STAIRS 2014
U. Endriss and J. Leite (Eds.)
© 2014 The Authors and IOS Press.
This article is published online with Open Access by IOS Press and distributed under the terms of the Creative Commons Attribution Non-Commercial License.
doi:10.3233/978-1-61499-421-3-151

Run-time Plan Repair for AUV Missions

Catherine HARRIS [a] and Richard DEARDEN [a] [1]
[a] *University of Birmingham, UK*

Abstract. Autonomous underwater vehicle (AUV) missions are challenging due to limited or no communication with the vehicle and uncertainty about what may be encountered during the mission. The vehicle has limited battery power and memory to store datasets and does not know how much of these resources its actions will consume. Managing the uncertainty by building contingency plans (as in previous work) is infeasible owing to the large number of contingencies. Purely online planning is also difficult because of restricted computation. We propose a mixture of off-line planning and on-line plan repair, presenting an approach for deleting parts of a plan when resources are tight, or stitching new plan fragments into an existing plan when additional resources are available. We discuss this novel approach using a simulated AUV mission domain.

1. Introduction

In recent years autonomous underwater vehicles (AUVs) have become increasingly popular for a wide variety of applications. As the cost of deploying a vehicle and the risk of loss or damage are often high, AUV missions typically consist of simple pre-scripted behaviours. Although designed to minimise risk to the vehicle and its scientific cargo, these behaviours are inevitably overly-conservative, reserving a significant proportion of battery as a contingency. Remedying this problem is difficult owing to our lack of knowledge about conditions at the ocean bottom. This makes it very hard to predict how much power a particular activity will use or how much memory datasets will consume. As improvements in battery technology allow much longer deployments, these pre-scripted missions are proving increasingly limiting. Finding ways to adapt the mission during execution has the potential to significantly improve the capabilities of these vehicles.

We consider AUV science missions, in which the vehicle collects a number of datasets from locations of interest. The AUV has limited battery power, which is required for all actions and cannot be recharged during a mission. The vehicle also has limited memory for storing data, although this can be reused if a dataset is transmitted mid-mission. A key challenge of the problem is that the resources used by each action are uncertain, so a good initial plan can fail if actions take more resource than expected, or waste resources if usage was lower than expected.

Similar problems have been discussed in the past (see Section 2). Contingency planning has been examined in multiple papers [7,13,8,2], but owing to the high uncertainty in the problem we prefer a replanning approach. However, since battery used for com-

[1]E-mail: C.A.Harris.1@cs.bham.ac.uk, richard.dearden@gmail.com

putation is no longer available for action, we aim to minimise online computation, using plan repair to efficiently update the existing plan rather than computing a new full plan.

The approach we take is to continually monitor the resource usage during execution to determine if and when updating the plan may be beneficial. Prior to execution, at key points in the plan we generate individual sub-plans for each goal in the problem, both those included in the current plan and those which may be added. These sub-plans provide an estimate of the resource cost and reward of each goal, whether adding or removing it from the plan, which we use as a heuristic to aid online plan modification. During execution, if resource usage deviates significantly from expected, the heuristic informs the selection of goals for addition or removal by representing the trade-off between resource cost and reward. Goal removal is achieved by removing all actions that only contribute to that single goal. Adding a goal involves combining the pre-generated sub-plan for the chosen goal with the existing plan, interleaving the two sets of actions until the causal structure of the resultant plan is valid. Should additional actions be required to achieve this, the system generates new plan fragments to join the two plans together. To minimise online computation, we use a classical deterministic representation for plan generation, treating resource usage as discrete (using the mean of the distribution) and then evaluate the resulting plan in the probabilistic model.

2. Related Work

The planning literature contains many examples of work with similar motivations to our own [5,6,17]. Representing a simplified version of the Mars rover domain as a Markov decision process (MDP) with continuous energy usage and time, Bresina et al. [2] compute the optimal value function for all contingencies within a branching plan. This represents the expected utility of each branch for all combinations of energy and time and can be used as a policy, dictating the optimal branch to take at run-time, given resource availability. In a follow-up paper, Feng et al. [9] use an approximate MDP approach, but this is restricted to very small problems as a value function over all continuous variables must be computed for each discrete state. Due to the high number of possible contingencies and the size of the state space in both the AUV and Mars rover domains, the use of a MDP solver is computationally infeasible for realistically-sized problems.

Solutions based on the construction and execution of branching plans are popular for domains with resource uncertainty [7,13,8,2]. Bresina and Washington [3] 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 is constructed offline by mission operators. Contingent branches are added to define the actions to take should a predictable action failure occur during execution. Whilst our approach also augments an initial plan with alternative options prior to execution, the combination of continuous resource uncertainty and a large oversubscribed goal set in our AUV domain would require many plan branches to be generated and considered at each step in the plan. We instead delay the decision, of which goals to add or remove from the problem at each point in the plan, until run-time when the resource usage may be observed. We reduce the number of goal combinations to consider (and consequently plan branches to generate) by only considering those which are applicable given the observed resource availability.

Bidot et al. [1] present a scheduling framework capable of managing uncertainty and potential failures. An initial partial schedule is generated over a short-time horizon, based

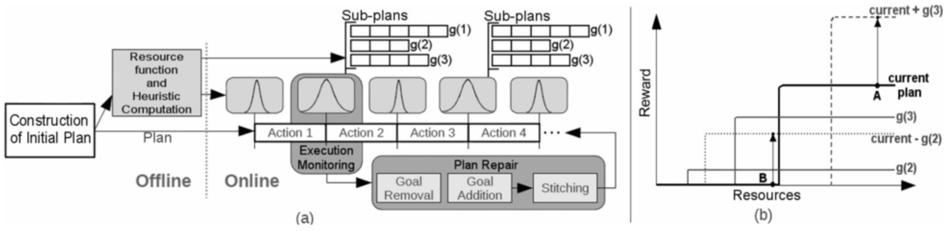

Figure 1. (a) Schematic of our approach. Execution monitoring uses the precomputed heuristic and observed resource usage to trigger plan repair. (b) Heuristic for triggering plan repair, plot shows the mean resource usage of the current plan and plan fragments: If the current resources are at A, the unused resource is sufficient to add in goal $g(3)$, increasing the expected reward. If current resources are at B (i.e. below the mean requirement of the current plan), removing $g(2)$ reduces the resource requirement and increases the expected reward.

on the most probable observations. If the duration uncertainty causes the expected value of the cost function to increase significantly, another schedule is generated online with additional ordering constraints to reduce computation. 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.

In plan repair van der Krogt and de Weerdt [19] use ideas from refinement planning [16,15], but propose an additional strategy, unrefinement planning, which allows the removal of constraints or actions which prevented goals from being met. If the current plan is no longer a solution, they create candidates for refinement by first unrefining the current plan, removing actions which depend on either the initial or goal state.

Fox et al. [10] describe plan repair as “adapting an existing plan to a new context whilst perturbing the original plan as little as possible”. They define a ‘plan stability’ metric to measure the difference between the original and repaired plans, and adapt the local-search based planner LPG [12] to use this metric when evaluating partial plans for refinement. They compare their plan repair strategy to that of replanning and find that repair produces plans more efficiently and with much greater stability than replanning.

Nebel and Koehler [18] show that in theory, the costs of modifying an existing solution to suit a new problem are higher than generating an entirely new plan. As the AUV domain has many constraining factors—limited computational resources, a large state space— we expect online plan repair to be a more effective solution in practice, outweighing the theoretical inadequacies of plan modification.

3. Algorithm

The general structure of our approach is shown in Figure 1 (a). Offline, a plan is generated along with estimates of the expected reward as a function of resources and a heuristic to aid online plan modification. During plan execution, the actual resource usage is monitored and if there is a significant deviation, plan repair is invoked to either add or delete goals. Each step is described below. For ease of exposition, we treat a plan as a sequence of actions or interchangeably as the sequence of states that occur if the plan is executed from the initial state.

Initial Plan Generation: We assume that the initial plan is generated prior to the deployment of the vehicle when resources are plentiful. As the AUV domain is oversubscribed, the initial goals must also be selected. For each promising non-mutex combination of goals a plan is generated using an external planner. Metric-FF [14] was chosen because it is capable of generating satisficing plans for the classical deterministic representation of the AUV domain (i.e. where uncertain costs are represented using the mean of the distribution). The expected reward of each plan is then computed using Monte Carlo simulation. The plan with the highest expected reward is selected for execution. The causal links in the plan are also computed so they are available during plan repair.

Computing the Heuristic: We gain the most information about the probability of a plan completing successfully, and thus its expected reward, after performing actions with the largest resource usage uncertainty. Consequently, it is at these points that considering plan modifications is most beneficial. To aid online plan modification, we compute a heuristic prior to execution by generating a sub-plan to complete each goal in the over-subscribed problem individually. We use the expected state following the most uncertain actions as the initial state for each sub-plan, as shown in Figure 1 (a). The ratio of the estimated resource cost vs reward of adding or removing each goal at these points is used to inform the decision of which modifications to make and when.

Plan Execution Monitoring: After each action is executed the current resources are observed and used to update the expected reward of the remaining plan. If the usage was higher than expected, there may be insufficient resources to complete the plan causing the success probability and expected reward to decrease. Significant gains in expected reward can be made by removing an existing goal, thus increasing the probability of achieving the remaining goals and finishing the mission successfully. If current resources are at point B in Figure 1 (b), they are below the mean required to complete the plan (indicated by the sudden drop in expected reward) and consequently the expected reward of this plan is now very low. By removing the goal $g(2)$ from the current problem, we decrease the expected resource usage of the resulting plan (as indicated by the line labelled $current - g(2)$) whilst increasing its expected reward. If resource usage was lower than expected, the success probability (and consequently expected reward) of the remaining plan increases. This situation is illustrated by point A in Figure 1 (b), where the heuristic suggests that given the current resource levels, there are sufficient resources to accommodate the new goal $g(3)$ and still complete the existing plan. By combining the plan fragment for $g(3)$ with the existing plan (as indicated by the line labelled $current+g(3)$) the expected reward and expected resource usage of the resulting plan both increase.

Goal Removal: In removing a goal from the planning problem, we can also remove any actions which were only present to achieve that goal. Given a goal g to remove, we construct the set D_g of actions to remove by first setting $D_g = \{a_g\}$, where a_g is a dummy action with the precondition g. We then repeatedly add to D_g any action a such that all causal links from a lead to actions 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.

Since finding D_g is fast, we select the goal to delete by computing D_g for each goal g which the heuristic suggests is plausible, starting with the goal whose sub-plan minimises the ratio of expected value to mean battery usage, to find the goal which maximises the expected value of the plan once D_g is removed. To approximate expected value, we first estimate the success probability of the remaining plan by combining the cumulative density functions (CDF) representing the resource usage of each action into a single distribution. The combined CDF represents the probability of the vehicle completing the plan given the current resources. The expected reward of a plan is the sum of the probability of achieving each goal multiplied by the reward gained. While battery and

Algorithm 1 Merging a plan fragment with an existing plan.

```
Input: $pf$—plan fragment, $p$—existing plan, $Z$—list of valid merges, initially $\emptyset$, filled during recursion.
 1: function MERGEPLANS($Z, pf, p$)
 2:   $valid \leftarrow false$
 3:   if $p$ contains all goals then
 4:     $Z \leftarrow Z + p$, return true        ▷ $p$ is a solution
 5:   else if $pf = \emptyset$ then return false  ▷ backtrack
 6:   else
 7:     $a \leftarrow$ first action in $pf$,
 8:     $pf \leftarrow pf - a$
 9:     $X \leftarrow$ all states in $s$ that meet preconditions of $a$
10:     for all $x \in X$ do
11:       if inserting $a$ at $x$ causes no threats then
12:         $merge \leftarrow p$ with $a$ inserted at $x$
13:         Update $s, c$ for new action
14:         $valid \leftarrow mergePlans(Z, pf, merge)$
15:       end if
16:     end for
17:     if $valid$ = false AND $a$ achieves no goals then
18:       $pf \leftarrow pf - a$                   ▷ skip $a$
19:       $valid \leftarrow mergePlans(Z, pf, merge)$
20:     else, return valid
21:     end if
22:   end if
23: end function
```

memory are not strictly independent (as battery may be used to transmit data in order to increase available memory), we assume independence for ease and speed of calculation.

Adding Goals: The system attempts to merge the plan fragment pf associated with the chosen goal and current state into the current plan p, interleaving actions to create a valid solution for the combined goal set. To create a valid merged solution, the recursive algorithm, shown in Algorithm 1, takes the first action from the new plan fragment and compares its preconditions with each state in the original plan, producing a list of candidate *merge points*—states which meet the preconditions of the action. The algorithm then checks whether causal links in the current plan are threatened by inserting the action at each merge point. If merging an action causes a threat, e.g. to the preconditions of another action, the algorithm searches the plan fragment for an action whose effect restores the threatened link and checks whether this effect is then maintained until the end of the plan fragment. If this is the case, the threat is marked as resolvable. If no links are threatened or all threats are resolvable, we add the action to the plan at the current merge point, update the plan's causal links and call the function again to insert the next action in the fragment. This continues until all valid merge points for each action in the plan fragment have been considered. Each complete merged plan 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.

If no valid plan orderings are found, 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 existing plan as the goal. By returning the state to this point, the stitching plan resolves all threats caused by the sub-plan. After generating the stitching plan, the two plans are again passed to Algorithm 1 to interleave them into the original plan.

4. Analysis

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. For replanning, Metric-FF [14] was used in the same configuration as when generating the initial plan, sub-plans and stitch-

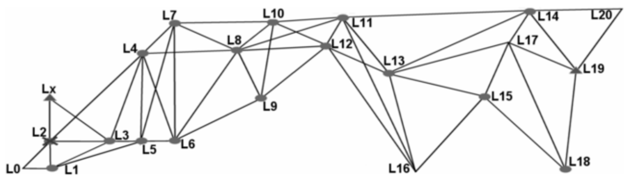

Figure 2. Connectivity of locations and datasets. Circles represent data-collection goals in the initial problem. Triangles represent goals added during runs of the experiment. The crossed $L2$ indicates the removal of this goal during the experiment.

ing 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 [11] problem file 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 2) and the same initial plan (with 66 steps) 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.

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 2) which is already on the route taken by the existing plan, thus requiring minimal plan modification. The second additional goal, to 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, for $D19$, 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. The modification algorithm's performance was very consistent, with a standard deviation of only 0.01 seconds, compared to 56.45 when replanning. This large difference is caused by replanning failing to return within the two-minute time-out period (as shown by the large plateau area, labelled a and b in Figure 3 (a)) 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 c and b). However, as it does not consider the wider search space, plan modification may miss solutions which replanning

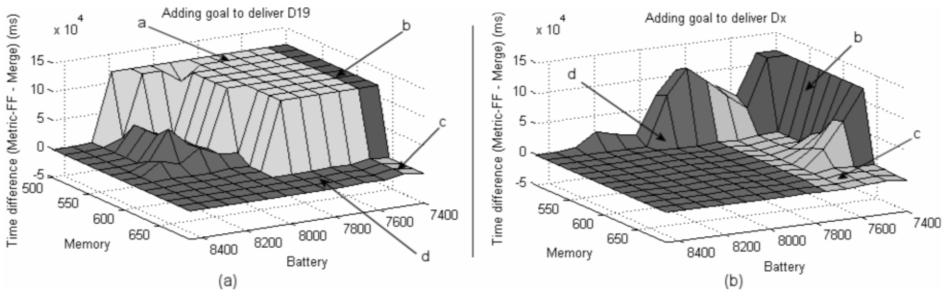

Figure 3. (a) Time difference between the two approaches as a function of available resources when adding the en-route data-collection goal $D19$. (b) Time difference when adding the off-route data-collection goal Dx. In both cases, the area labelled b represents cases where both Metric-FF and the modification 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.

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. We believe this difference in plan generation time is due to whether Metric-FF is able to replan using enforced-hill-climbing (as when resources are plentiful and do not significantly restrict the search) or has to resort to slower best-first search [14].

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. 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; however, when only considering successful runs, replanning was an average of 0.95 seconds faster. 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 [11] problem file and additional actions using Metric-FF. The 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.

Discussion: The reason stitching is required when adding the off-route goal is illustrated in Figure 5 (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 $move(L0, L1)$ 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 exist-

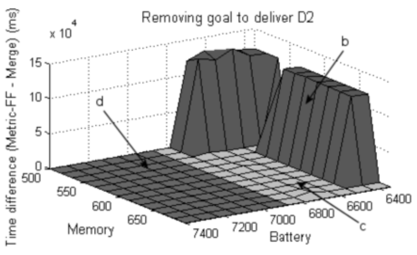

Figure 4. Time difference between the two approaches as a function of available resources when removing the goal to deliver $D2$. Area b represents cases where both Metric-FF and the modification algorithm failed to find a valid solution; c, cases where only plan modification failed; and d, cases where both were successful.

ing plan as the goal state. Actions added to the resulting plan from the stitching plan are shaded in Figure 5 (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 2). 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 goal by making minimal changes, as shown in Figure 5 (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 1. 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, resulting in a valid plan.

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 4). 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 4) and 0.13 seconds slower when comparing only successful runs. 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 5 (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,

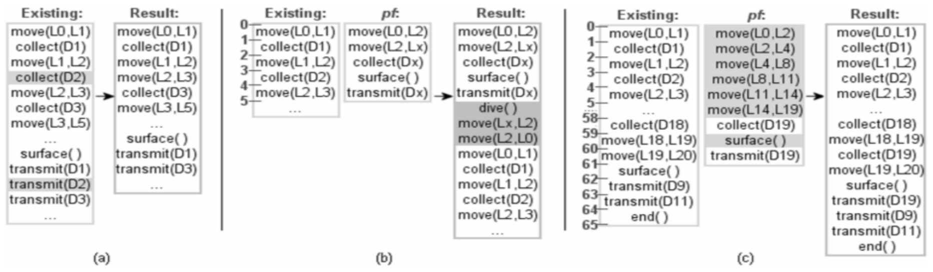

Figure 5. Plans used when: (a) removing the goal to deliver $D2$, removed actions are shaded; (b) adding the goal to deliver Dx, actions from the stitching plan are shaded; (c) adding the goal to deliver $D19$, skipped actions are shaded.

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.

5. Conclusions

We have presented a novel approach which uses a mixture of offline planning and online plan repair to produce solutions for oversubscribed domains with significant resource uncertainty. We analysed the plan modification algorithm, showing its greater speed and consistency in comparison to replanning. Minimising the time and battery required for onboard computation is important because resources used for planning are then unavailable for completing goals. As the modification algorithm is constrained by the causal structure of the initial plan, preventing large changes to the overall mission, it was unable to find solutions for some resource and goal combinations where replanning was successful. However, Fox et al. [10] argue that plan repairs should make minimal changes to the existing plan, which we consider especially true for high-risk domains such as AUV and Mars rover missions. Investigating why the AUV community has yet to widely adopt adaptive mission planning, Brito et al. [4] found uncertain vehicle behaviours to be the largest concern (39.7%) of expert AUV operators, which our approach addresses.

Our analysis of the plan modification algorithm shows promise for use within the AUV domain. Further analysis of the overall system based on complete mission runs is needed to establish the effectiveness of the full approach. As the system is domain independent, we also plan to investigate the performance of both the plan modification algorithm and the system as a whole on other oversubscribed domains, such as logistics problems.

References

[1] 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).

[2] 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*, pp. 77–84, Alberta, Canada, (2002). Morgan Kaufmann.

[3] John L Bresina and Richard Washington, 'Robustness via run-time adaptation of contingent plans', in *Proceedings of the AAAI-2001 Spring Syposium: Robust Autonomy*, Stanford, CA, (2001).

[4] 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)*, Southampton, (September 2012). IEEE.

[5] Rebecca Castano, Tara Estlin, Robert C. Anderson, Daniel M. Gaines, Andres Castano, Benjamin Bornstein, Caroline Chouinard, and Michele Judd, 'Oasis: Onboard autonomous science investigation system for opportunistic rover science', *Journal Field Robotics*, **24**, 379–397, (May 2007).

[6] Steve A. Chien, Russell Knight, Andre Stechert, Rob Sherwood, and Gregg Rabideau, 'Using iterative repair to improve the responsiveness of planning and scheduling', in *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling Systems*, pp. 300–307, Breckenridge, Colorado, (April 2000).

[7] 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*, Montpellier, France, (August 2012). IOS Press.

[8] 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*, Thessaloniki, Greece, (September 2009). AAAI.

[9] 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*, pp. 154–161, Banff, Canada, (2004). AUAI Press.

[10] 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*, pp. 212–221. AAAI, (June 2006).

[11] Maria Fox and Derek Long, 'PDDL2.1: An extension to PDDL for expressing temporal planning domains', *Journal of AI Research*, **20**(1), 61–124, (December 2003).

[12] Alfonso Gerevini, Alessandro Saetti, and Ivan Serina, 'Planning through stochastic local search and temporal action graphs in LPG', *Journal of AI Research*, **20**, 239–290, (2003).

[13] 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*, Whistler, Canada, (June 2004). AAAI.

[14] Jörg Hoffmann, 'The Metric-FF planning system: Translating "Ignoring delete lists" to numeric state variables', *Journal of AI Research*, **20**, 291–341, (2003).

[15] Subbarao Kambhampati, 'Refinement planning as a unifying framework for plan synthesis', *AI Magazine*, **18**(2), 67–97, (1997).

[16] 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).

[17] 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, California, (July 2009). AAAI.

[18] Bernhard Nebel and Jana Koehler, 'Plan reuse versus plan generation: a theoretical and empirical analysis', *Artificial Intelligence*, **76**(1-2), 427–454, (1995).

[19] 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*, pp. 161–170, Monterey, California, (June 2005). AAAI.