDDL2.2 (Edelkamp and Hoffmann, 2004), predicates may be defined according to rules, known as derived predicates, instead of being created by action effects specified in the initial state. For example, a rule may specify that when at a location a vehicle is not at all other locations, and will generate the predicates required to express this fact. While our system does not currently support derived predicates, we expect that relatively minor adjustments would be needed in order to include this additional functionality. As the truth of derived predicates is specified by a rule, we would need to extend our causal link extraction algorithm to also consider the applicability of each rule in each state, creating causal links to and from derived predicates and states/actions.

7.3 Application of our merge algorithm to existing domains

Given an over-subscribed goal set, our merge algorithm (presented in Section 5.7.3) 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*). We also present an example of another well-known domain, Blocksworld, which is less suited to our approach due to its highly interconnected goal literals. All three

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 (Hoffmann, 2003) 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 5.6.3). Due to the similarities between our AUV domain and that of the *Rovers* domain (as used by Bresina et al. (2002)), we considered evaluating the performance of our merge algorithm on the *Rovers* domain. The two domains share many characteristics including independent goals, a single agent and few constraints on the order in which goals are completed. However, we decided that the two domains are too similar to facilitate discussion of the merge algorithm's capabilities. We wanted to instead trial the merge algorithm on sufficiently different domains to investigate its performance on domains outside the subset it was designed to solve. By doing this, we aim to discover more about the merge algorithm's strengths and weaknesses, identifying domain characteristics which suit our approach.

7.3.1 Hiking

Hiking is a planning domain authored by Lee McCluskey which was first introduced at the deterministic track of the Eighth International Planning Competition in 2014 (see *IPC 2014*). The 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 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. Goals represent the final destination of the hike for a particular couple, and thus multiple couples may walk to additional goal destinations.

- 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). While for this example we have kept the original domain without any such extensions, it made logical sense to select a domain which might easily be extended into one for which our online plan modification system would be a viable solution.

- it is suitably different from our AUV domain. If we ignore the resource usage in our AUV domain, the vehicle's actions are relatively unconstrained. The collection of a dataset must happen before it can be transmitted, but the order in which they are collected does not matter. The edges which connect all locations are also bi-directional and dead-end states (provided we continue to ignore resource usage) may only be reached by executing the *EndMission* action. In contrast, 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. We were interested in investigating whether our plan merging algorithm would be able to successfully add goals for problems in the *Hiking* domain, given the additional constraints and complexity in causal structure.

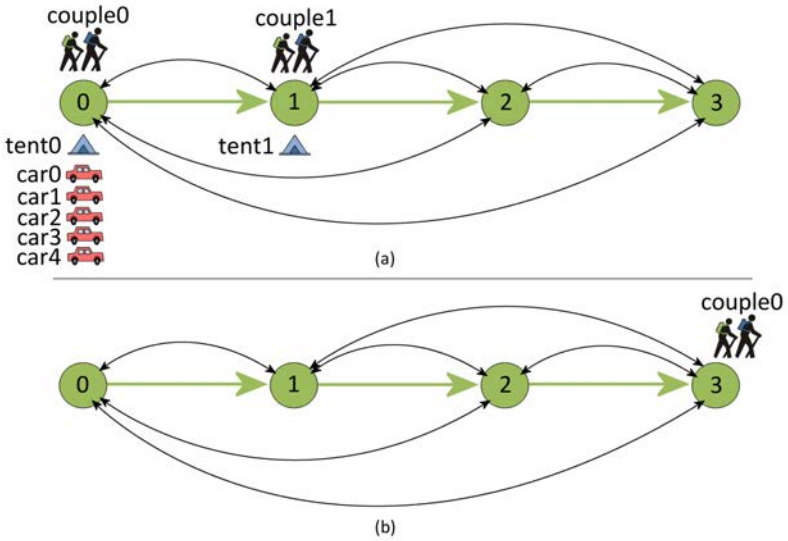

Figure 7.2: (a) Schematic illustrating the initial state of the initial plan in the *Hiking* domain. (b) Schematic illustrating the goal state of the initial plan. The location of cars, tents and *couple1* are not defined in the goal state. In both diagrams, green circles represent places, green arrows represent the hiking route and black arrows represent journeys which may be taken in a car.

Example output

Given the initial state and goals shown in Figure 7.2, 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, the plan is unnecessarily convoluted as the first nine actions result in swapping the locations of *tent0* and *tent1*. There is no need for this as *tent1* was up at *place1* in the initial state, ready for *couple0* to walk to. 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*. The remainder of the plan, o_{10} onwards, is very straight-forward, *girl1* progressively moves *tent1* along the route using *car4*, ready for *couple0* to walk each leg of their hike.

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 3.2).

7.3.2 Transport

The *Transport* domain was introduced in the Sixth International Planning Competition in 2008 (see *IPC-2008 Deterministic Part: HomePage*). The domain 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. By delivering a package, space is created within the truck. Each truck may move between locations with an incurred action cost equal to the length of the connecting road, while collecting and delivering packages each have an action cost of one. 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:

- as in the *Hiking* and AUV domains, the goals are relatively independent. Packages must be transported to specified destinations and so goals may be added representing additional packages to deliver.

- the *Transport* scenario may be logically extended to include resource uncertainty and rewards, making it a suitable candidate for use with our full approach. At present, travelling along a road increases the total cost of the plan by an amount equal to the distance between two locations. It would be very straightforward to associate uncertainty with these travel costs. Equally, it would be a logical extension to associate numeric rewards with the delivery of each package. Such minor changes would result in realistic over-subscribed problems for this domain.

- 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 (e.g. the delivery of package 2 is not conditional on first delivering package 1) and both domains seek to optimise a metric. However,

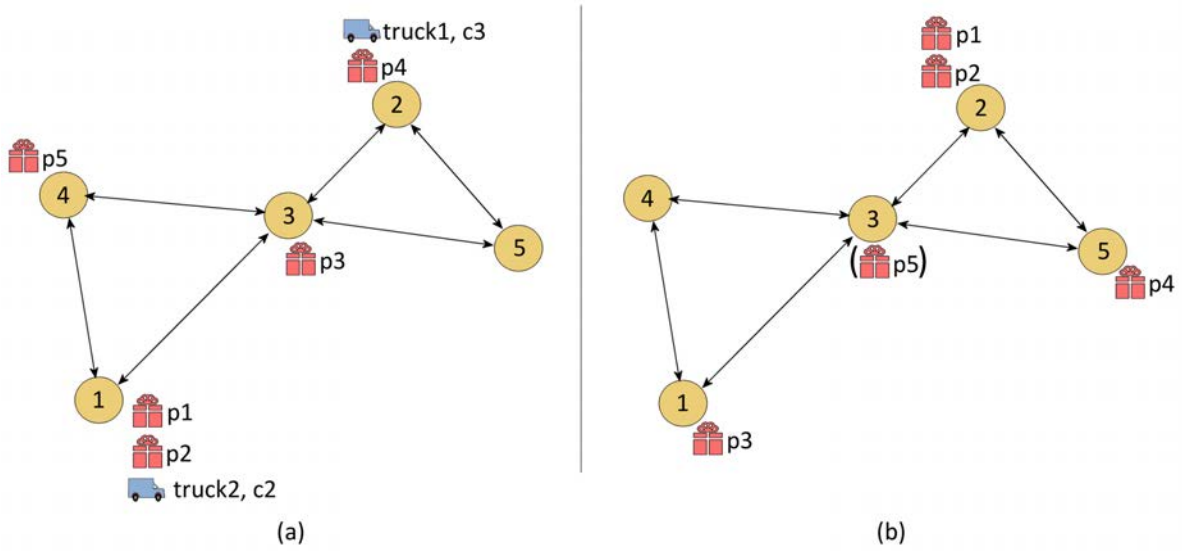

Figure 7.3: (a) Schematic illustrating the initial state of the initial plan in the *Transport* domain. (b) Schematic illustrating the goal state of the initial plan (the additional goal, to move $p5$ is shown in brackets). The location of trucks are not defined in the goal state. In both diagrams, yellow 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 also 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 (whereas the transmission and delivery of datasets in the AUV domain are independent of vehicle location).

Example output

Given the initial state and four delivery goals to complete with two trucks, as illustrated in Figure 7.3, 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 5.6.3). 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.

7.3.3 Blocksworld

Blocksworld is a famous and widely used planning domain. The scenario involves a hand rearranging stacks of blocks on a table from an initial to a goal configuration. A block may only be picked up from the table or unstacked if there are no blocks on top of it and the hand is empty. Once holding a block, it may be put down or stacked on top of other blocks. We chose Blocksworld as an illustrative example of a planning domain which does

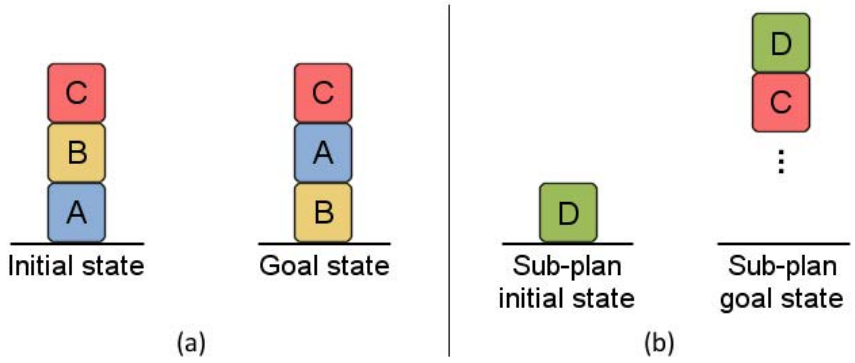

Figure 7.4: Schematic illustrating the initial and goal states of both the original and sub-plans.

not suit our approach. Blocksworld shares very few characteristics with our AUV domain as it is not over-subscribed and the goals are highly interdependent. This greatly limits the applicability of our approach within this domain.

We present an example of a case where our merge algorithm was able to successfully add a goal, as well as a discussion of why the causal structure of Blocksworld problems prevents its wider use in this domain.

Example output

Given the initial state:

`[ontable(a), on(b,a), on(c,b), clear(c), handempty]`

and the goal state:

`[ontable(b), on(a,b), on(c,a)]`

both illustrated in Figure 7.4 (a), Metric-FF generated the following initial plan:

o_1: `unstack(c, b)`

o_2: `putdown(c)`

o_3: `unstack(b,a)`

o_4: `putdown(b)`

o_5: `pickup(a)`

o_6: `stack(a,b)`

o_7: `pickup(c)`

o_8: `stack(c,a)`

After executing the second action, o_2, we then generated a plan to achieve an additional goal, *on(d,c)*, introducing a new block (d) to the problem. The initial and goal states of the sub-problem are shown in Figure 7.4 (b). The plan to achieve this new goal is shown below:

pf_1: `pickup(d)`

pf_2: `stack(d,c)`

The merge algorithm was able to perform a straight-forward merge (requiring neither stitching or the skipping of existing actions), adding block d to the top of the stack of blocks:

o_1: `unstack(c, b)`

o_2: `putdown(c)`

`- - - current state - - -`

o_3: `unstack(b,a)`

o_4: `putdown(b)`

o_5: `pickup(a)`

o_6: `stack(a,b)`

o_7: `pickup(c)`

o_8: `stack(c,a)`

pf_1: `pickup(d)`

pf_2: `stack(d,c)`

The actions of the sub-plan could not be merged immediately at the current state because block d needed to be placed on top of the original stack. Picking up block d earlier would have violated causal-links, requiring block d to first be put down again. Our algorithm is only able to add a goal in the Blocksworld domain if the goal places a single block on top of a stack or on the table, without further interaction between existing blocks, such as placing the new block mid-stack. For example, adding a goal to place block d

between blocks a and b would prevent completion of the existing goal *on(a,b)* because, to complete both goals, block b and block d would both need to be directly on top of block a — an arrangement which is not possible within the Blocksworld domain definition. Such conflicting pairs of goals are 'mutually exclusive' or 'mutex', i.e. the completion of one prevents the completion of the other. As goals within the Blocksworld domain specify the desired position of a block or blocks relative to other blocks within the problem, the goal predicates are often highly interconnected. If the new goal is mutex with the existing goals, no updates will be made to the current plan.

CHAPTER 8

CONCLUSIONS

In this thesis we have considered the problem of automated planning for uncertain, resource-constrained environments, reviewing the state-of-the-art and presenting the development and evaluation of a novel online plan modification and execution monitoring algorithm. Our work was motivated by the challenges of performing long-range and persistent AUV missions, in which a vehicle may travel thousands of kilometres in a single deployment, spending several months at sea (see Chapter 1). In order to fully capitalise on the increased capabilities of these new generation AUVs, such vehicles will require much greater levels of autonomy than current conservative mission formats allow.

We modelled long-duration AUV missions as over-subscribed planning problems, where finite amounts of battery power and data storage space limit the number of tasks, or goals, which a vehicle is able to perform (see Chapter 2). Consequently, a choice of goals to achieve must be made according to their associated resource cost and perceived scientific value. Calculating this trade-off is complicated by the fact that the exact amount of resources required to complete each goal is uncertain and is not known prior to the execution of the plan. This continuous resource uncertainty, 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 (see Section 3.1) and contingency plans (see Section 3.3).

Using an online approach allows us to reason about goals and reduce the risk and

uncertainty associated with the rest of a plan by using observations made during plan execution. Our approach (presented in Chapter 5), 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. If resources are plentiful, plan fragments associated with additional goals may be interwoven or 'stitched' into the current plan using our novel plan merging algorithm (defined in Section 5.7.3), thus allowing the vehicle to utilise surplus resources to maximise reward. Conversely, if the probability of successfully completing the mission falls below a user-defined risk threshold, our algorithm removes goals and associated actions to prioritise mission completion and the safe recovery of the vehicle.

We evaluated our algorithm, presenting the results in Chapter 6 and summarizing the findings in Section 6.6. 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. We 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.

Finally, we discussed the applicability of our approach to existing domains from the planning literature (see Chapter 7) and considered possible future directions for our work. Whilst we formally evaluated our full approach using our AUV domain, we presented annotated examples of applying our plan merging algorithm to widely-published domains. To then enable the reader to consider whether our approach is appropriate for use when attempting to solve their own planning problem, we presented a discussion and summary

of the applicability of our algorithm and others in the literature, given the presence or absence of key domain characteristics.

To conclude, the recent development of a new generation of AUVs brings fresh challenges for the AI planning community. Whilst implementing our approach on a real vehicle was outside the scope of this thesis, the results from testing our approach in simulation are encouraging and confirm it has promise as an effective solution to planning for long-duration missions. Although we concentrated on the motivating example of AUV missions, our work is also applicable to a much larger class of oversubscribed planning domains with resource uncertainty. We formalised the problem of long-duration AUV missions as a planning domain and hope that this contribution will allow other researchers to further both our work and the fields of AI planning and AUV science.

Appendices

APPENDIX A

GLOSSARY

The definitions provided in this glossary are as used in this thesis. Please note that when terms are used in other fields and contexts, the exact definitions may differ from those given here.

Actions — Defined behaviours which an agent may perform to change the state of the world.

Admissible — An admissible heuristic never over-estimates the cost of reaching a goal.

AUV — Autonomous Underwater Vehicle.

AUV problem — As defined in Section 2.5.

Belief state — In a POMDP, due to partial state observability, there is uncertainty about the current state of the world. The belief state represents this uncertainty as a probability distribution over states.

Branch point — Points in the plan at which choices can be made. In contingency planning approaches, contingent branches may be taken. In our approach, modifications may be made to the plan at branch points.

Branching plan — Instead of a single straight line plan, a branching (or contingency) plan is a tree structure, where 'branches' may be taken if certain pre-defined conditions are met.

Catastrophic failure — Total unavoidable failure of the mission. In our AUV domain, there is a uniform probability of catastrophic failure during the execution of all

actions. This is represented as a discount factor.

Causal links — A causal link between two actions represents that a precondition of one is satisfied by the other.

CDF — See **Cumulative Distribution Function**.

Classical planning — Concerned with problems that are deterministic and have a finite number of fully observable states.

CMDP — Continuous Markov Decision Problem.

Complete — A planning algorithm is said to be 'complete' if it will always find a plan if one exists.

Contingency plan — See **Branching plan**.

Cumulative distribution function — The cumulative distribution function, $D(x)$, represents the probability that sampling a random variable will result in a value less than or equal to x (Weisstein, 2015a). The cumulative distribution function of a continuous random variable is the integral of its probability density function.

Current plan — The plan currently selected for execution.

Current state — The set of variables representing the present state of the world.

Datasets — In our AUV domain, datasets may be associated with locations and collected during execution. Returning a dataset results in a reward.

Deterministic effects — Action effects that will always update the state in the same known way, i.e. the effects are repeatable.

Discount factor — A weight, γ, between 0 and 1 that determines how distant rewards should be compared with immediate rewards.

If $\gamma = 0$, only immediate rewards are considered of any value.

If $\gamma = 1$, all rewards are considered equally regardless of plan length.

Domain — Specification of a narrow class of planning problems.

Effects — The updates that will be made to a state upon executing a particular action.

erfc — The 'complementary error function', defined as (Weisstein, 2015b):

$$erfc(z) = \frac{2}{\sqrt{\pi}} \int_z^{\infty} e^{-t^2} dt$$

Expected value — The reward for completing a plan, weighted by the probability of successfully completing it.

Full observability — A problem is fully observable if the current value of all its state variables is known at all times during execution.

Goal — A requirement that the planning problem seeks to satisfy. In the literature, 'hard goals' must be satisfied for a plan to be considered a valid solution. 'Soft goals' are optional and may be ignored, although this typically incurs a penalty.

Goal state — A state in which all goal predicates are met.

Heuristic — A heuristic function provides an estimate of the cost to get to the goal state from a given state.

Initial plan — The original plan prior to execution and any subsequent modifications.

Initial problem — The subset of the overall problem that was used to construct the initial plan.

Initial state — The state prior to the execution of the first action in the plan. The initial state is defined in the initial problem.

Locations — Waypoints that the vehicle may visit or travel through.

Macro-action — A sequence of actions that are treated as a single action. The preconditions of the macro-action are all the unmet preconditions of the action sequence. The effects of the macro-action are those which result from the action sequence.

MDP — Markov Decision Problem.

Metric — An optimisation criteria used to maximise the quality of a plan, which may be specified using PDDL2.1 (Fox and Long, 2003).

Mutex — See **Mutually exclusive**.

Mutually exclusive — Two goals are considered to be mutually exclusive if the completion of one prevents the completion of the other.

Non-classical planning — Concerned with problems that do not meet the assumptions of classical planning. Typically, by involving partial state observability, continuous states and/or stochasticity.

Non-monotonic resource usage — If resource availability both increases and decreases, resource usage is non-monotonic. If resource availability only changes in one direction, e.g. always decreases, it is *monotonic*.

Observations — In a POMDP, as states are not fully observable, observations convey information about the current state.

Observation function — In a POMDP, the observation function defines the probability of making a given observation having performed a particular action in a state.

Optimal value function — As defined by Bresina et al. (2002), the optimal value function shows the expected utility of performing the optimal policy.

Overall problem — The complete over-subscribed problem, that includes all possible goals (i.e. both those present in the initial problem and those which may be added during execution).

Over-subscribed problem — A problem with more goals than can be met with the available resources. Therefore, a choice of which goals to satisfy must be made.

PDDL — Planning Domain Definition Language.

Penalties — Negative rewards.

Policy — A policy specifies the action to take in a given state. Unlike a plan, a policy is not a linear sequence of actions and is used when an action may result in one of multiple possible outcomes.

POMDP — Partially Observable Markov Decision Problem.

$P(success)$ — See **Success probability**.

Preconditions — Requirements that must be met in order to execute an action in a given state.

Problem — An instance of a domain. The problem defines the initial state and goals.

Refinement planning — Considers plan generation as the iterative addition of actions

and/or constraints to the set of all possible plans until only solutions remain.

Replanning — An approach used for planning under uncertainty that discards the current plan and generates an entirely new one, at run-time.

Resources — In our AUV domain, the vehicle has two resources: battery which is non-renewable and memory which may both increase and decrease.

Rewards — A numeric quantity that is used to specify valuable states or goals in both MDPs and over-subscribed planning domains, which a planner seeks to maximise. Negative rewards are referred to as penalties.

Reward function — In both MDPs and POMDPs, the reward function defines the immediate reward for performing an action that results in a transition to a given state.

Sound — A planning algorithm is said to be 'sound' if all plans produced are valid solutions to the problem.

State variables — Variables that comprise the state representation, e.g. $battery = (positive\ real)$.

State-space — The set of all possible states that the vehicle/world may be in.

States — Sets of variables that represent both the vehicle and the world in which it operates.

Stitching plan — A plan fragment that is generated to resolve threats during plan merging. The initial state of the stitching plan is the goal state of the sub-plan to be merged, and the goal state of the stitching plan comprises the unsatisfied preconditions of the remainder of the current plan.

Stochastic effects — Effects that update the state randomly, with a given probability.

STRIPS — Stanford Research Institute Problem Solver.

Sub-plans — Plan fragments that achieve a single goal. A sub-plan is a solution to a sub-problem.

Sub-problems — Problems used for generating sub-plans, where each achieves a single goal within the overall problem, from a state within the initial plan.

Success probability — The probability that the vehicle will complete the execution of all of the actions in a plan without exhausting the available resources.

Survey areas — Locations at which datasets may be collected.

Transition function — In an MDP, the transition function defines the probability that applying an action in a state will result in a transition to any other given state.

γ See **Discount factor**.

APPENDIX B

DERIVATION OF A TRANSITION FUNCTION

If we assume that current resource availability is a fixed known quantity, to calculate the probability that following the execution of an action, the resultant resource availability will fall within a discrete resource level (labelled 'Result' in Figure B.1), we sample the cumulative distribution function (CDF) at both $Result_{max}$ and $Result_{min}$:

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

The probability of transitioning from the current availability to within *Result*, $P(Result|current)$ is therefore:

$$\text{P(Result|current)} = CDF(Result_{max}) - CDF(Result_{min}) \tag{B.2}$$

We do not know the current resource availability at a given point in the plan in advance as this depends on the initial resource availability and the sampled usage of any actions prior to this point. Consequently, we have to represent the resource availability prior to the execution of a particular action in terms of a known quantity, the expected usage of that action, μ. As the expected availability of a resource following the execution of an action a is $current - \mu_a$ (where μ_a is the mean of the expected resource usage of a), in terms of μ, the current availability is therefore 2μ, as illustrated in Figure B.2. The maxima of the Gaussian distribution is at $current - \mu$ when considering total resource

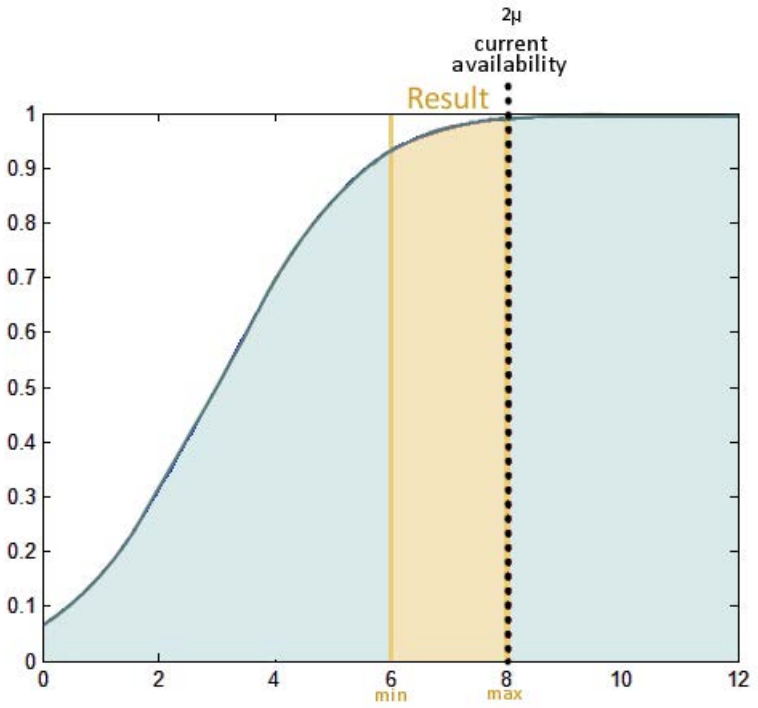

Figure B.1: Graph showing the cumulative distribution function (CDF). A discrete resource level (labelled Result) is shown in yellow. We seek to calculate the probability that, following the execution of the action (whose expected resource usage is represented by the CDF) the resource availability of the vehicle will transition from the current availability (2μ) to within Result.

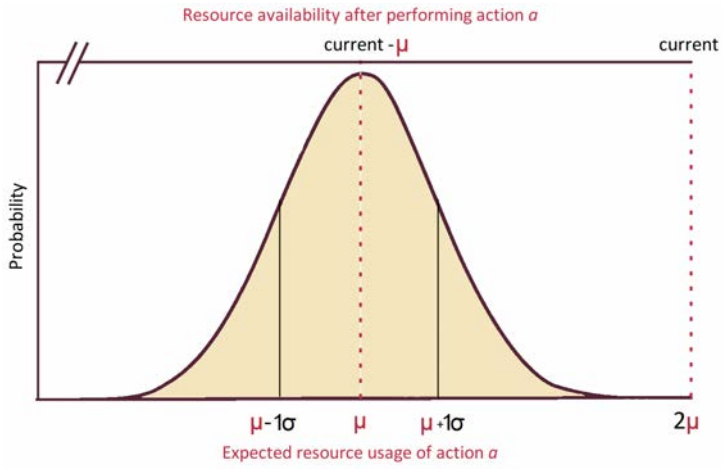

Figure B.2: Graph illustrating the relationship between current resource availability and the mean usage of the action to be executed, μ.

availability (as shown in the top axis of Figure B.2). However, the equivalent point when instead considering the expected resource usage of the action is μ (shown in the bottom axis of Figure B.2). From the graph in Figure B.2, as:

$$current - \mu = \mu \tag{B.3}$$

therefore:

$$current = \mu + \mu \tag{B.4}$$

$$current = 2\mu \tag{B.5}$$

However, when defining the transition function, we do not have a fixed value for current availability and instead wish to know the probability that the resulting resource availability will fall within each discrete resource level given that the current resource availability is within an initial level. We refer to the initial 'Source' level as S and the 'Result' level as R. As shown in Figure B.3, the current availability may range between S_{min} and S_{max}. Therefore, to update Equation B.2 to calculate the probability that a resource availability within S will transition to within R upon performing action a, we integrate the CDF to vary the value of current (2μ) between S_{min} and S_{max}:

$$\mathrm{P(R|S)} = \left(\int_{R_{max}|2\mu=S_{max}}^{R_{max}|2\mu=S_{min}} CDF.dx \right) - \left(\int_{R_{min}|2\mu=S_{max}}^{R_{min}|2\mu=S_{min}} CDF.dx \right) \tag{B.6}$$

Equation B.6 may be used to compute the probability of transitioning from one level to another (or to within the same level), regardless of the number of levels.

The discrete resource levels do not need to be of equal size. As an example, we can use Equation B.6 to calculate the probability of resource availability falling within each of three discrete levels: A, B or C, where A represents the highest level of availability and C the lowest, given the initial resource availability is within level A:

$$\mathrm{P(A|A)} = \left(\int_{2\mu}^{2\mu+|\mathrm{A}|} CDF.dx \right) - \left(\int_{2\mu-|\mathrm{A}|}^{2\mu} CDF.dx \right) \tag{B.7}$$

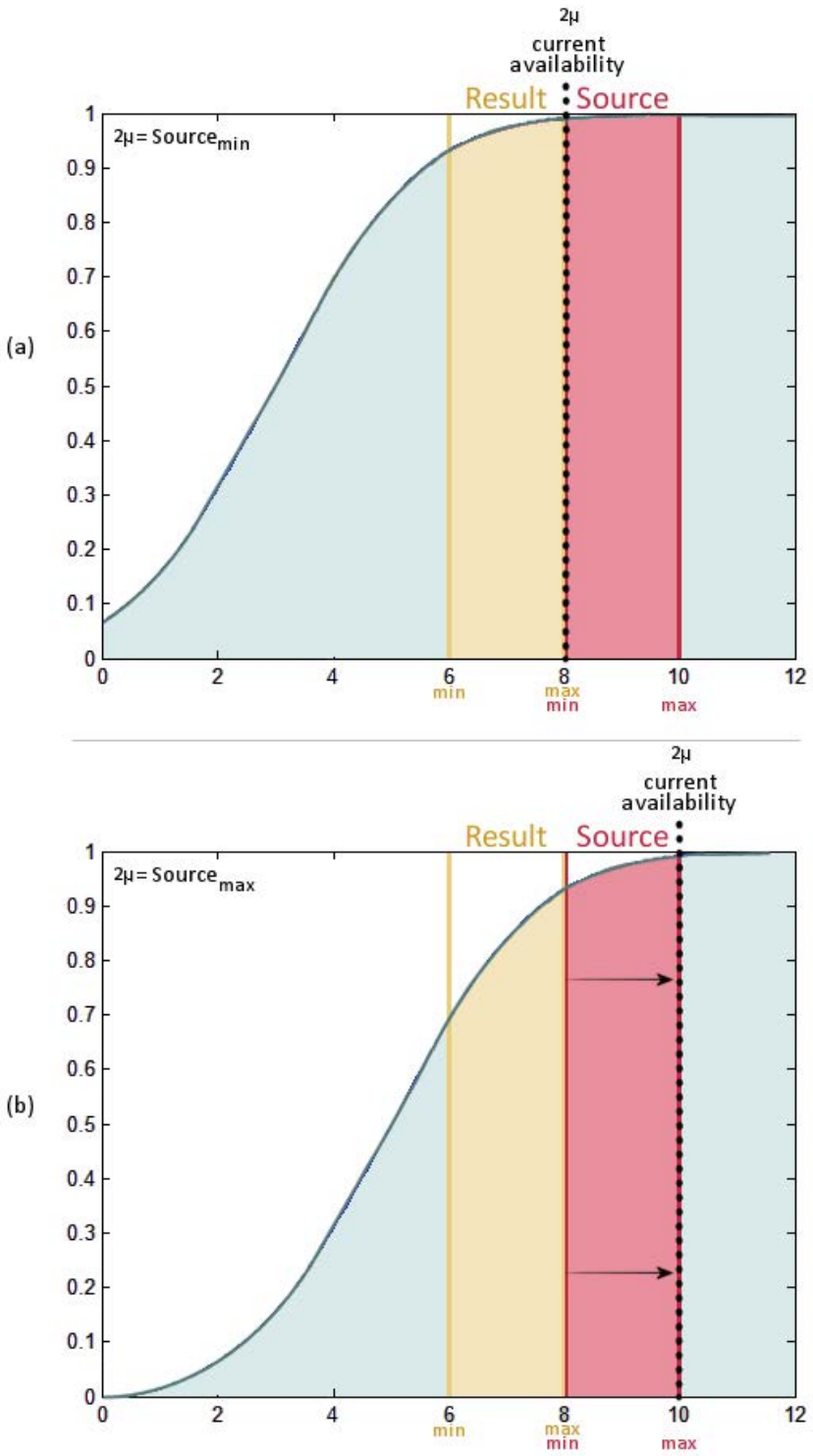

Figure B.3: Pair of graphs showing the limits of the integrals in Equation B.6, to calculate the probability of transitioning to anywhere in 'Result' from anywhere in 'Source'. (a) Shows the definition of the upper limits in Equation B.6. Current availability (2μ) is defined at $Source_{min}$. (b) Shows the definition of the lower limits in Equation B.6. Current availability (2μ) is defined at $Source_{max}$.

$$\mathrm{P(B|A)} = \left(\int_{2\mu-|\mathrm{A}|}^{2\mu} CDF.dx \right) - \left(\int_{2\mu-|\mathrm{A}|-|\mathrm{B}|}^{2\mu-|\mathrm{B}|} CDF.dx \right) \tag{B.8}$$

$$\mathrm{P(C|A)} = \left(\int_{2\mu-|\mathrm{A}|-|\mathrm{B}|}^{2\mu-|\mathrm{B}|} CDF.dx \right) - \left(\int_{2\mu-|\mathrm{A}|-|\mathrm{B}|-|\mathrm{C}|}^{2\mu-|\mathrm{B}|-|\mathrm{C}|} CDF.dx \right) \tag{B.9}$$

The MDP (which used this transition function) was generated using a simplified version of the AUV domain which included only a single monotonically-decreasing resource, battery power. However, if we were to also include the renewable memory resource, the equations used in the transition function would still be valid as actions which increase memory have a negative mean. As we treat battery and memory as independent, we would have to calculate the transition probabilities for each resource separately.

APPENDIX C

CALCULATION OF THE NUMBER OF POSSIBLE CONTINGENCIES

The worst case number of possible plans in our AUV domain, if we only permit goal additions, where G equals the total number of goals in the *overall problem* and J equals the size of the current goal set, is therefore:

$$|plans_{add}| = \sum_{n_1=J}^{G} {}^{G-J}P_{n_1-J} \times \left(\sum_{n_2=n_1}^{G} {}^{G-n_1}P_{n_2-n_1} \times \cdots \times \left(\sum_{n_B=n_{B-1}}^{G} {}^{G-n_{B-1}}P_{n_B-n_{B-1}} \right) \cdots \right) \tag{C.1}$$

The first term of Equation C.1:

$$\sum_{n_1=J}^{G} {}^{G-J}P_{n_1-J}$$

represents the number of possible permutations of additional goals at the first branch point. Subsequent terms represent the remaining possible permutations of additional goals at all subsequent branch points. That is to say, at the first branch point our algorithm may add 0 to $G - J$ of the additional goals. If 0 goals are added, a maximum of $G - J$ goals may then be added at the second branch point. Conversely, if $G - J$ goals are added at the first branch point, 0 goals may be added at all subsequent branch points. Therefore, in the general case, the number of goals which may be added at a given branch point is the total number which may be added, $G - J$, minus the number added at previous branch points. Hence, Equation C.1 is recursive with a depth equal to the

number of branch points B.

If we then separately consider only permitting the removal of existing goals, we produce the following equation:

$$|plans_{remove}| = \sum_{n_1=0}^{GR_1} {}^{GR_1}P_{n_1} \times \left(\sum_{n_2=0}^{GR_2} {}^{GR_2}P_{n_2} \times \cdots \times \left(\sum_{n_B=0}^{GR_B} {}^{GR_B}P_{n_B} \right) \cdots \right) \tag{C.2}$$

where GR_x equals the number of goals which may be removed at branch point x. As we execute the plan, the vehicle will complete goals. These completed goals cannot be removed by the algorithm and thus the number of goals which may be removed at a branch point, GR_x, will decrease to 0 at the end of the plan.

Depending on the characteristics of the domain, the removal of goals may be commutative, i.e. the order in which two or more goals are removed does not change the resulting plan. If this is the case, Equation C.2 must consider combinations, ${}^{GR_x}C_{n_x}$, rather than permutations, ${}^{GR_x}P_{n_x}$ throughout. Adding goals using our merge algorithm (see Section 5.7.3) cannot be guaranteed as commutative for any domain and so Equation C.1 must consider permutations, ${}^{G}P_{n_x}$.

If we were to combine Equation C.1 and Equation C.2 to represent the total number of possible plans, accounting for both the addition and removal of goals, GR_x (the number of goals which may be removed) would no longer strictly decrease as goals which have been added at previous branch points may now be removed. It can therefore be seen that for a large overall goal set and a large number of branch points, the number of contingencies and thus plans which may be executed by the vehicle, is significantly large. This makes the computation of a complete branching plan intractable for our AUV domain.

APPENDIX D

AUV DOMAIN PDDL LISTING

```
;;  AUV domain - PDDL 2.1 domain definition - Catherine Harris
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(define (domain auv)
(:requirements :typing :equality :fluents :adl)
(:types auv location dataset)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; predicates
(:predicates
  (on_surface ?v - auv)               ;; vehicle is on the surface
  (at_loc ?v - auv ?l - location)     ;; vehicle is at a location
  (mission_ended ?v - auv)            ;; status of the mission
  (data_collected ?d - dataset)       ;; vehicle has collected a dataset
  (data_transmitted ?d - dataset)     ;; vehicle has transmitted a
   dataset
  (data_delivered ?d - dataset)       ;; vehicle has delivered a dataset
  (data_with_scientists ?d - dataset) ;; vehicle has delivered data to
   scientists
  (is_neighbour ?l - location ?m - location)  ;; two locations are
   connected
  (data_to_collect ?l - location ?d - dataset);; location has data to
   collect
  (is_end_location ?l - location)     ;; location is end mission
   location
)
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; functions
(:functions
  (battery)
  (memory)
  (reward)
  (reward_end_location)
  (move_battery_usage ?y - location ?z - location)
  (surface_battery_usage)
  (sd_surface_battery_usage)
  (dive_battery_usage)
  (sd_dive_battery_usage)
  (transmit_battery_usage ?d - dataset)
  (collect_battery_usage ?d - dataset)
  (sd_collect_battery_usage ?d - dataset)
  (mean_memory_usage ?d - dataset)
  (sd_memory_usage ?d - dataset)
  (mean_data_reward ?d - dataset)
)
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;; actions
(:action move
  :parameters (?x - auv ?y - location ?z - location)
  :precondition (and (is_neighbour ?y ?z)
          (at_loc ?x ?y)
          (not(on_surface ?x))
          (not(mission_ended ?x))
          (not(at_loc ?x ?z))
          (> (battery) (+ (move_battery_usage ?y ?z) (* (
   move_battery_usage ?y ?z) 0.25)))
  )
  :effect (and (at_loc ?x ?z)
          (not(at_loc ?x ?y))
          (decrease (battery) (move_battery_usage ?y ?z))
  )
)
```

```
(:action collect_data
  :parameters (?x - auv ?z - location ?d - dataset)
  :precondition (and (at_loc ?x ?z)
          (data_to_collect ?z ?d)
          (not(data_collected ?d))
          (not(on_surface ?x))
          (not(mission_ended ?x))
      (> (memory) (+ (mean_memory_usage ?d) (sd_memory_usage ?d)))
      (> (battery) (+ (collect_battery_usage ?d) (
   sd_collect_battery_usage ?d)))
  )
  :effect (and (data_collected ?d)
          (decrease (memory) (mean_memory_usage ?d))
          (decrease (battery) (collect_battery_usage ?d))
  )
)

(:action surface
  :parameters (?x - auv)
  :precondition (and (not(on_surface ?x))
          (not(mission_ended ?x))
          (> (battery) (+ (surface_battery_usage) (
   sd_surface_battery_usage)))
  )
  :effect (and (on_surface ?x)
          (decrease (battery) (surface_battery_usage))
  )
)

(:action dive
  :parameters (?x - auv)
  :precondition (and (on_surface ?x)
          (not(mission_ended ?x))
```

```
            (> (battery) (+ (dive_battery_usage) (sd_dive_battery_usage)))
  )
  :effect (and (not(on_surface ?x))
            (decrease (battery) (dive_battery_usage))
  )
)

(:action deliver_data
  :parameters (?x - auv ?d - dataset)
  :precondition (and (data_collected ?d)
            (on_surface ?x)
            (not(data_transmitted ?d))
            (not(data_delivered ?d))
            (mission_ended ?x)
  )
  :effect (and (data_with_scientists ?d)
            (data_delivered ?d)
            (increase (memory) (mean_memory_usage ?d))
            (increase (reward) (mean_data_reward ?d))
  )
)

(:action transmit_data
  :parameters (?x - auv ?d - dataset)
  :precondition (and (data_collected ?d)
            (on_surface ?x)
            (not(data_transmitted ?d))
            (not(mission_ended ?x))
            (> (battery) (+ (transmit_battery_usage ?d) (* (
   transmit_battery_usage ?d) 0.1)))
  )
  :effect (and (data_transmitted ?d)
            (data_with_scientists ?d)
            (decrease (battery) (transmit_battery_usage ?d))
```

```
          (increase (memory) (mean_memory_usage ?d))
          (increase (reward) (mean_data_reward ?d))
  )
)

(:action end_mission
  :parameters (?x - auv ?l - location)
  :precondition (and (not(mission_ended ?x))
          (on_surface ?x)
          (at_loc ?x ?l)
  )
  :effect (and (mission_ended ?x)
          (when
            (and (at_loc ?x ?l)
            (is_end_location ?l)
            )
          (and (increase (reward) (+ (reward_end_location) 500)))
          )
          (when
            (and (at_loc ?x ?l)
            (not (is_end_location ?l))
            )
          (and (increase (reward) 500))
          )
  )
)
)
```

LIST OF REFERENCES

IPC 2014. Online, available at: `http://helios.hud.ac.uk/scommv/IPC-14/`, no date. Accessed: 2015-02-24.

Seyed Reza Ahmadzadeh, Petar Kormushev, and Darwin G. Caldwell. Autonomous robotic valve turning: A hierarchical learning approach. In *Proceedings of the IEEE International Conference on Robotics and Automation*, ICRA'13, pages 4629–4634, Karlsruhe, Germany, May 2013. Institute of Electrical and Electronic Engineers.

Jorge A. Baier and Sheila A. McIlraith. Planning with preferences. *AI Magazine*, 29(4): 25–36, 2008. AAAI Press.

R Bellman. *Adaptive Control Processes: A Guided Tour*. 'Rand Corporation. Research studies. Princeton University Press, 1961.

Richard Bellman. A Markovian Decision Process. *Indiana University Mathematics Journal*, 6(4):679–684, 1957. Indiana University.

J. Benton, Amanda Jane Coles, and Andrew Coles. Temporal planning with preferences and time-dependent continuous costs. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, ICAPS'12, São Paulo, Brazil, June 2012. AAAI Press.

Julien Bidot, Thierry Vidal, Philippe Laborie, and J. Christopher Beck. A theoretic and practical framework for scheduling in a stochastic environment. *Journal of Scheduling*, 12(3):315–344, 2009. Springer.

Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1-2):281–300, February 1997. Elsevier.

Guido Boella and Rossana 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, pages 183–192, Varna, Bulgaria, September 2002. Springer.

Blai Bonet and Bob Givan. 5th International Planning Competition. Online, available at: `http://ldc.usb.ve/~bonet/ipc5/`, 2006. Accessed: 2015-01-07.

John Bresina, Richard Dearden, Nicolas Meuleau, Sailesh Ramakrishnan, David Smith, and Rich 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, pages 77–84, Alberta, Canada, August 2002. Morgan Kaufmann.

John L Bresina and Richard Washington. Robustness via run-time adaptation of contingent plans. *Proceedings of the AAAI-2001 Spring Syposium: Robust Autonomy*, pages 24–30, March 2001. AAAI Press.

M.P. Brito, N. Bose, R. Lewis, P. Alexander, G. Griffiths, and 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, Southampton, UK, September 2012. Institute of Electrical and Electronic Engineers.

Fred Caswell. *Success in Statistics*. Richard Clay Ltd., Bungay, UK, 2nd edition, 1989.

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, Montpellier, France, August 2012. IOS Press.

Patrick R. Conrad, Julie A. Shah, and Brian C. Williams. Flexible execution of plans with choice. In *Proceedings of the Nineteenth International Conference on Automated Planning and Scheduling*, ICAPS'09, Thessaloniki, Greece, September 2009. AAAI Press.

Philip Cooksey, Frédéric Py, Paul Morris, and Kanna Rajan. An issue in goal addition in continuous robotic plan execution. In *Proceedings of the Workshop on Intelligent Robotic Systems at the Twenty-Seventh AAAI Conference on Artificial Intelligence*, Bellevue, WA, July 2013. AAAI Press.

R. Dearden, N. Meuleau, S. Ramakrishnan, D. Smith, and R. Washington. Incremental contingency planning. In *Proceedings of the Workshop on Planning under Uncertainty at ICAPS'03*, pages 38–47, Trento, Italy, June 2003. AAAI Press.

IPC-2008 Deterministic Part: HomePage. Online, available at: `http://ipc.informatik.uni-freiburg.de/`, no date. Accessed: 2015-02-24.

Colin Devey, Chris German, Sidney Mello, Lucia Campos, Anton Le Roux, Cindy Van Dover, Gwyn Griffiths, Koichi Nakamura, Hidenori Kumagai, Jiabiao Li, and Marcia Maia. Long-range exploration of the ridge crests — workshop report. In *An international workshop — InterRidges Long-range Exploration Working Group*, Southampton, UK, June 2010. InterRidge.

IPC 2014 domains. Online, available at: `http://helios.hud.ac.uk/scommv/IPC-14/domains_sequential.html`, no date. Accessed: 2015-02-24.

Denise Draper, Steve Hanks, and Daniel Weld. Probabilistic planning with information gathering and contingent execution. Technical Report 93-12-04, University of Washington, Seattle, WA, 1993.

Mark Drummond, John Bresina, and Keith Swanson. Just-In-Case Scheduling. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, AAAI-94, Seattle, WA, July 1994. AAAI Press.

S. Edelkamp and J. Hoffmann. PDDL2.2: The language for the classical part of the 4th International Planning Competition. Technical Report 195, Albert-Ludwigs-Universität Freiburg, Institut für Informatik, January 2004.

Zhengzhu Feng, Richard Dearden, Nicolas Meuleau, and Richard Washington. Dynamic programming for structured continuous Markov decision problems. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, UAI'04, pages 154–161, Banff, Canada, July 2004. AUAI Press.

Richard E. Fikes and Nils J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. In *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, IJCAI'71, pages 608–620, San Francisco, CA, September 1971. Morgan Kaufmann.

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

M. Fox, D. Long, and D. Magazzeni. Plan-based policies for efficient multiple battery load management. *Journal of Artificial Intelligence Research*, 44:335–382, 2012a. AAAI Press.

Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20(1):61–124, December 2003. AAAI Press.

Maria Fox, Alfonso Gerevini, Derek Long, and Ivan Serina. Plan stability: Replanning versus plan repair. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling*, ICAPS'06, pages 212–221, Cumbria, UK, June 2006b. AAAI Press.

Maria Fox, Derek Long, and Daniele Magazzeni. Plan-based policy-learning for autonomous feature tracking. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, ICAPS'12, pages 38–46, São Paulo, Brazil, June 2012b. AAAI Press.

Maaten E. Furlong, Dave Paxton, Peter Stevenson, Miles Pebody, Stephen D. McPhail, and James Perrett. Autosub Long Range: A Long Range Deep Diving AUV for Ocean Monitoring. In *Proceedings of IEEE/OES Autonomous Underwater Vehicles*, AUV'12, Southampton, UK, September 2012. Institute of Electrical and Electronic Engineers.

Arthur Gelb. *Applied Optimal Estimation*. Massachusetts Institute of Technology Press, Cambridge, MA, 1974. Written by the Techinical staff of the Analytic Sciences Corperation.

Alfonso Gerevini and Ivan 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, pages 13–22, Toulouse, France, April 2002. AAAI Press.

Christopher R. German, Dana R. Yoerger, Michael Jakuba, Timothy M. Shank, Charles H. Langmuir, and Koichi Nakamura. Hydrothermal exploration with the Autonomous Benthic Explorer. *Deep Sea Research Part I: Oceanographic Research Papers*, 55(2):203–219, 2008. Elsevier.

C.R. German, M.V. Jakuba, J.C. Kinsey, J. Partan, S. Suman, A. Belani, and D.R. Yoerger. A long term vision for long-range ship-free deep ocean operations: persistent presence through coordination of Autonomous Surface Vehicles and Autonomous Underwater Vehicles. In *Proceedings of IEEE/OES Autonomous Underwater Vehicles*, AUV'12, Southampton, UK, September 2012. Institute of Electrical and Electronic Engineers.

Malik Ghallab and Hervé Laruelle. Representation and control in IxTeT, a temporal

planner. In *Proceedings of the Second International Conference on Artificial Intelligence Planning Systems*, AIPS'94, pages 61–67, Chicago, IL, 1994. AAAI Press.

Malik Ghallab, Dana Nau, and Paulo Traverso. *Automated Planning, Theory and Practice*. Morgan Kaufmann, 2004.

B. Gilmour, G. Niccum, and T. O'Donnell. Field resident AUV systems — Chevron's long-term goal for AUV development. In *Proceedings of IEEE/OES Autonomous Underwater Vehicles*, AUV'12, pages 1–5, Southampton, UK, September 2012. Institute of Electrical and Electronic Engineers.

Jonathan Gough, Maria Fox, and Derek Long. Plan execution under resource consumption uncertainty. In *Proceedings of the Workshop on Connecting Planning Theory with Practice at ICAPS'04*, pages 24–29, Whistler, Canada, June 2004. AAAI Press.

Steve Hanks and Daniel S. Weld. A domain-independent algorithm for plan adaptation. *Journal of Artificial Intelligence Research*, 2:319–360, 1995. AAAI Press.

Catherine Harris and Richard Dearden. Run-time plan repair for AUV missions. In *Proceedings of the 7th European Starting AI Researcher Symposium (STAIRS)*, ECAI'14, pages 151–160, Prague, Czech Republic, August 2014. IOS Press.

Ruijie He, Emma Brunskill, and Nicholas Roy. PUMA: Planning under uncertainty with macro-actions. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, pages 1089–1095, Atlanta, GA, July 2010. AAAI Press.

Brett W. Hobson, James G. Bellingham, Brian Kieft, Rob McEwan, Michael Godin, and Yanwu Zhang. Tethys-class long range AUVs - extending the endurance of propeller-driven cruising AUVs from days to weeks. In *Proceedings of IEEE/OES Autonomous Underwater Vehicles*, AUV'12, Southampton, UK, September 2012. Institute of Electrical and Electronic Engineers.

Jesse Hoey, Robert St-Aubin, Alan Hu, and Craig Boutilier. SPUDD: Stochastic planning using decision diagrams. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, UAI'99, pages 279–288, Stockholm, Sweden, July 1999. Morgan Kaufmann.

Jörg Hoffmann. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research*, 20:291–341, 2003. AAAI Press.

Jörg Hoffmann and Bernhard Nebel. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14(1):253–302, May 2001. AAAI Press.

R.A. Howard. *Dynamic Programming and Markov Processes*. Massachusetts Institute of Technology Press, Cambridge, MA, 1960.

Subbarao Kambhampati. Refinement planning as a unifying framework for plan synthesis. *AI Magazine*, 18(2):67–97, 1997. AAAI Press.

Subbarao Kambhampati, Craig A. Knoblock, and Qiang Yang. Planning as refinement search: a unified framework for evaluating design tradeoffs in partial-order planning. *Artificial Intelligence*, 76(1-2):167–238, 1995. Elsevier.

Iain Little and Sylvie Thiebaux. Probabilistic planning vs replanning. In *Proceedings of the ICAPS-07 workshop on the International Planning Competition: Past, Present and Future*, Providence, RI, September 2007. AAAI Press.

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

F. Maurelli, A. Mallios, Y. Petillot, R. Haraksim, S. Krupinski, and P. Sotiropoulos. Portability investigation of space docking techniques for AUV docking. In *Proceedings*

of IEEE OCEANS, Bremen, Germany, May 2009. Institute of Electrical and Electronic Engineers.

Mausam, Emmanuel Benazera, Ronen Brafman, Nicolas Meuleau, and Eric A Hansen. Planning with continuous resources in stochastic domains. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, volume 19 of *IJCAI'05*, pages 1244–1251, Edinburgh, UK, July 2005. Morgan Kaufmann.

Drew McDermott, Malik Ghallab, Adele Howe, Craig Knoblock, Ashwin Ram, Manuela Veloso, Daniel Weld, and David Wilkins. PDDL—the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, October 1998.

Bernhard Nebel and Jana Koehler. Plan reuse versus plan generation: A theoretical and empirical analysis. *Artificial Intelligence*, 76(1-2):427–454, July 1995. Elsevier.

Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, 1985.

Miles 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*, pages 1–5, Aberdeen, UK, June 2007. Institute of Electrical and Electronic Engineers.

J. S. Penberthy and D Weld. UCPOP: A sound, complete, partial-order planner for ADL. In *Proceedings of the Third International Conference on Knowledge Representation and Reasoning*, KR'92. Morgan Kaufmann, October 1992.

Pascal Poupart. *Exploiting Structure to Efficiently Solve Large Scale Partially Observable Markov Decision Processes.* PhD thesis, Department of Computer Science, University of Toronto, 2005.

Louise Pryor and Gregg Collins. Planning for contingencies: A decision-based approach. *Journal of Artificial Intelligence Research*, 4:287–339, 1996. AAAI Press.

Martin L. Puterman and Moon Chirl Shin. Modified policy iteration algorithms for discounted Markov decision problems. *Management Science*, 24(11):1127–1137, July 1978. Institute for Operations Research and the Management Sciences (INFORMS).

Gregg Rabideau, Russell Knight, Steve Chien, Alex Fukunaga, and Anita Govindjee. Iterative repair planning for spacecraft operations using the ASPEN system. In *Proceedings of the International Symposium on Artificial Intelligence Robotics and Automation in Space*, i-SAIRAS'99, Noordwijk, The Netherlands, June 1999. ESA Publications Division.

S.J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Series in Artificial Intelligence. Prentice Hall, 2nd edition, 2003.

Stuart Russell and Eric Wefald. Principles of Metareasoning. *Artificial Intelligence*, 49 (13):361–395, 1991. Elsevier.

Zeyn Saigol. PDDL-driven GraphPlan implementation. Online, available at: `http://www.zeynsaigol.com/software/graphplanner.html`, no date. Accessed: 2015-03-01.

Laura Sebastia, Eva Onaindia, and Eliseo Marzal. Decomposition of planning problems. *AI Communications*, 19(1):49–81, 2006. IOS Press.

M.L. Seto, L. Paull, and S. Saeedi. Introduction to autonomy for marine robots. In Mae L. Seto, editor, *Marine Robot Autonomy*, pages 1–46. Springer, 2013.

Werner Stephan and Susanne Biundo. Deduction-based refinement planning. In *Proceedings of the Third International Conference on Artificial Intelligence Planning Systems*, AIPS'96, pages 213–220, Edinburgh, Scotland, May 1996. AAAI Press.

Gerald J. Sussman. A computational model of skill acquisition. Technical report, Massachusetts Institute of Technology, Cambridge, MA, 1973.

Roman van der Krogt and Mathijs de Weerdt. Plan repair as an extension of planning. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling*, ICAPS'05, pages 161–170, Monterey, California, June 2005. AAAI Press.

Minlue Wang, Sebastien Canu, and Richard Dearden. Improving robot plans for information gathering tasks through execution monitoring. In *Proceedings of International Conference on Intelligent Robots and Systems*, IROS'13, pages 5285–5291, Tokyo, Japan, November 2013. Institute of Electrical and Electronic Engineers.

Richard Washington, Keith Golden, and John L. Bresina. Plan execution, monitoring, and adaptation for planetary rovers. *Electronic Transactions on Artificial Intelligence*, 4(A):3–21, 2000. Royal Swedish Academy of Sciences.

Eric W Weisstein. "Distribution Function." From MathWorld–A Wolfram Web Resource. Online, available at: `http://mathworld.wolfram.com/DistributionFunction.html`, 2015a. Accessed: 2015-03-16.

Eric W Weisstein. "Erfc." From MathWorld–A Wolfram Web Resource. Online, available at: `http://mathworld.wolfram.com/Erfc.html`, 2015b. Accessed: 2015-03-16.

Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6): 80–83, December 1945. International Biometric Society.

Mark Woods, Andy Shaw, Dave Barnes, Dave Price, Derek Long, and Derek Pullan. Autonomous Science for an ExoMars Rover-like Mission. *Journal of Field Robotics*, 26 (4):358–390, April 2009. John Wiley and Sons Ltd.

Russell B. Wynn, Veerle A.I. Huvenne, Timothy P. Le Bas, Bramley J. Murton, Douglas P. Connelly, Brian J. Bett, Henry A. Ruhl, Kirsty J. Morris, Jeffrey Peakall, Daniel R. Parsons, Esther J. Sumner, Stephen E. Darby, Robert M. Dorrell, and James E. Hunt. Autonomous Underwater Vehicles (AUVs): Their past, present and future contributions to the advancement of marine geoscience. *Marine Geology*, 352:451–468, 2014. 50th Anniversary Special Issue. Elsevier.

Qiang Yang. *Intelligent Planning: A Decomposition and Abstraction Based Approach.* Springer, London, 1997.

Sungwook Yoon, Alan Fern, and Robert Givan. FF-Replan: A Baseline for Probabilistic Planning. In *Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, ICAPS'07, pages 352–378, Providence, RI, September 2007. AAAI Press.

Sungwook Yoon, Alan Fern, Robert Givan, and Subbarao Kambhampati. Probabilistic planning via determinization in hindsight. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, AAAI'08, pages 1010–1016, Chicago, IL, July 2008. AAAI Press.

Håkan L. S. Younes and Reid G. Simmons. VHPOP: Versatile Heuristic Partial Order Planner. *Journal of Artificial Intelligence Research*, 20(1):405–430, 2003. AAAI Press.

Håkan L. S. Younes, Michael L. Littman, David Weissman, and John Asmuth. The first probabilistic track of the International Planning Competition. *Journal of Artificial Intelligence Research*, 19:851–887, 2005. AAAI Press.