Intelligent Robotic Systems: Papers from the AAAI 2013 Workshop

An Issue in Goal Addition in Continuous Robotic Plan Execution

Philip Cooksey
California State University
Monterey Bay
Seaside, California, 93955
pcooksey@csumb.edu

Frédéric Py
MBARI
7700 Sandholdt Road,
Moss Landing, California, 95039
fpy@mbari.org

Paul Morris
NASA Ames Research Center
Moffett Field, California, 94035
Paul.H.Morris@nasa.gov

Kanna Rajan
MBARI
7700 Sandholdt Road,
Moss Landing, California, 95039
Kanna.Rajan@mbari.org

Abstract

Robotic plan execution has traditionally assumed that goals are articulated prior to mission execution. As robots have become persistent and increasingly moved into real-world environments, this assumption is not necessarily true; for instance a user can decide to give a new objective to the robot for inclusion in the plan being formulated, add newer goals, or modify others queued for execution. In most systems this leads, at best, to a suboptimal final plan or possibly to the exclusion of objectives, either of which could have been avoided, should the robot have executed its initial plan differently. We first articulate and then demonstrate a preliminary approach to this problem motivated by a marine robotics domain. We do so with an execution policy that is sufficient to disambiguate actions for execution within a flexible temporal continuous plan execution system. The resulting algorithmic complexity is linear in the number of actions and causal links of an existing partial plan.

Introduction

As autonomous robots become more robust and persistent, the likelihood of receiving new or modified objectives *during* an ongoing mission is high. Generative continuous planning, integrated within the robot's architecture, is one approach to enhance persistence while dealing with evolving objectives and unanticipated situations. Earlier works, (Ambros-Ingerson and Steel 1988; Haigh and Veloso 1998; Alami et al. 1998; Muscettola et al. 1998; Chien et al. 1999; Finzi, Ingrand, and Muscettola 2004; Py, Rajan, and McGann 2010), have contributed control architectures, which embed planning engines. They support a rich representation dealing with durative actions, while ensuring partial plans are flexible for robust execution (Lemai-Chenevier 2004).

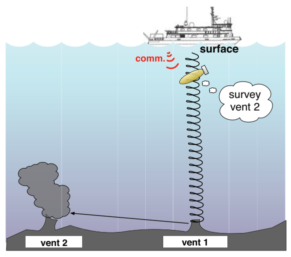
While these approaches have been necessary to correctly execute a *flexible temporal plan*, they view the agent itself as synchronously fulfilling goals provided a priori. They do not take into account the introduction of new goals which could alter the mission state and execution during mission execution. Consequently, the natural choice was to make the agent execute its actions as early as the temporal plan would allow, so that the the robot would finish its plan as early as possible providing more "temporal room" for potential change in the plan. Such an assumption has proved to be very efficient in dealing with unanticipated execution threats with an exception in dealing with dynamic controllability (Morris, Muscettola, and Vidal 2001) — acting as early as possible keeps the remaining partial plan flexible. Moreover, should the plan break, the robot has more time left to identify and execute an alternate recovery strategy. An early dispatch policy has therefore proved to be the most efficient approach for the robot to robustly execute its plan when the set of goals for the mission are fully instantiated.

Our experience in marine robotics shows that in a number of instances, the full set of objectives of the robot is not known a priori. While our missions last on the order of a day, it is often feasible to have an initial plan that could be executed in a few hours. For example, a typical mission can start with the scientific goal of collecting data on an area for half of the mission and an operational goal to be at the recovery location by the end of the mission. During this time, a scientist receives and analyzes data collected by the robot, which allow him to produce new goal(s) for dispatch to the robot, while still in the water. While starting an action proactively can often add flexibility to the current plan, it can also at times negatively impact the outcome should a new goal directly conflict with an action undergoing execution.

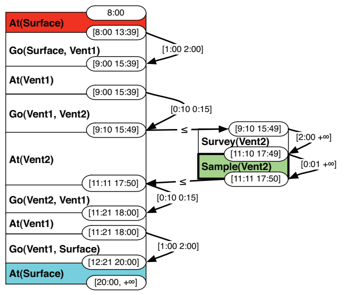
In this paper we focus on a study demonstrating that strict execution policies can present problems when new goals can emerge at any time within mission scope; we are agnostic to the planning process itself. We provide a simple solution that exploits temporal plan structure to identify the execution policy of the next potential set of actions.

An Illustrative Example The context of this work is in the oceanographic domain with an autonomous underwater vehicle (AUV) with an illustrative issue arising with dynamic goal emergence. The AUV can autonomously plan and execute goals sent from shore. In this domain, an AUV

Copyright © 2013, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.



(a)



(b)

Figure 1: (1a) illustrates the domain and (1b) an initial plan for the problem. The AUV is initially at *Surface* at 8:00 and the mission is expected to complete by 20:00 with the goal — marked with thick borders — to sample *Vent 2* and the operational goal to be back at the surface for recovery by mission end.

can either transit from one location to another within the water column, or survey a location in order to collect data about a feature of interest; an example being a hydrothermal vent depicted in Fig. 1. Note that the AUV must pass through *Vent1* to get to *Vent2*. The vehicle is deployed initially with the scientific objective of sampling *Vent 2* and the operational goal of being back at the *Surface* by the end of the mission which lasts 12 hours. The support ship can send short commands to the AUV and receive limited data via acoustic communications. The traditional approach to such AUV surveys has been to separate exploration and exploitation into two different surveys (Yoerger et al. 2007). By improving how the vehicle handles the inclusion of new goals, we can greatly improve the efficacy of such surveys.

Given the initial problem in Fig. 1a, the onboard planner on the AUV produces the flexible temporal plan shown in Fig. 1b for execution. However, at any time during the mission scientists can decide, for example, that they also want to *sample Vent 1* as a consequence of data processing during the traverse from the surface to *Vent 2*. Time on site for such bathymetric exploration is limited and expensive, so dynamic goal generation is a reality. Finding a balance between executing a plan action *proactive*ly or waiting, is the dichotomy that the plan executive needs to resolve. Not doing so impacts the mission efficiency.

Exclusively *deferring* the actions is not acceptable since it results in the vehicle sitting at the surface, diving only at the very last minute. Subsequently, if a new goal arrives there is no time left to execute outstanding objectives, leaving the robot to exclude one of the goals or fail to be at the surface as planned by 20:00. Equally, the *proactive* approach can be problematic, since the vehicle will go through the actions of the plan and be back at the surface early (12:21). Even if the new goal is provided early enough (by 14:00), the vehicle will have to dive back in order to survey and sample *Vent 1*. Since most vents are usually deep (1500 meters or deeper often taking 3 or more hours to dive to depth) such a situation results in two extraneous actions, of at least an hour each, that could have been avoided should the AUV have stayed at the bottom. Neither of these approaches are satisfactory; *defer*ment can lead to inefficiency, while both

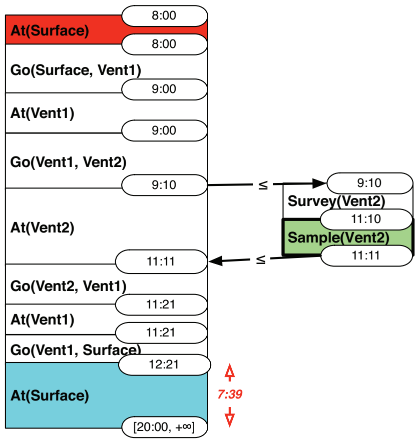


Figure 2: A proactive solution for the plan from Fig. 1b resulting in a Surfacing event at the end of mission lasting in excess of 7 hours.

deferment and *proactive* execution can lead to insensitivity to additional goals. Fig. 2, is an example of a solution aimed at *proactive* execution.

After analyzing the objectives, we note that these policies are because of divergent goal semantics. While visiting *Vent 1* was reflecting a *scientific need*, the goal to be back at the surface is an *operational requirement* to avoid risk due to the battery running out. While fulfilling both goals are equally important for mission success, they do not carry the same notion of urgency. It is desirable for the AUV to visit vents as early as possible but the motivation to return to the surface is at the end of the mission. Therefore, a more balanced policy would be to alternate between the two different approaches depending on what objective(s) the next action contributes to. The AUV could have been *proactive* until it starts to survey *Vent 2* and then *defer* heading back to the surface, continuing to survey the same area, until it either receives a new goal or the latest start time of the *Go* to *Vent1* action is reached ($\sim$ 17:50). In the event of a new goal, the resulting mission scenario would then be more efficient in terms of its *makespan*. We use the latest start time which does not compensate for uncertainty in the plan and can be easily swapped with another policy. Our approach revolves around marking mission objectives with a policy depending on the exploration of the plan structure, during plan execution, to identify if the action is linked to a goal–in which case it needs to be executed *proactively*–otherwise leading towards a *deferred* execution policy for the action.

Related Work

Research in temporal plan execution is not new. However, dealing with anticipatory dispatching has not been a focus of prior work in the literature. It becomes important in our context for three reasons. First, our system is embedded in a real-world robotic explorer which can sustain itself for extended periods in which new goals can evolve. Second, our integrated architecture `T-REX` (McGann et al. 2008a; Py, Rajan, and McGann 2010; Rajan and Py 2012) plans *continuously* during mission execution. Finally, the situated agent encodes plans in a rich and flexible representation which allows such cost-effective inference. While our onboard controller is influenced by work cited here, this work is complementary to much of our earlier efforts.

In (Lemai-Chenevier 2004), the emergence of new goals triggers replanning; most architectures handle emergent goals similarly, often related to an off-nominal situation such as plan execution failure. Prominent work in this area is related to dispatchability in simple temporal networks (STN) so efficient temporal propagation prior to execution encapsulates appropriate planning constraints (Muscettola, Morris, and Tsamardinos 1998). When dealing with least-commitment planning this decision is usually deferred to the plan executive. However, the executive does not address what value should be set for a given time-point from a possible interval of values.

(Morris, Muscettola, and Vidal 2001) proposes an algorithm that inserts **wait** constraints within the temporal network during planning for any uncontrollable event that may occur. (Gallien and Ingrand 2006) also uses a similar approach to reduce makespan. They do so in the context of an exogenous event–such as the duration of a navigation task which depends on external factors–where it is necessary to defer actions for correct plan execution. Work in soft temporal constraints with preferences (Khatib et al. 2001) on dispatching has another related approach with preferable start times. However, they are plagued by poor performance (Bartak 2002). Moreover the solution proposed optimizes the plan *a-priori* reducing plan flexibility leading to potential execution time failure, which is not suitable for real-world embedded systems. In most literature, the executive exclusively uses a *proactive* execution policy motivated by the intuition that it will give more time in the future should an unexpected situation occur. However, most such prior work focuses on improving plan quality but does not consider the impact of execution on the agent to handle unanticipated goals.

The efforts above decouple execution from planning, unlike our work where planning is not only continuous, but execution is intertwined with planning. Our interest is in deferring temporal decisions to the executive to adapt the plan to new objectives without heavily relying on plan-repair or re-planning solutions that could impact reactivity in an embedded system.

Executing a Flexible Temporal Plan

The focus of this paper is on the study of unanticipated threats due to new goals arising *during* mission execution, while preserving room for their inclusion and subsequent execution. Our robot generates and dispatches a partial plan, with a lower level controller that further decomposes these plans towards atomic actions; these tasks ultimately actuate hardware on our AUV. The world responds by sending sensory feedback as *observations* up through the hierarchical controller (McGann et al. 2008b; Rajan and Py 2012). Details of the controller are beyond the scope of this paper and are not directly relevant to the concepts described.

In order to explain our execution method, we first consider some basic definitions related to a plan consistent with (Nau, Ghallab, and Traverso 2004). A plan has a given set of operators and states. Operators link a set of states, conditions, to a new set of states, effects. We refer to this link as a causal link in the plan. Recursive causal links build a path in the plan from an initial state into a goal state. The plan structure is similar to the `STRIPS` model in the definition of a plan.

Within the plan, each state has a *start*, T_s and *end*, T_e time. We refer to these as *timepoints*. Each timepoint is therefore a *flexible* interval of time $[t, T]$, which includes all possible values between t and T. A *token* is a property that holds within two such timepoints, which represents a state (Py, Rajan, and McGann 2010); one to start and the other to end the state. Such flexible temporal plans are in a dispatchable form as defined by (Muscettola, Morris, and Tsamardinos 1998). The role of the executive is to identify and decide as time advances, which timepoint within the dispatched plan can be properly instantiated without generating inconsistency within the plan. If no solution exists, the executive can only report to the planner and wait for a new plan. In the literature, this decision was often supported by a strict policy for every timepoint. However, as noted in the example this can lead to issues in when to dispatch actions.

To handle such problems, we assume that some states have a defined execution policy. This execution policy impacts how their start time will be handled by the executive. For example, an early execution policy implies a preference to set the token's start with the smallest possible value without breaking the plan. We consider that these execution policies are determined beforehand by a user for a limited number of tokens. Typically, the tokens that have a defined execution policy are the goal states of the plan. This leaves states without an execution policy, in which case we either derive a policy or use a default. In the rest of this paper our default policy is set to a strict latest start; however any policy can be used.

We also assume that the plan can evolve in the future due to the introduction of new goals. This has an impact on how to handle plan execution. While deciding when to initiate a state transition within the plan, one needs to ensure that execution will not limit the ability of the agent to treat emergent goals. In light of this, the agent should attempt to balance the impact of execution on the next available action. We choose to execute as early as possible or delay with the eventuality that new goals might occur. Going to *Vent2* early is viable; however, going back to the surface too early would result in blocking the AUV at the Surface until 20:00. The solution which provides the most flexibility is for the agent to alternate between the two policies.

Algorithmic Description

As a new token's start-timepoint is to be executed, the executive needs to evaluate how it relates to the policies of other states in the plan. Intuitively, if a state S_1 was generated by (or causally linked to) another state S_2, which has a defined policy, then S_1 should be executed using the policy of S_2. Therefore, while evaluating the token within the partial plan representing S_1, the executive needs to do a forward search of the causal links related to the token to see if it leads to another token with a defined policy. Algorithm 2 will discuss the case when a token is connected to two or more different policies. Note that our example only uses two policies for execution.

Algorithm 1 The function $ExecutionPolicy$ uses the $SearchForPolicy$ to determine the execution policy for Token T. We use the two policies of earliest and latest start.

```
function EXECUTIONPOLICY(Token $T$)
    Policy $P$ = SEARCHFORPOLICY($T$)
    if $P$ = Policy(EarliestStart) then
        return Dispatch $T$
    else if $T$ start upper bound $\leq$ current tick then
        return Dispatch $T$
    else
        return Don't dispatch $T$
    end if
end function
```

Algorithm 1 is key in determining how a token should be executed, and is called at every execution cycle while evaluating the next set of tokens to be executed. It also plays the role of the executive and, while specific to our example, does not impact the generality of Algorithm 2.

Algorithm 2 The function $SearchForPolicy$ does a forward search along the causal links to determine a policy for Token T.

```
function SEARCHFORPOLICY( Token $T$ )
    if $T$ has Policy then
        return $T \rightarrow Policy$
    else
        Set $Policies$ = {}
        List $Fringe$ = { Actions that $T$ is Condition of }
        Mark $T$
        for all Action(s), $A$, in $Fringe$ do
            Mark $A$
            for all Effects, $E$ of $A$; $\neg Marked(E)$ do
                Mark $E$
                if $E.Policy \neq \emptyset$ then
                    $Policies$ = $\{E.Policy\} \cup Policies$
                end if
                List $L$ = { Action(s) that $E$ is Condition of }
                $Fringe$ = $Fringe \cup \{a \in L; \neg Marked(a)\}$
            end for
        end for
        return SelectPolicy($Policies$)
    end if
end function
```

To inform Algorithm 1, we use Algorithm 2 which searches the causal links connected to token T using a forward-search. If a token with a defined policy is found, then it is added onto the set of potential execution policies for T. After searching all the tokens connected to T, we call $SelectPolicy$ to determine which execution policy should be returned. The $SelectPolicy$ function takes the set of all policies found during the search and returns the resulting unique policy in order to arbitrate conflicts. In our current implementation we always return *LateStart* policy if this set is empty, otherwise we select the element of the set with the highest fixed priority ($EarliestStart > LateStart$); an alternate implementation could be introduced without impacting the generality of our algorithm. The algorithmic complexity of Algorithm 2 is $O(N + E)$ where N is the number of tokens and E the number of causal links that connect these tokens in the plan.

An alternate version of Algorithm 1 that propagated policies through causal links during planning was also attempted. While functionally equivalent, the overhead during backtracking–requiring removal of propagated information during search–along with continuous plan refinement process in our system, did not demonstrate any significant performance benefit for evaluation.

Experiments

The experiments follow a mission similar to the original plan given in Fig. 1b. The AUV needs to *Sample Vent2* and return to the *surface* by the end of the mission. The AUV must also be at the *surface* around 13:00. Fig. 3 illustrates the resulting plan after being processed by Algorithm 1 using `T-REX` which synthesizes temporal plans. We have also defined the policy of earliest start time to the goals in plan.

As in Fig. 3, the AUV is *At Surface* initially. At execution time, the next *Go* token will be dispatched early, since it encounters the *Sample Vent2* goal when searching the causal links. This occurs similarly for all tokens that are starred. *Go(Surface)* gets deferred, because it is not connected to a goal using the forward search. Consequently, the resulting execution requires the AUV to stay at *Vent2* rather than heading to the *Surface* immediately.

We then introduced a new goal to *Sample Vent1* at 11:30 with the resulting plan shown in Fig. 4. Since there is not enough time to *Sample Vent1* and be at the surface by 13:00, the goal is placed after the AUV visits the surface. Propagation results in marking the tokens, which were thus far deferred and are now causally linked to the new goal of *Sample Vent1*, as having the policy of the new goal. The resulting

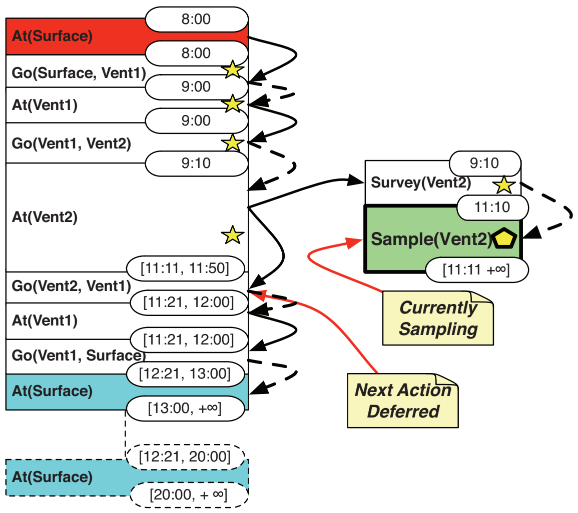


Figure 3: Solid lines indicate conditions and dashed lines indicate effects. Pentagons indicate goals and stars indicate tokens that were deduced as *proactive*.

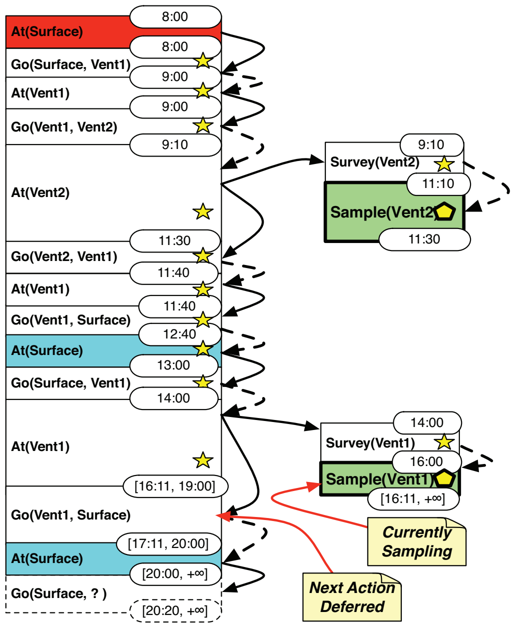


Figure 4: The solution after receiving external request for the plan from Fig. 3

starred tokens get dispatched proactively which includes all tokens related to the mid-day surfacing token.

Meanwhile, the tokens related to the final surfacing remain deferred and won't be executed until 19:00 allowing the vehicle to sample *Vent1* for a duration of 2:50 hours. After returning to the *Surface*, our planner inserts a partially instantiated token *Go(Surface, ?)* (dashed in the figures). This is an artifact resulting from the plan model which specifies that an *At* token is necessarily followed by a *Go*. However, our algorithm does not dispatch this token as it is not connected to a goal and its upper bound start time ($+\infty$) will not be met within the scope of the mission.

Performance Analysis

In our experiments, the simulated AUV missions ran on a Linux virtual machine running on a MacBook Pro allocated one core from a 2.33 GHz Intel Core 2 Duo processor. A short mission duration (200 seconds) was selected to allow rapid assessment with multiple runs; increasing mission duration has no significant impact.

Fig. 5 shows a mission run that is similar to the example in Fig. 1b. To increase plan complexity a total of 8 intermediate locations were added between the *surface* and *vent 1*, with all evaluations taking less then 1 millisecond. The largest performance spikes were due to the need to evaluate the dispatching condition for an *At* token which is followed by a *Go* token resulting in two forward searches within the plan structure. This is because the dispatch window is larger than the minimum duration of the *At* token; since our plan uses a *Go* immediately after, this *Go* token also needs to be evaluated.

We also identified a clear trend of a decrease in time as the agent comes closer to the completion of either *Goal #1* or *Goal #2*. This is correlated to the reduction in search distance toward these goals as we execute new tokens. We see an important variability in execution time which is likely due to context switching between processes and threads within our agent; having multiple cores in the virtual machine is likely to remove this variability. The bump in timing between time-step 56 and the introduction of *Goal #2* at 127 is because the algorithm is continuously searching the graph from the next *Go*, which during this period, is not connected to a goal. The resulting exhaustive search in the plan graph during this period is on an average above 0.2 milliseconds. At tick 127, the *Go* is no longer deferred and gets dispatched.

Fig. 6 shows the distribution of execution time for 500 simulated missions. It shows that 90% of the time our algorithm completed its search in less than 0.2 milliseconds. The distribution is long tailed with the worst performance between 2 and $13ms$ occurring for less than 5% of the runs. Such performance is exhibited when the tokens evaluated are meant to be deferred, which is not critical to the overall execution.

Discussion and Future work

Increased robustness in hardware has led to persistence of robots often in hostile environments such as our oceans. The versatility of onboard autonomy has resulted in the need to anticipate and adapt to goals during mission execution. Most existing autonomy approaches fall short of dealing with the ensuing dichotomy of proactive dispatch versus deferment. Typically, the solution pursued is to integrate such knowl-

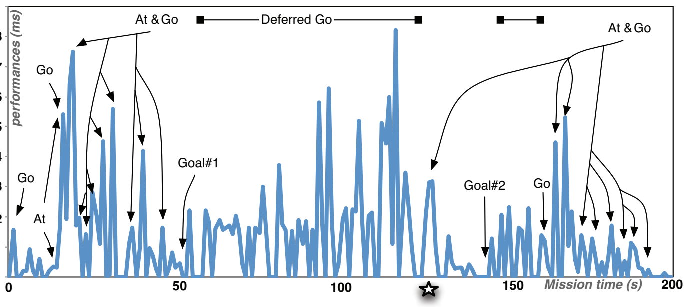


Figure 5: A mission run that shows the time needed for dispatching. The star represents when Goal #2 is received and integrated into the plan. Lines with boxes at the end represent the deferred action *Go* Because of space, we didn't put an arrow to every *At*. Our concern were the peaks that showed where the system had to dispatch both *At* and *Go*.

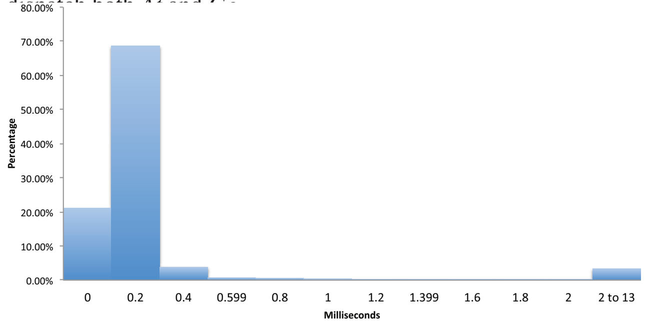


Figure 6: Algorithm 1 was timed at every increment (1 Hz) for 500 simulated missions. One goal was inserted into the plan at random during the mission. The histogram shows the distribution of the time, which is heavily skewed to the left.

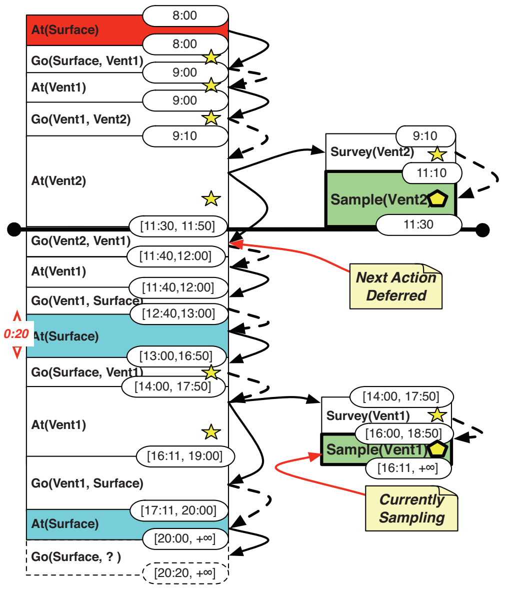


Figure 7: The ideal plan solution for Fig. 4. The thick line at 11:30 indicates current time. Since *At Surface* is constrained to be at 13:00, the AUV would be stuck for 20 minutes.

edge within the system domain description which often results in a more complex model making it harder to maintain and therefore error prone. This motivates our approach to provide an efficient solution to deduce execution policy based on the supplementary information namely that of defined policies in the plan. The work in this paper focuses on plan structure in abstraction without direct consideration of timing constraints applied to each goal in the plan. We are able to use breadth-first search within the existing plan structure to select different execution policies for each token within the plan. By doing so, we can improve overall mission execution by avoiding proactivity that could lead to inefficiency in handling new goal requests.

A number of open issues remain to be explored, however. The causal link traversal proposed is agnostic to the temporal constraints applied to timepoints of the plan. This design choice was made as we consider our approach as a heuristic supporting executive decisions. Therefore the management of the temporal constraints and their propagation remained the reponsibility of the executive itself. Still, including temporal constraints could greatly increase the quality of our solution. For instance, in Fig. 4, where the consequent was a *Surface* token, planning instantiated the goal of surveying *Vent1*. Currently, the result would make all tokens related to the *Surface* token proactive due to the goal immediately following it. This can potentially lead to premature surfacing which would prohibit the agent from acting on goals until 13:00. Typically the *Surface*, which is to end after 13:00, is acting as a guard on the propagation. Therefore, ideally only those tokens that are directly after this *Surface* token should have been marked as proactive as shown in Fig. 7, while keeping actions preceding 13:00 deferred.

Such refinement can be done only by considering the temporal constraints applied to plan timepoints. Should the *Surface* not have been constrained to be around 13:00, starting its execution proactively would not have blocked the vehicle at the surface. We hope to explore how the analysis of the simple temporal network supporting the plan can help identify and deduce a token's execution policy. The insertion of **wait** actions during planning in (Morris, Muscettola, and Vidal 2001) and the possibility of using STNUs (STNs with uncertainty) within our plan representation, also needs investigation with extensions to execution time changes in plan structure.

These refinements are likely to improve overall execution in our agent despite uncertainty in the number and nature of goals during mission execution. In our domain such a requirement is critical as our AUV is in a dynamic environment and it is far more cost-effective to use the limited resources (scientists and on-station time) judiciously.

References

Alami, R.; Chatila, R.; Fleury, S.; Ghallab, M.; and Ingrand, F. 1998. An Architecture for Autonomy. *Intnl. Journal of Robotics Research*.

Ambros-Ingerson, J., and Steel, S. 1988. Integrating Planning, Execution and Monitoring. *Proc. 7th American Assoc. for Artificial Intelligence (AAAI)*.

Bartak, R. 2002. Modelling soft constraints: a survey. *Neural Network World* 12(5):421–431.

Chien, S.; Knight, R.; Stechert, A.; Sherwood, R.; and Rabideau, G. 1999. Integrated Planning and Execution for Autonomous Spacecraft. *IEEE Aerospace* 1:263–271.

Finzi, A.; Ingrand, F. F.; and Muscettola, N. 2004. Model-based Executive Control through Reactive Planning for Autonomous Rovers. In *Proceedings of IROS*.

Gallien, M., and Ingrand, F. 2006. Controlability and makespan issues with robot action planning and execution. In *ICAPS, Workshop on Planning under Uncertainty and Execution Control for Autionomous Systems*.

Haigh, K., and Veloso, M. 1998. Interleaving Planning and Robot Execution for Asynchronous User Requests. *Autonomous Robots* 5:79–95.

Khatib, L.; Morris, P.; Morris, R.; Rossi, F.; et al. 2001. Temporal constraint reasoning with preferences. In *International Joint Conference on Artificial Intelligence*, volume 17, 322–327.

Lemai-Chenevier, S. 2004. *IxTeT-EXEC: Planning, Plan Repair and Execution Control with Time and Resource Management*. Ph.D. Dissertation, Institut National Polytechnique de Toulouse, Toulouse, France.

McGann, C.; Py, F.; Rajan, K.; Ryan, J. P.; and Henthorn, R. 2008a. Adaptive Control for Autonomous Underwater Vehicles. In *AAAI*.

McGann, C.; Py, F.; Rajan, K.; Thomas, H.; Henthorn, R.; and McEwen, R. 2008b. A Deliberative Architecture for AUV Control. In *Intnl. Conf. on Robotics and Automation*.

Morris, P.; Muscettola, N.; and Vidal, T. 2001. Dynamic Control Of Plans With Temporal Uncertainty. In *International Joint Conference on Artificial Intelligence*, 494–502.

Muscettola, N.; Nayak, P.; Pell, B.; and Williams, B. 1998. Remote Agent: To Boldly Go Where No AI System Has Gone Before. *AI Journal* 103:5–48.

Muscettola, N.; Morris, P.; and Tsamardinos, I. 1998. Reformulating temporal plans for efficient execution. In *Proceedings of 6th Internationl Conf. on Principles of Knowledge Representation and Reasoning (KR)*.

Nau, D.; Ghallab, M.; and Traverso, P. 2004. *Automated Planning: Theory & Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Py, F.; Rajan, K.; and McGann, C. 2010. A Systematic Agent Framework for Situated Autonomous Systems. In *9th International Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*.

Rajan, K., and Py, F. 2012. T-REX: Partitioned Inference for AUV Mission Control. In Roberts, G. N., and Sutton, R., eds., *Further Advances in Unmanned Marine Vehicles*. The Institution of Engineering and Technology (IET).

Yoerger, D.; Jakuba, M.; Bradley, A.; and Bingham, B. 2007. Techniques for Deep Sea Near Bottom Survey Using an Autonomous Underwater Vehicle. *The International Journal of Robotics Research* 26(1):41–54.