Anticipatory Continuous Plan Execution in a Robotic Agent

Paper #

Abstract

Introduction

As their mission scope becomes longer, autonomous robots are more prone to receive new objectives during the course of their mission. Situated planning is a common mean within robotics architectures to reevaluate the mission startegy to better fullfill the current agent objectives and readapt it when unanticipated situations occur or fresh new objectives are communicated to the robot. Based on this belief, earlier work done in `IPEM` (Ambros-Ingerson and Steel 1988), `ROGUE` (Haigh and Veloso 1998), and the LAAS architecture (Alami et al. 1998) have all helped to make situated planning possible. These initial sucesses lead to modern goal directed architectures that have sucessfully worked in the real-world such as the Remote Agent Experiment (Muscettola et al. 1998a), the Autonomous Spacecraft Experiment (Chien et al. 1999) or more recently the Teleo Reactive Executive (McGann et al. 2008; Py, Rajan, and McGann 2010). Nowadays, these architectures support planning engines that have a rich representation of both durative actions and resources while keeping a plan that is flexible enough to be robustly executed (Lemai-Chenevier and Ingrand 2004).

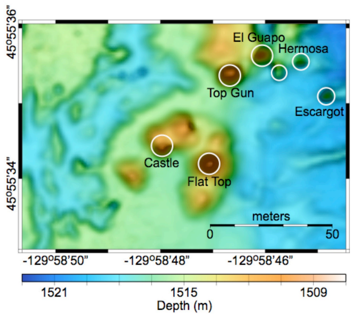
The planning community dis also provide a lot of work on efficients approaches to dispatch the plan to the executive in order to ensure that its execution is made robustly and soundly (Muscettola, Morris, and Tsamardinos 1998; Morris, Muscettola, and Vidal 2001). While these approaches are necessary to properly execute a flexible temporal plan, they do not really consider the fact that new goals can be introduced later. This oversight is understandable as these goals are by nature unknown and therefore impossible to integrate within the plan limiting the ability to formally take them into account. Instead the natural choice was to make the executive execute its action as early as the plan and current context allow with the assumption that, by doing so, the robot will finish its plan as early as possible giving then more room for potential future objectives. While this solution works in the classical case where all the goals of the agent are known a priori, it can be problematical when new objectives are added during the course of mission execution. For example, the agent may prematurely leave an area before new objective for this area are formulated resulting on superfluous actions from the robot. On the other hand, waiting inordinately could wast time and leave less flexibility for coping with diverse execution scenarios. In this context, it is important for the agent to make a distinction between actions that can be taken proactively versus other actions that may not be urgent.

In this work, we propose a systematic approach to allow the agent to make such a distinction for each action by tracing the causal relations between goals within the current plan. This allow us to determine the best dispatch strategy based on the nature of the goals this action contributes to. While our approach manipulate the plan structure, it is worth mentionning that it is focused on the plan execution within the agent than the process of producing such a plan. This alow us to consider planning at an bastract level and provide an approach that can be applied to the execution of a large class of planners.

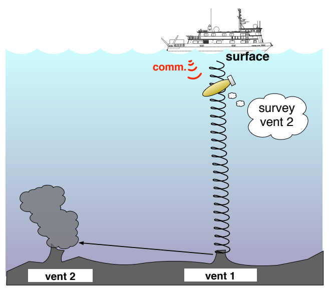
The structure of this paper is as follow. We introduce an example illustrating the problem of flexible plan execution. We then discuss previous works and how they tend to focus on a single policy or not consider the posibile emergence of new goals during the mission. We then present algorithmic solutions that allow to decide at execution when actions should be started. Finally, after presenting the results, we conclude and discuss potential directions for future research.

An Illustrative Example

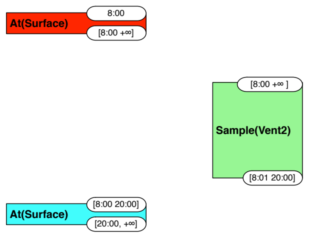
We define a simple authomous underwater vehicle (AUV) mission that will illustrate potential issues arising with dynamic goal emergence and motivate our apprach. In this domain we have an AUV that can do two specific actions: Go from one location to another within the water mass and survey a location in order to sample data of a feature of interrest such as a vent similar to the one depicted in Fig. 1a. Our domain is illustrated in Fig. 1b, the vheicle is deployed initially with the science objective to sample the *Vent 2* and the operational goal to be back at the *Surface* by the end of the mission which lasts 12 hours.



(a) Bathymetry of vent sites off of NW United States



(b) An illustration of our domain



(c) Initial problem

Figure 1: 1a shows bathymetry of actual vent sites off the coast of Oregon. (1b) illustrate the domain along with the initial partial plan for this problem (1c). In this domain our AUV is initially at *Surface* at 8am snd the mission will complete by 8pm.

Given this initial problem (Fig. 1c) the AUV planner produces the flexible plan given in Fig. 2 which can then be executed by the system. However, at any time during the mission scientists can decide for example that they also want to *sample Vent 1* (for exmaple due to the processing of the data the AUV sent through the accoustic link while flying above this location). Such possibility raises the issue of balancing the decisions during plan execution between staring one plan action as early as possible — which we will call *proactive* – or wait until the action should necessarily start – called later. There is a clear difference between the two approaches but when should, for example, the AUV wait or start early and how either will impact the mission execution when the new objective will be integrated by the AUV ?

Deferring all the actions is obviously not acceptable as it results the vehicle sitting at the surface for the most part of the mission and starting to diving only at the very last minute (14:39 in our plan). As a result if the new goal arrives after this time there's no time

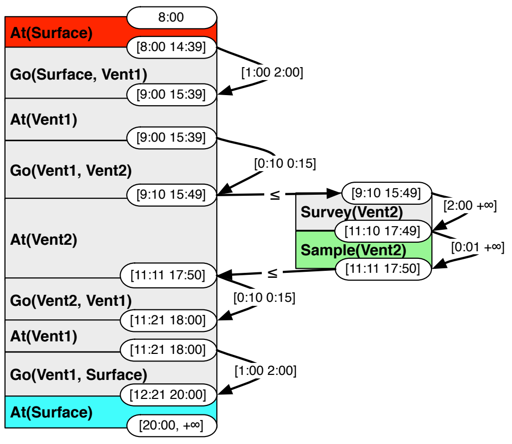


Figure 2: The flexible plan solution of our domain in Fig. 1

left to do both objectives and the probot will have to exclude one of the two science goals or fail to be at the surface by 20:00. While better, the *proactive* approach is still not optimal. Indedd in that case the vehicle will skim through al the action of the plan and be back at the surface by 12:21. If we still consider that scientists still provide this new goal by 14:00 it means that vehicle will then have to dive back underwater in order to survey and sample *Vent 1*. Eben though the plan didf initial provide a solution optimal in term of makespan the resulting mission scenario is not anymore as the vehicle went back and forth during the mission. Moreover in this specific case the Vents are $1500m$ deep which result on 2 actions of at least 1 hour each that could have been avoided should the new goal have been received while the AUV was still near the ocean floor.

None of these 2 approaches are satisfactory; *defer* actions result on the plan execution not being robust to the addition of new goals while blind *proactive* execution can result on redundant actions within the mission that negatively impact the ability of the robot to do as much as it could. A more balanced execution policy would have been for the executive to alter betwen the two policies during the mission.

Looking back at our initial objective we can note that these are different by their source and nature; while sampling *vent 1* was reflecting a scientist need which served the puropose of the mission, the goal to be back at the surface was more related to the operators requirements in order to have regular operation hours and not risk to lose the vehicle before its battery ran out. While fullfilling these two goals is equallly important for the mission success they do not have the same notion of urgency. While it is desirable for the AUV to go sample the *vent* as early as possible, there is no strong motivation to hurry and go back to the surface before we are gettong close to the end of the day. Therefore a better policy for the executive is to alternativelly switch between the two policies depending on what objective(s) they contribute to. By doing so the resulting mission would have been for the vehicle to be *proactive* until it starts to *Survey* the *Vent 2* and then *defer* to go back at the surface – meanning that it would continue to *Survey* the same place – until it either receive the new goal to sample *Vent 2* or it is really time to go back to the *Surface* (around 18:00). Therefore when the AUV receive the new objective to sample *Vent 1* it would be still underwater and will just need to alter its plan to go in this new location. The resulting mission scenratio would then be more efficient both in term of its makespan and timing. Our approach revolve atround this idea to have mission objectives marked as either *urgent* and then explore the plan structure during the plan execution to identify if the action is linked to an *urgent* goal – in which case it need to be executed *proactively* – or not leading on a *deferred* execution policy.

Previous Approaches to Dispatching

Dealing with plan execution is not a new problem and we can find a lot of work that relates to this over several decades. Still, it is pretty rare to see work that envisage that actions could be postponed except when this is necessary to not break the current plan.

The most prominent work is related to the dispatchability of simple temporal networks (STN) (Muscettola, Morris, and Tsamardinos 1998). The core of the problem is to ensure that the temporal constraints can be propagated efficiently within the plan in order to allow the executive to decide quickly whether an action should be started or not while ensuring the plan consistency. In order to accomplish such a task, the STN supporting the plan to be dispatched is transformed into a All-Pairs network and stripped of unnecessary edges, often resulting in a more compact temporal network that lessens the propagation cost of updates. The role of the executive is to select time-points within the current execution bounds and propagate its value within the simplified network. Still, while this work contribute to ensure that execution time are correctly propagated within the plan with a limited cost, it does not directly address how to decide what value should be set for a given time-point in the scope of its possible values. More specifically it is still the role of the executive to decide whether it should start an action as early as possible or consider it as not urgent.

When dealing with least-commitment planning solution this decision is deferred to the plan executive. For example in (Muscettola et al. 1998b), the executive is defined as having two responsibilities: the selection and scheduling of plan events for execution. The executive needs to be highly reactive as it is necessary to function in a real-time environment. One solution offered for dispatching events efficiently is the proactive approach. This approach greatly reduces the plan flexibility, and therefore robustness, as all start time-points are grounded to a specific value which is compatible with the initial constraints.

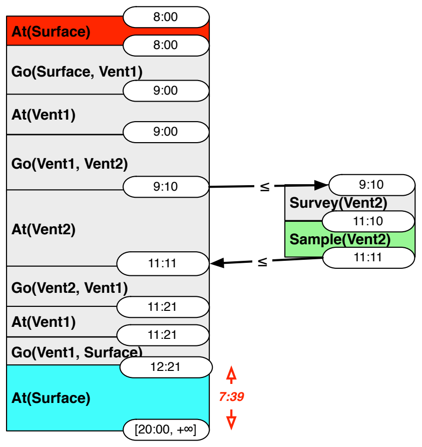


Figure 3: Proactive solution for the plan from Fig. 2 resulting on the end of mission Surface lasting more than 7 hours in the initial plan

While in the common case this might be acceptable it may become problematic when put in he context of potential new objectives emerging during the mission. Take our AUV example, and apply the proactive approach globally — assuming that all actions can be completed on their minimum time. As shown in Fig. 3, the proactive approach allows the AUV to *Sample Vent2* before noon but as continue through the plan it results in the AUV getting back to the *Surface* by 12:21 and being stuck in the context of the current plan for the next 7 hours and 39 minutes. Should the scientist want to *Sample Vent1* the AUV will then be forced to re-plan accordingly and go back and forth between the surface and the locations all other again. By being blindly proactive, the AUV made its overall strategy less efficient than if it took the option to procrastinate at *Vent2* until it needs to get back to the surface.

One case we have seen in literature where a time-point is considered to be deferred is related to the dynamic controllability issue with STNs with uncertainty (STNU). In (Morris, Muscettola, and Vidal 2001), they propose an algorithm that during planning insert wait constraints within the solution temporal network that will allow one time-point to wait until an uncontrollable event occurs. In this case, the deterrence of the execution of this time-point is enforced in order to ensure robust plan execution despite exogenous uncontrollable events. In (Gallien and Ingrand 2006), this approach is further discussed along with the Makespan issue while dealing with least-commitment planners which can insert unjustified waits within the plan potentially decreasing in turn the overall performance of the system. Both of these approaches show that when dealing with uncontrollable temporal constraints – such as for example the duration of a navigation task which depends on external factors – it is necessary to defer some actions in order to ensure the plan execution.

An alternative approach is related to work with soft constraints within the temporal domain indicating preferences on actions timing such as in (Khatib et al. 2001). This should provide a general solution for users to specify whether actions of the plan should be preferably started. One problem with general frameworks dealing with soft constraints though is their poor performances (Bartak 2002) as the general problem is NP-hard. Even though it can be reduced to a more tractable (polynomial) solution by specifying preferences as convex functions (Rossi et al. 2006), the complexity is still cubic in relation to the number of variables. Moreover the solutions proposed do optimize the plan *a-priori* reducing then the plan flexibility which in turn makes it more prone to failure during execution.

Our work is complementary to much of the previous work. While we do not explicitly deal with dynamic controllability, our approach is not inconsistent with that work. In any case, our focus is not on the observation of events during execution but rather on the possibility of new objectives arising at any time, and we want to avoid doing actions that are not "urgent" in order to preserve the context for those objectives. Similarly, we have not implemented temporal preference specification during planning, which has the effect of limiting execution-time flexibility. Our intent is to defer temporal decisions to the executive allowing it to adapt the plan to new objectives without heavily relying on plan-repair or re-planning solutions that could badly impact system reactiveness.

Planning Definitions

To consider the overall planning and plan execution within an agent we use the definition of a temporal domain, planning problem and solution as provided by (Nau, Ghallab, and Traverso 2004):

Definition 1 *A temporal planning domain is a triple $D = (\Lambda_\Phi, O, X)$, where:*

- *Λ_Φ is the set of all temporal databases that can be defined with the constraints and the constant, variable, and relation symbols in our representation.*
- *O is a set of temporal planning operators.*
- *X s a set of domain axioms.*

Definition 2 *A temporal planning problem in D is a tuple $P = (D, \Phi_0, \Phi_g)$, where:*

- *$\Phi_0 = (F, C)$ is a database in Λ_Φ that satisfies the axioms of X. Φ_0 represents an initial scenario that describes not only the initial state of the domain but also the evolution predicted to take place independently of the actions to be planned.*
- *$\Phi_g = (G, C_g)$ is a database that represents the goals of the problem as a set G of tokens together with a set C_g of objects and temporal constraints on variables of G.*

Definition 3 *A plan is a set $\pi = \{a_1, ..., a_k\}$ of actions, each being a partial instance of some operator in O. We define λ as the state-transition function.*

π is a solution of a problem $P = (D, \Phi_0, \Phi_g)$ iff there is a database in $\lambda(\Phi_0, \pi)$ that entails Φ_g.

In this work, our focus is on executing a given plan π which was computed by the agent planner. However, in order to reflect the dynamic interaction of the agent with its environment we need to refine the definition of the sets Φ_0 and Φ_g.

Indeed as the world evolves new observations (or refinement of existing ones) are added into Φ_0. Similarly, the agent operator can request new future goals to be added to the agent Φ_g as mission time advance. We note $\Phi_0(t)$ and $\Phi_g(t)$ the value of these sets at the timepoint t. For the sake of simplicity we consider that alteration of these sets is purely additive with time:

$$\forall\{t, t'\} : t \leq t' \Rightarrow \Phi_0(t) \subseteq \Phi_0(t') \wedge \Phi_g(t) \subseteq \Phi_g(t')$$

The dynamically growing nature of Φ_0 reflects the cumulative observation as the agent execute its plan π. In nominal situation new elements of Φ_0 are refinements the plan – for example by asserting that a planned command just started[1]. We also consider that the agent can receive at any point new objectives that will be added to Φ_g. This assumption have an impact on how it is preferable to handle plan execution. Indeed while deciding when to start an action within the plan, one need to make sure that the execution of this action will not limit the ability fo the agent to treat potential future emerging goals. In the light of it the agent should at the best of its knowledge try to balance the impact of the next available action as early as possible or prefer to delay it in the eventuality new goal occur. In our example, it was making sense to go to Vent2 early, but on the other hand going back to the surface too soon would result on the current plan locking the AUV – within its current plan – at Surface until 8 pm. The solution providing the most freedom for the AUV was therefore for it to alternate between the two policies depending on the action impact.

In order to help the AUV have a better knowledge on the nature of the goals we do consider that each goal provide information on its priority. In that purpose, we define that Φ_g is partitioned into 2 sets:

- the internal goals Φ_{gi} which represent goals the agent *need* to maintain internally. These goals will be considered as objectives that are not of the higher priority and therefore their actions can be deferred during execution.
- the external goals Φ_{ge} which represents the goal received by the agent externally. As these goals are requested by the user, we consider them as to be of higher important – ie the agent *wants* to execute them. Therefore, their execution should be preferrably proactive.

At any point we need to evaluate an action within our plan π we consider that this plan is up to date and

[1]This preclude situations where new observations invalidate the plan which is out of the scope of this paper

provide a solution of all the goals of both Φ_{gi} and Φ_{ge} that can reasonably be done within the current mission scope.

Proposed algorithms

As a new action can be dispatched for execution, the executive needs to evaluate how it relates to the goals of the plan. Intuitively if this action was generated by (or contribute to) an internal goal of Φ_{gi} it needs to be taken proactively, while otherwise we can consider it as non-urgent. Therefore, when evaluating if the token representing this action within the plan the executive needs to do a forward search on the causal links related to this token to see if they lead to an internal goal as implemented in Algorithm 1

Algorithm 1 The function *DispatchToken* finds if there is a goal in Φ_{ge} that is connected to the token, t, and, if so, dispatches the token.

```
function DispatchToken(Token $T$)
    $BooleanGoal = SearchForGoal(T)$
    if $Goal$ = True then
        return Dispatch $T$
    else if $T$ start upper bound $\leq$ upper bound for the current tick then
        return Dispatch $T$
    else
        return Don't dispatch $T$
    end if
end function
```

Algorithm 2 The function $SearchForGoal$ does a Forward search looking for a token that is in the set Φ_{ge}.

```
function SearchForGoal( Token $T$ )
    List $Search$ = Action(s) that $T$ is a Condition
    for all Action(s), $A$, in $Search$ do
        for all Effect(s), $E$, of Action $A$ do
            if $E$ is a Goal in $\Phi_{ge}$ then
                return True
            else if $E$ is a Condition of an Action $A_c$ then
                Add $A_c$ to back of list $Search$
            end if
        end for
    end for
    return False
end function
```

This Algorithm is the central deciding point for how a token should be dispatched. By finding out that a token is connected to a goal in Φ_{ge}, we conclude that the token is a sub-goal, and thus dispatch it immediately being proactive. On the other hand, if the token is not connected to a goal then we defer dispatching it until necessary. This demonstrates our distinction between how we dispatch tokens, proactive or deferred.

A crucial part for deciding how to dispatch a token is finding whether the token in question is connected to a goal in Φ_{ge}, call it G. Algorithm 2 does the function of searching for G in a forward-search manner. We define a causal link within our plan that must be met in order for our search to function.

Definition 4 *A causal link is defined as linking a goal as an effect of an action whose conditions are needed in order to complete the goal and, thus, are subgoals. This link can be recursive as the conditions themselves may be the effect of an action causing a causal chain to build.*

During our search, if we find G then we know that the original token is part of the solution for completing G. As such, we want to be proactive with completing the token early so as to ultimately complete G. If we don't find G then the token has no connection to an external request. The token still needs to be dispatched, however, there is no one explicitly requesting it to be accomplished. Thus, there is no reason to be proactive.

This algorithm is equivalent to a breadth first search along the planning structure starting from the action we need to evaluate. Therefore, its complexity is $O(N+E)$ where N is the number of tokens within the plan and E the number ofcausal links that relate these tokens.

Dynamic solution during planning

Searching for a goal as Algorithm 2 does can be quite computational expensive particularly if there are many tokens that are continuously being dispatched. Completing a full search every time a token needs to be dispatched can severely slow down the execution process, which needs to remain quick to ensure proper execution. Therefore, our next algorithmic approach distributes the full search within the creation of the plan. Resulting in spreading out the full cost of the search. In order to not repeatedly search the plan, we save the tokens that are connected to a goal in Φ_{ge} found during the search. In this way, we acquire a list of tokens, $List_{goals}$, that should be dispatched early.

An alternative solution is to embed the propagation of these value during the planning search. The algorithm uses the same dispatching method as algorithm 1. The difference is that rather than searching for the goal using algorithm 2, it only searches the list, $List_{goals}$, to see if the token is in it. The actually searching for the goals and causally connected tokens happens in algorithms 5, 7.

Algorithm 3 Saves goals as they are added to plan

```
function NotifyAdded( Token $T$ )
    if $T$ is a Goal in $\Phi_{ge}$ then
        Insert $T$ into $List_{goals}$
    end if
end function
```

Algorithm 4 Removes the token after it is removed from the plan

```
function NOTIFYREMOVED( Token $T$ )
    Remove $T$ from $List_{goals}$
end function
```

Algorithm 5 Searches for tokens connected to goals

```
function NOTIFYACTIVATED( Token $T$ )
    if $T$ is a goal in $\Phi_{ge}$ or $T$ is linked to a goal through one causal link then
        Recursively search the reverse causal link and add the tokens into $List_{goals}$
    end if
end function
```

Algorithm 6 Deactivates token in plan

```
function NOTIFYDEACTIVATED( Token $T$ )
    if $T$ is not a goal and not one causally linked to a goal then
        Remove $T$ from $List_{goals}$
    end if
end function
```

Algorithm 7 Searches plan when tokens are merged

```
function NOTIFYMERGED( Token $T$ )
    if $T$ is goal in $List_{goals}$ then
        Recursively search the reverse causal link of the active token merged with $T$ and add tokens to $List_{goals}$
    else if The active token of $T$ is in $List_{goals}$ then
        Recursively search the reverse causal link of $T$ and add tokens to $List_{goals}$
    end if
end function
```

Algorithm 8 Removes token when split

```
function NOTIFYSPLIT( Token $T$ )
    if $T$ is not a goal or not one causally linked to a goal then
        Remove $T$ from $List_{goals}$
    end if
end function
```

In order to distribute the search, we situate our algorithm within the planning search of the Europa Planner,(Frank and Jónsson 2003), which offers callback functions for when a token in the plan is altered. The majority of the searching happens when tokens are either activated or merged. For a token to merge with another token it has to be compatible with another token already in the plan. Splitting happens when they are no longer compatible. We have designed our algorithm around the Europa Planner, however, we believe that the general approach can work on any other planner.

Taking full advantage of the planning search, we use a backwards search from the goal following the reverse causal link to the connected tokens. We fully search from the goals because we know that all the tokens connected through the causal link are sub-goals. By contrast, fully searching each token could be wasteful because there is no certainty that it will be linked to a goal and, therefore, could bring little value to our search. However, some tokens may get added to the plan or linked to a goal after we have already searched the goals. Therefore, for every token we do a local forward search of one causal link to verify if it is connected to a goal in our saved list. If so, we do a full backwards search from the token since it has now proven to be valuable. After the plan has been searched, it is as easy as searching a list for a token to see if it should be dispatched early or be deferred to later.

This approach is potentially more costly than the previous algorithm as it needs to do local updates whenever the plan is altered by the search including retracting past updates if a backtrack occur during the search. Still it is compelling in the fact that this cost occurs during planning reducing the decision problem during execution to simply check if the given action has been marked during planning. It has been the solution we have preferred within our system for this reason as it is functionally equivalent to previous algorithm while reducing extra computation cost as the plan is executed. Planning phases occur within the plan only when either the initial plan failed to execute or the set of goals has been altered. Therefore, it is safe to assume that planning should occur more sporadically than execution decisions which then give an edge to this latter algorithmic solution.

Experimental results

Need to present here both practical results that illustrates the outcome of our solution and how it benefits … eg show a case where new goals are introduced as we go along

Also need some analysis – potentially numerical – on the overhead and impact fo both solutions relative to each other but also potentially to a more direct classic approach. To finally discuss why one solution was preferred on our system

Our experiment follows that of the original plan given in Fig. 2. At the beginning, the AUV need to *Sample Vent2* and it has to return to the surface by the end of the mission. We implemented this on our executive with plans being produced by the Europa planning engine (Frank and Jónsson 2003).. Figure 4 demonstrates the resulting plan from our algorithm and will guide our explanation. Both algorithms will result in the same execution of the plan but their approach is quite different.

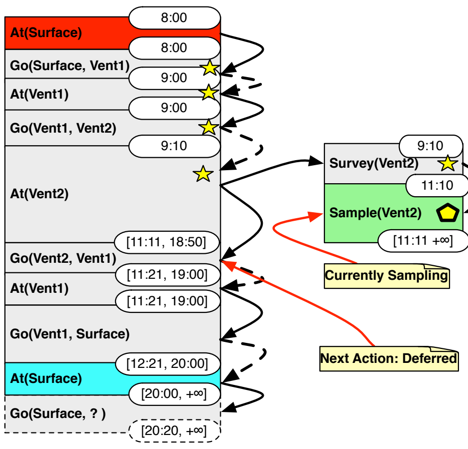


Figure 4: Our algorithm solution for the plan from Fig. 2. Solid lines indicates conditions and dashed lines indicates effects. Pentagons indicate *urgent* goals, stars indicate tokens that were deduced as *proactive* by our algorithm.

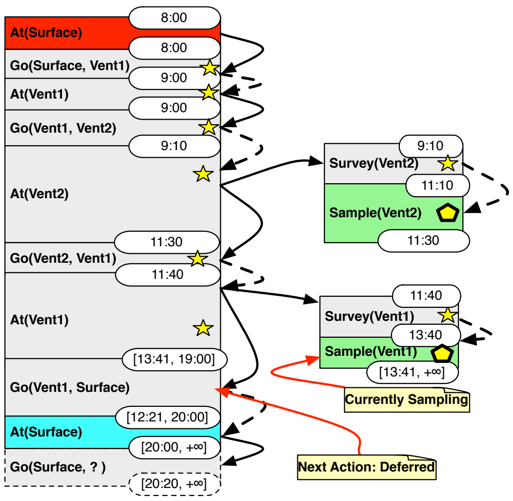


Figure 5: Our algorithm solution after receiving external request for the plan from Fig. 4

For Algorithm 1, the search is quite straight forward. For example in Fig. 4, if we are *At Surface* then the next *Go* token will be dispatched. Because the search will follow the causal links forward, where upon, it will find the goal *Sample Vent2* resulting in dispatching proactively. Similarly, this happens for all of the tokens that are starred. The next token to *Go* to *Vent1* is continuously searched but deferred for later dispatching, because it is not connected to a goal. The resulting AUV stays at *Vent2* rather than heading to the *Surface* immediately like in Fig. 3.

For the distributed algorithm approach, each token is checked during the creation of the plan to see if it is an external goal in Φ_{ge}, or connected to one through a causal link. When the *Sample Vent2* is checked, we immediately find that it is a goal. We then follow the reverse causal link and find *Survey Vent2*. However to better illustrate the algorithm, we can imagine that only the *Survey Vent2* has been causally connected to the goal so far. Therefore, we only star those two tokens. The path from *Surface* to *Vent1* to *Vent2* has yet to be built. When the path has finally been built, and *At Vent2* is checked, our algorithm searches one causal link and finds *Survey Vent2* which is starred. The search then follows the reverse causal link and stars the rest of the path. The starred tokens will then be proactively dispatched while the non-starred tokens will be deferred until later. Having similar results to Algorithm 1.

Our reasoning for keeping the AUV at *Vent2* is that nothing is requiring it to go *Surface* as soon as possible and that more external requests may come in the near future. To demonstrate this imagine that while the AUV is at *Vent2* it gets a request at 11:30 asking it to *Sample Vent1*. We show the resulting plan in Fig. 5. Again, both algorithms will result in the same conclusion. Algorithm 1 will search from *Go(Vent2, Vent1)* and will now find *Sample Vent1*. The distributed algorithm will find and star the new tokens when the plan gets updated. The resulting new starred tokens will be proactively dispatched.

As the end of the day approaches, the AUV will need to start heading to the surface. At 19:00 both algorithms will find that the *Go(Vent1, Surface)* is still not connected to a goal but that the upper bound time for starting the token has been reached. Therefore, we will dispatch the token because it has become necessary for completing the plan. After returning to the *Surface*, the plan shows that the AUV will then *Go(Surface, ?)* (dashed in the figures). This is an artifact resulting from the plan model which specifies that a *At* token is followed by a *Go*. However, our algorithm will not be dispatching this token as it is not connected to a goal and its upper bound start time ($+\infty$) will not be met.

Conclusion & future directions

As autonomous agents become more and more flexible it becomes difficult to deal with all the possible changes that occur during the mission. While there has already been work around the topic of robust plan execution, they always have been focused on threats coming from execution feedback while not considering how the potential emergence of new goals could impact how the current plan should be executed. In this paper, we discussed this aspect and proposed a solution that relies on extra information on the urgency of an objective. By propagating this through the plan structure, we were able to leverage this information for supporting the plan executive decision starting actions proactively or not. A second algorithm allowed to propagate this information during the planning phase ensuring that the executive ca ntify this information quickly.

As we stated in the related works our approach does not really address dynamic controllability and has the more classic assumption present in many planning frameworks that time-points are controllable. A side effect of this is that in its current state it may result on the system to decide to defer action as late as possible. In our example, this would result on the AUV leaving Vent1 as late as 19:00 making the rest of its plan brittle to any delay due for example to downward water currents on its way to the surface. This needs to be further addressed in the future and, especially, how our work can be integrated with work presented in (Morris, Muscettola, and Vidal 2001).

Further as of today, we consider that the qualification of the goal is predefined when the goal is submitted by the planner. It is possible though that part of this can be refined on some cases based on the nature of the goal. Looking back at our domain, one can note that the 2 goals provided are constrained differently on their start time; while *Sample Vent2* start time is limited only on its upper bound, the returning to surface conversely is constrained only on the lower bound of its start time. This difference hints on some of the issues we presented. While we do consider that explicit information of these goals help the plan execution to be improved when such information is not initially present. We also are aware that the nature of the constraints within the goal itself can help identify the best policy to be done. It is obvious that for Sampling Vent2 it is better to be proactive on the actions that contribute to this goal. Conversely, the other goal only matter if it appears fairly late in the plan which means that it is probably better to not start completing this part of the plan too aggressively. We plan to further explore how we can refine the distinction between the different policies by using the information provided by the constraints of the different objectives.

References

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

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

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

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

[Frank and Jónsson 2003] Frank, J., and Jónsson, A. 2003. Constraint-based Attribute and Interval Planning. *Constraints* 8(4):339–364.

[Gallien and Ingrand 2006] 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 and Veloso 1998] Haigh, K., and Veloso, M. 1998. Interleaving Planning and Robot Execution for Asynchronous User Requests. *Autonomous Robots* 5:79–95.

[Khatib et al. 2001] 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 and Ingrand 2004] Lemai-Chenevier, S., and Ingrand, F. 2004. Interleaving Temporal Planning and Execution in Robotics Domains. In *Assoc. for the Advancement of Artificial Intelligence, National Conference (AAAI).*

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

[Morris, Muscettola, and Vidal 2001] 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 et al. 1998a] Muscettola, N.; Nayak, P.; Pell, B.; and Williams, B. 1998a. Remote Agent: To Boldly Go Where No AI System Has Gone Before. *AI Journal* 103:5–48.

[Muscettola et al. 1998b] Muscettola, N.; Morris, P.; Pell, B.; and Smith, B. 1998b. Issues in temporal reasoning for autonomous control systems. In *Proceedings of the second international conference on Autonomous agents*, AGENTS '98, 362–368. New York, NY, USA: ACM.

[Muscettola, Morris, and Tsamardinos 1998] 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, Ghallab, and Traverso 2004] Nau, D.; Ghallab, M.; and Traverso, P. 2004. *Automated Planning: Theory & Practice.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

[Py, Rajan, and McGann 2010] 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).*

[Rossi et al. 2006] Rossi, F.; Sperduti, A.; Venable, K.; Khatib, L.; Morris, P.; and Morris, R. 2006. Learning and solving soft temporal constraints: An experimental study. In *Principles and Practice of Constraint Programming-CP 2002*, 249–264. Springer.