AAAI Rough Draft

Paper

Abstract

Introduction

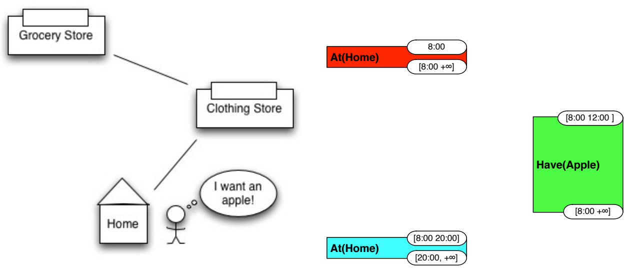
In order for an agent to operate in a dynamic environment, the agent needs to be constantly observing the external world so that it can keep up to date with how the world is changing. Therefore, a robotic agent needs to be planning in situ so that it can alter, or update, its plan according to the changes. The actual execution of the plan will also need to change accordingly. The execution also provide constant feedback on how the world is evolving. Thus as a result of the dynamic environment, the robotic agent will need to anticipate constant changes in the plan and need to evaluate how to properly execute the plan.The ability to plan in-situ leverage the ability to dynamically add new objectives dynamically to the agent during the mission execution instead of being forced to sepcify them beforehand. These capabilitiwes allow to more felxible and autonomous missions but also makes more challenging the overall mission execution by itself.

In the pursuit of planning in a dynamic environment, 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 real-world planning possible. Based on these past successes, there have been multiple experiments that have successfully functioned in the real-world, the Remote Agent Experiment `RAX` (Muscettola et al. 1998a) and the Autonomous Spacecraft Experiment `ASE` (Chien et al. 1999). In particular, we focus on constraint-based temporal planning which has been demonstrated to be a viable solution for real-world planning (citation).

Most of these modern systems are using at the high-level constraint-based planner with rich representation of resources and time such as the ones presented in (Frank and Jónsson 2003; Lemai-Chenevier and Ingrand 2004). These systems provide a rich plan structure which often implements least-commitment planning giving a solution that do not commit on specfic values (such as the start and end time ofn action) unless it is required by the model. This flexibility also puts the burden to plan executive to decide a which time a specific action within all its possible value. Resolving this problem resulted on a large amount of research both considering how to represent the plan to be dispatched to the executive with a compact yet complete representation, along with more refinements on how to deal with uncontrolable constraints. Still, remains the issue of deciding whether the next possible(s) action(s) should be started immediately or if it should be postponed for a later start date.

To avoid the system to procrastinate the solution selected has often been to start actions as early as possible while ensuring that it does not impact the plan brittleness. This choice is lead by the intuition that by starting actions early it gives more room in the future to deal worst case scenarios of execution. While this solution works in the general case where all the goals of the agent are known a priori, it is challenged when one consider that the agent can have new obsejectives added as the mission execute. Indeed taking actions to early can lead the agent in a situation where the new goal received is not integrated as efficiently as it would have been should the agent have waited before taking action. In this context, it is imporant for the agent make a distinction between actions that can be taken proactively versus other actions that may not be yet urgent. In this paper, we propose a systematic approach to allow the agent to make such distinction for each action by tracing the causal relations with goals within the current plan. This aloow us to deternmine the best startegy based on the nature of the goals this action contribute to.

An Illustrative Example We define a simple Shopping agent that will be used to illustrate different approaches and issues that arise when dispatching. First, the agent can either be at a location or be traveling. Second, the agent starts with a certain amount of energy (resource) that will decrease as it travels. Third, the agent has some actions that allow it to interact with the environment. It can go to a location, buy items at stores, and eat the edible items for energy. Lastly, the



(a) Domain illustration (b) Initial problem

Figure 1: A description of our shopping agent problem with an illustation of the domain (1a) along with the intial partilal plan for this probelem (1b). In this domain our agent is initially at *Home* at 8am, he wants to have an *Apple* before noon and needs to be back at *Home* before 8pm (noted 20:00 here).

agent can want to buy an item. In a typical Shopping agent mission, the goals are to buy a list of items from different stores and be back home by the end of the night.

However, an issue with finding a balanced approach is deciding whether it starts to starts its [?]n as early as possible – which we will call proactive [?]or wait until the action should necessary start – called later deferred . makeThere is a clear difference between the two approaches but when should, for example, the shopping agent wait or start early ? Lets demonstrate an ideal scenario where the shopping agent balances both approaches illustrated in Figure 1.

The agent, waking up at 8 AM, *wants* to buy an apple and *needs* to be home by 8 PM. The agent then leaves as early as possible to start shopping. Considering that that it takes 20 minutes to half an hour to go from Home to the Clothing Shop, 10 to 15 minutes to go from the Clothing shop to Grocery and 5 to 10 minutes in order to buy an apple, a general plan solution is presented in Figure 2. The plan presented here is partially instantiated giving the ganet the freedom to decide when to start an action within the valid boundary of the solution. The agent can then for example go early on buy its apple so he is sure to have it as early as possible. Conversely, the agent may decide to walk around and fo window shopping throughout the rest of the day as there is no hurry yet. By 7pm though, it should eventually leave so it can arrive back home by 8pm. In this scenario, we see that agent did alternate between deciding to execute actions early or proacrstinate depending on the the nature of the action it need to take next or more accurately the nature of the objectives related to this action. Indeed, the agent was proactive on leaving his house as it was related to him *wanting* an apple, on the other hand he just *needed* to get back home by 8 PM allowing him to procrastinate at the grocery store. By doing so he will be around in case his wife is calling him to buy extract things before

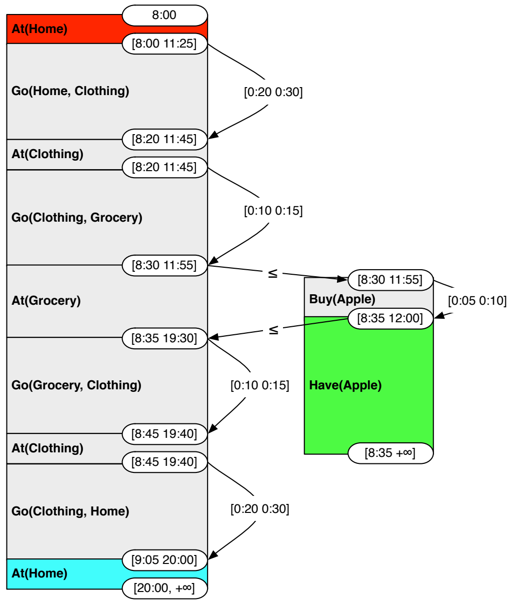


Figure 2: The flexible plan solution of our domain in Fig. 1

he gets back home.

While this example may appear academic at first, it reflects situations er have seen within embedded agent execution in our domain. Indeed, we do daily operations wehere our AUV is deployed and scientists can remotely send new objectives as the mission goes along to the vehicle as they see new area of interest. At the same time the vehicle has also operational objectives such as going to a place wheer its recovery will be easier for operators. This gives a similar distinction between the science objectives and operation objectives. Similarly to our shopping agent we do not realy want the AUV to get back to recovery area too early as a new science goal could be sent to him which in turns would rather be fulfilled as early as possible.

This paper discusses the problem of dispatching when trying to execute a plan. In particular, dispatching in a dynamic environment where the plan is known to change either by uncontrollable events or external requests with new directives. External requests can occur at any time which make them in essence uncontrollable events. Specifically, we focus on how these new requests, coming from the external world, will affect the way we dispatch the plan, rather than how they will be integrated into the plan or any part of the planning process. The reason for our separation from planning is that oftentimes planning and executing are split up into two different jobs. Where a robotic agent is give an already created plan, and it must then choose how to execute that plan. Therefore, our focus is on how to dispatch a plan after it has already been created, while understanding that the plan may still change in the near future.

The approach we have taken on dispatching looks at the token level of a plan, specifically at the externally requested tokens which we define as goals. Because they are requested by an external person with the intent of being completed, they have a high priority. In contrast, there are tokens that only describe the evolution of a timeline, which we define as non-goals. In order to keep the plan valid, the agent is obligated to complete the non-goals, but there is no rush. Thus, the non-goals have a low priority. Therefore, we want to complete the goals as early as possible in order to give adequate time for the possibility of new goals, and complete the non-goals as they become necessary for the validity of the plan. Some may argue that finishing the goals early doesn't guarantee that there will be enough time for new goals, however, that is an issue with planning, and our concern is with dispatching.

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 during the last decades. Still, it is pretty rare to see work that envisage that actions could be postponned 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, resulting often on a more compact temporal network and lessen the propagation cost of its updates. The role of the executive is to select timepoints 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 timepoint 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 timepoints are grounded to a specific value which is compatible with the initial constraints. In order to avoid having a

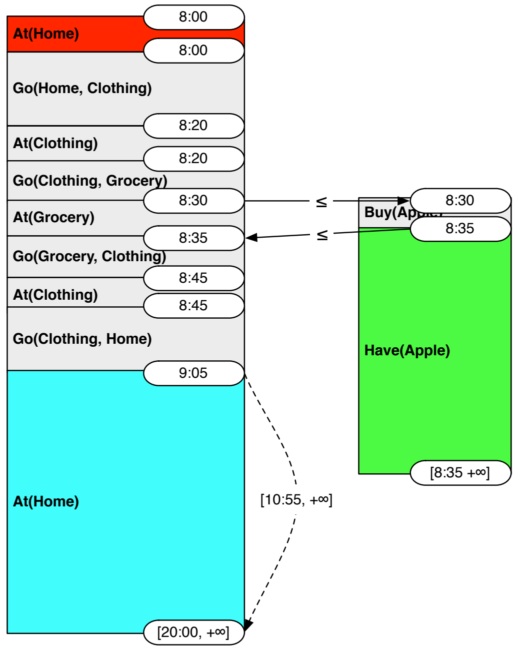


Figure 3: proactive solution for the plan from Fig. 2

procrastinating system the overall agreement in term of po[illegible] is to start an action as early as possible (ref ?). There have been many uses of these two approaches when dispatching plans, as previously shown, but a compromise to one is most often used rather than a balance. Demonstrated in the tool MAPGEN (Bresina et al. 2003) which used the Earliest Time Solution (ETS) for generating and displaying plans very quickly. Then allowing the users to manipulate the plan afterwards, getting around the problem of ETS generating undesirable plans. What is needed is a way of using both proactive and deferred together giving a balanced approach to dispatching.

While in the general case it can be acceptable it may become problematic when put in he context of potential new objectives emerging during the mission. Take our shopping 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 agent to get its apple safely before noon but as we go along with the remaining part of the plan it results in the agent getting back home by 9:05 and being stuck in the context of the current plan for the next 10 hours and 55 minutes. Should he receive a call from his wife to buy extra things he will then be forced to re-plan accordingly and get back and forth between his home and the stores all other again. By being blindly proactive the agent made its overall strategy less efficient than if he took the option to procrastinate at the mall until he needs to get back home.

The only case we have seen in literature where a timepoint 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.

Our work on the other hand is more complementary to this one, indeed while we do not really deal with dynamic controllability in our problem – which can anyway be treated as in this previous work – the core of our problem is not that much the observation of events during execution (which is more a observation related impact) but instead we are more focused on the knowledge that, within our agent, new objectives can emerge at any time, and we want to avoid as much as possible doing actions which are not "urgent". By doing so, we take the risk of doing unnecessary actions as new goals need to be integrated in the plan.

Contribution

However, an issue with finding a balanced approach is deciding when to use least-committed and when to us EST. There is a clear difference between the two approaches but when should, for example, the shopping agent wait or start early. Lets demonstrate an ideal scenario where the shopping agent balances both approaches. The agent, waking up at 8 AM, wants to buy some pants and t-shirts and needs to be home by 8 PM. The agent then leaves as early as possible to start shopping. The agent continues to walk around and shop throughout the day because there is no reason to rush home yet. Around 7 PM, the agent decides that its time to go home so that it can make it there by 8 PM. What we have demonstrated is a clear example of how the shopping agent can start early while procrastinate about going home. This gives the agent some flexibility to do more shopping in the day, if needed, while still accomplishing its goal of being home by 8 PM.

Now lets look at how to create this ideal agent. There is clearly a distinction being made between two different types of goals. There are the wants of the agent, which is buying stuff, and the requirements like being home by 8 PM. The agent starts as early as possible when it wants something, using EST, and waits until it has to for the requirements, using least-committed. This distinction is a simple but effective way of balancing the two approaches. In planning, the goals can have subgoals or conditions that must be met before the goal can be completed. Lets say the Shopping agent wants an apple then a sub-goal would be for the agent to buy an apple and even further to go to a store that sells apples. All these sub-goals are linked to the original goal and thus should be considered as goals themselves. Depending on the original goal, the new goal will either follow the ETS or least-committed approach.

However, we need to define a relation between the goals and their sub-goals, which will allow us to easily identify all the sub-goals. As stated earlier, the subgoals are conditions that must be met before the original goal can be achieved. The conditions are not linked directly to the original goal, the sub-goals are actually the conditions to an action that will satisfy the original goal.

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 tques 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.

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 defintion of the sets Φ_0 and Φ_g.

Indeed as the world evolve new observations (or refinement of existing ones) are added into Φ_0. Simlarly, 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 time-point t. For the sake of simcplicity 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 cumul of observation as the agent execute its plan π. In nominal situation new elelments 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 nerw objectives that will be added to Φ_g. This assumption have an impact onhow it is preferreeable to handle plan execution. Indeed while deciding when to start an action within the plan, one need to amae sure that the execution of this action will not limit the ability fo the agent to treat potential future emerging goals. In the ligh of it the agent should at the best of its knowledeg 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 the Grocery early, but on the other hand going back home to soon would result on the current plan locking the agent – within its current plan – at home unitl 8 pm. The solution providing the most fredom for the agent was therefore for him to alternate between the two policies depending on the action impact.

In order to help the agent have a better knowledge on the nature of the goals we do consider that each goal provide information on its priority. In that putspose 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 providea soltuiona of all the goals of both Φ_{gi} and Φ_{ge} that can reasonably be done within the current mission scope.

Algorithm Draft

As a new action can be dispatched for execution, the executive needs to evaluate how it relates to the goals of the plan. Intuitevely 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

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

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 ≤ upper bound for the current tick then
        return Dispatch T
    else
        return Don't dispatch T
    end if
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.

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

```
function SearchForGoal( Token T )
    for all Actions(s), A, that T is a Condition do
        for all Effect(s), E, of Action A do
            if E is a Goal in Φge then
                return True
            else if E is a Condition of an Action then
                return SearchForGoal( E )
            else
                return False
            end if
        end for
    end for
end function
```

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 meet 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, no reason to be proactive.

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 dispatch 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.

The algorithm uses the same dispatching method as algorithm 1. The big 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 3, 4, 5.

Algorithm 3 Saves goals as they are added to plan

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

Algorithm 4 Removes token from list

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 Φ_{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

In order to distribute the search, we situate our algorithm within the planning search which offers callback functions for when a token in the plan is altered. 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-goal. On the contrary, 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.

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 agent wants to *Buy* an apple and it has to be *Home* by the end of the mission. We implemented this on our executive with plans being produced by europa planning engine (Frank and Jónsson 2003). Figure 4 demonstrate 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.

For Algorithm 1, the search is quite straight forward. For example in Fig. 4, if we are *At Home* then the next *Go* token will be dispatched. Because the search will follow the causal links forward, where upon, it will find the goal *Have Apple* resulting in dispatching proactively. Similarly, this happens for all of the tokens that are starred. The rest of the tokens are deferred for later dispatching. The resulting agent stays at the *Grocery* rather than heading *Home* 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 *Have Apple* is checked, we immediately find that it is a goal. We then follow the reverse causal link and find *Buy Apple*. However to better illustrate the algorithm, we can imagine that only the *Buy Apple* has been causally connected to the goal so far. Therefore, we only star those two tokens. The path from *Home* to *Clothing* to *Grocery* has yet to be built. When the path has finally been built and *At Grocery* is checked our algorithm searches one causal link and

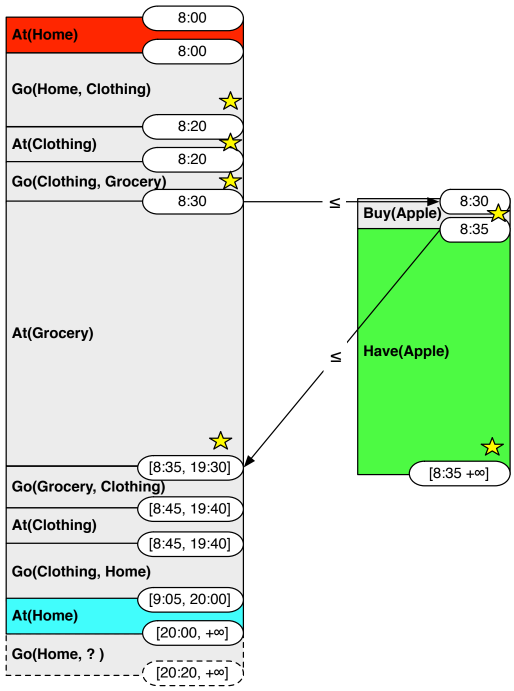


Figure 4: Our algorithm solution for the plan from Fig. 2. Stars indicate tokens that were marked as "proactive".

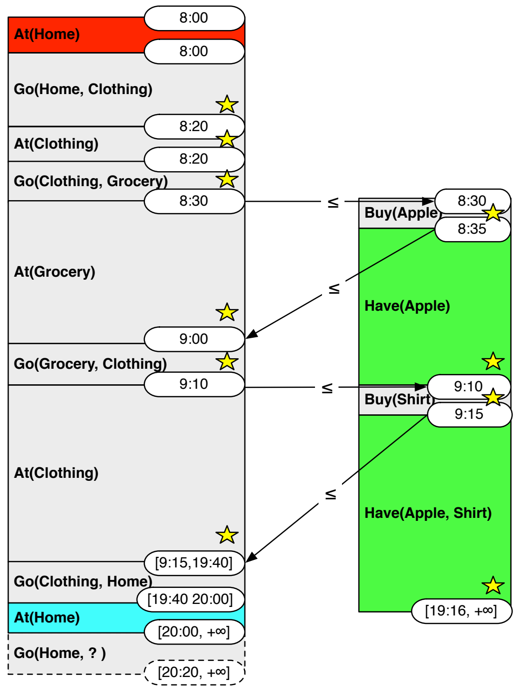


Figure 5: Our algorithm solution after receiving external request for the plan from Fig. 4

finds *Buy Apple* 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 agent at the *Grocery* is that nothing is requiring it to go *Home* as soon as possible and that more external requests may come in the near future. To demonstrate this imagine that while the agent is at the *Grocery* it gets a call at 9:00 asking it to buy a shirt. We show the resulting plan in Fig. 5. Again, both algorithms will result in the same conclusion. Algorithm 1 will search from *Go(Grocery, Clothing)* and will now find *Have Shirt*. 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 agent will need to start heading home. At 19:40 both algorithms will find that the *Go(Clothing, Home)* 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 getting *Home*, the plan shows that the agent will then *Go(Home, ?)* (dashed in the figures). This is an artefact resulting from the plan model which specfies that a *At* token is followed by a *Go*. However, our algorithm will not be dispatch this token as it is not connected to a goal and it's upper bound start time ($+\infty$) will not be met.

Conclusion & future directions

N[illegible]argarph that resume what was presented

As we stated in the realted works our approach do not really adrress dynamic controllability and hacve the more classic assumption present in many planning frameworks that timepoints are controllable. A side effect of this is that in its current state it may result on the system to decide to dferr action as late as possible. In our example, this would result on the agent leaving the Grocery as late as 19:30 making the rest of its plan brittle to any delay due for example to traffic jam on its way back. This need ot 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 goals is submitted by the planner. It is possible though that part of this can be refined on some case based on the nature of the goal. Looking back at our domain, one can note that the 2 goals provided rae constrained differently on their start time; while "have an apple" start time is limited only on its upper bound, the being back at home converesely is constrained only on the lower bound of its start time. This difference hints on some of the issues we presented. Whiel we do consider that explicit information of tese goals help the plan execution to be improved when such infromation is not initialy 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 obviouds that has have an apple have a deadline 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 mena that it is probably better to not atrt to complete this [part of the plan too agressively. 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).*

[Bresina et al. 2003] Bresina, J.; Jonsson, A.; Morris, P.; and Rajan, K. 2003. Constraint Maintenance with Preferences and Underlying Flexible Solution. In *Proceedings of Online Constraint Solving: Handling Change and Uncertainty CP2003 workshop.*

[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.

[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).*

[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.