Temporal Milestones in HTNs [*]

Fusun Yaman and **Brett Benyo** and **Alice M. Mulvehill**
Raytheon BBN Technologies, 10 Moulton St, Cambridge, MA 02138
{fusun, bbenyo, amm}@bbn.com

Abstract

We present *temporal milestones* for hierarchical task networks to enable the complex synchronization of tasks. A temporal milestone of a task is an intermediate event that occurs during the execution of a complex task, e.g., the start time, the end time or a milestone of any of its subtasks. Unlike landmark variables, introduced in existing work, temporal milestones respect the task abstraction boundaries and preserve structural properties enabling much more efficient reasoning. Furthermore, temporal milestones are as expressive as landmark variables. We provide analytical and empirical evidence to support these claims.

Introduction

Hierarchical Task Networks (HTNs) are widely used in real world applications and have been extended for temporal reasoning (Tate, Drabble, and Kirby 1994; Wilkins 1997; Laborie and Ghallab 1995; Myers et al. 2002; Castillo et al. 2006). Most of temporal reasoning in HTNs is based on Simple Temporal Networks (STNs) (Dechter, Meiri, and Pearl 1991). The extensions use the start and end times of the tasks for defining temporal constraints between tasks in the same network, i.e., between a parent and its children and/or between siblings. This has desirable computational properties, such as sibling restricted constraint propagation which reduces the computational space and time complexity of consistency checking (Yorke-Smith 2005).

This approach however, can not represent complex temporal synchronizations between tasks that don't have the same parent. For example, consider a space application where two rovers are performing parts of a big experiment at different sites and exchange information occasionally. During these communication periods the rovers need to make sure that they are within communication range. Communication tasks that need to be synchronized are not part of the same task network, i.e., they are subtasks in each rover's mission. Furthermore, they are clearly intermediate events that occur during the execution of each mission.

The only work that addresses this kind of synchronization is based on landmark variables (Castillo et al. 2006). However, their approach crosses the task abstraction boundaries by requiring the user to know the tasks that are further down in the decomposition hierarchy. There is also no trivial way to handle multiple occurrences of the same task. Finally, the STNs that are created using landmark variables violate the structural advantages that let the problem be solved efficiently. This is important because in real world applications the size of the problem can grow rapidly, rendering even low order polynomial algorithms inefficient.

Our solution is *temporal milestone* augmented HTNs. In addition to the start and end time we associate milestone events with tasks that signify certain stages of task execution. Thus a milestone of a task allows selective visibility to the task events further down in the decomposition without the need to know what that lower task is. Temporal milestone augmented HTNs are as expressive as HTNs with landmark variables although STNs generated with temporal milestones are bigger (more nodes and edges) then the ones generated with landmark variables.

Temporal milestones preserve the level of abstraction between tasks as well as the structural properties of the sibling restricted STNs thus providing a representation for complex synchronization while enabling efficient reasoning. To that end we show how to generalize existing algorithms to work with milestone augmented HTNS. We also present a novel algorithm for reasoning with milestones. The algorithm is especially suitable for mixed initiative systems where the deconfliction of user controlled temporal constraints is important. We show how temporal milestones can be used in a military application domain that requires complex synchronization and how we leverage milestones to efficiently deconflict the goal constraints enforced by the users.

Our experimental results on this application domain show that despite the overhead introduced by milestones the reasoning can be done much more efficiently than without milestones. In this specific application where the complexity of each sub HTN dominated the overhead of milestone events, the gain was linear in computational space and quadratic in computational time.

In the rest of this paper we will first introduce the basic concepts of HTNs and STNs and then explain in detail the complex-synchronization problem in the context of the Air

[*]This work is based upon work funded by the Air Force Research Laboratory (AFRL), Contract No. FA8750-10-C-0184.

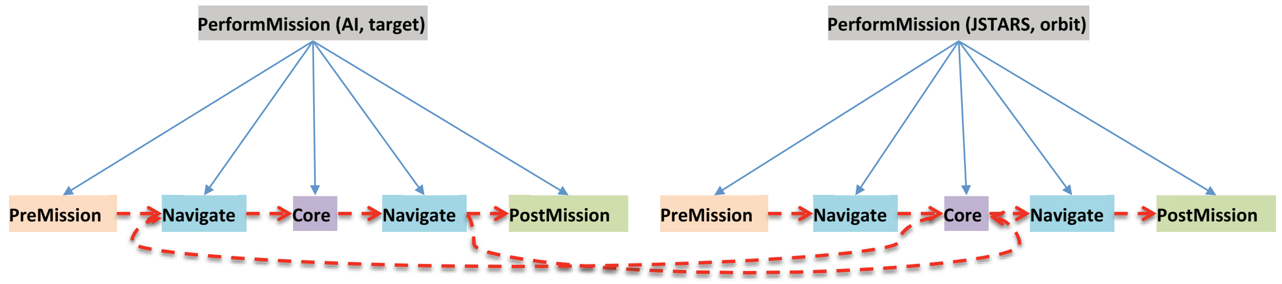


Figure 1: A two mission problem from Air Tasking Order domain. Solid blue arrows show decomposition and red dotted arrows show temporal relationships.

Tasking Order (ATO) domain while pointing out the shortcomings of the existing work. Next we will present our solution, temporal milestones, along with an analysis of how it compares with standard representations. We then present algorithms for reasoning with temporal milestones and conclude with empirical results that demonstrate the benefits of our approach.

Background

In the Hierarchical Task Network (HTN) planning paradigm (Erol, Hendler, and Nau 1994) goals are high level tasks and the planning process simply decomposes these tasks progressively into subtasks (possibly high level tasks themselves) via applicable methods, ultimately yielding a set of simple tasks which can be achieved through operators. HTNs have been extended to support the temporal relationships between the tasks (Tate, Drabble, and Kirby 1994; Wilkins 1997; Laborie and Ghallab 1995; Myers et al. 2002; Castillo et al. 2006; Yaman and Nau 2002). These extensions support constraints that reference the start and end times of tasks in the same network. A majority of these extensions and planning algorithms build on Simple Temporal Networks (STNs) (Dechter, Meiri, and Pearl 1991) to reason with temporal planning domains.

An STN is a directed graph where vertices are time points (events) and every arc represents a single temporal constraint defined on two time points. Quantitative temporal constraints allow users to express numeric constraints such as event A should end at least 15 minutes before event B starts. The basic approach for checking consistency of an STN is to ensure path-consistency (PC) using Floyd Warshall's all pairs shortest-path algorithm (Floyd 1962), which has a complexity of $O(n^3)$ where n is the number of events (nodes) in the STN. After the shortest path computation, if the distance of any node to itself is negative then the set of constraints represented by the STN is inconsistent. For consistent STNs consistency checking also yields the minimal domain for each event. There are also incremental approaches that maintain the path consistency (Cesta and Oddi 1996; Planken 2008; Mohr and Henderson 1986) to efficiently update the STNs as new constraints are added or existing ones are modified.

The temporal constraints in an HTN can be represented as an STN which has one node for temporal reference (e.g., start of the time line), and two nodes per task, representing the start and end time of the task. The weighted edges between the start and end nodes establish the ordering and delay constraints. Any edge between the temporal reference point and another node restricts the domain of the node, e.g., restricts the earliest start time for a task. In addition, an HTN entails a set of constraints between a parent task and its children. For example, the sub-tasks can not start before or end after the parent task. Thus, there are inferred edges between the start and end nodes of a task and its children.

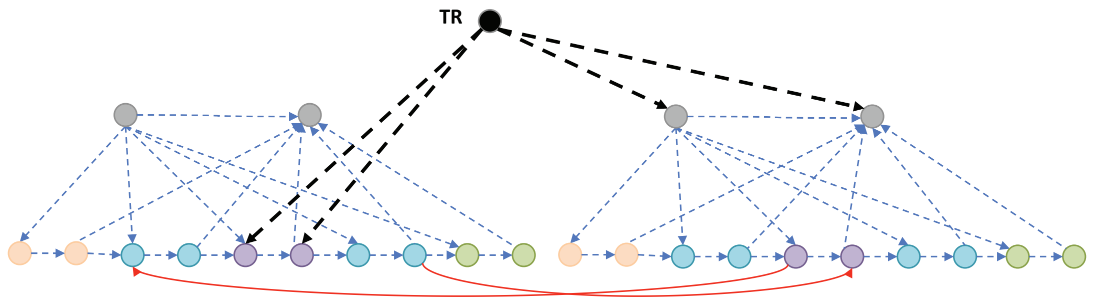
An STN that is generated from an HTN is *sibling restricted*. In a sibling restricted STN all the temporal constraints are either between parent and child tasks or between sibling tasks. (Yorke-Smith 2005) have shown how to exploit that structure to efficiently check the consistency of these networks and also compute the minimal domain of each temporal variable.

Example Domain and Problem Definition

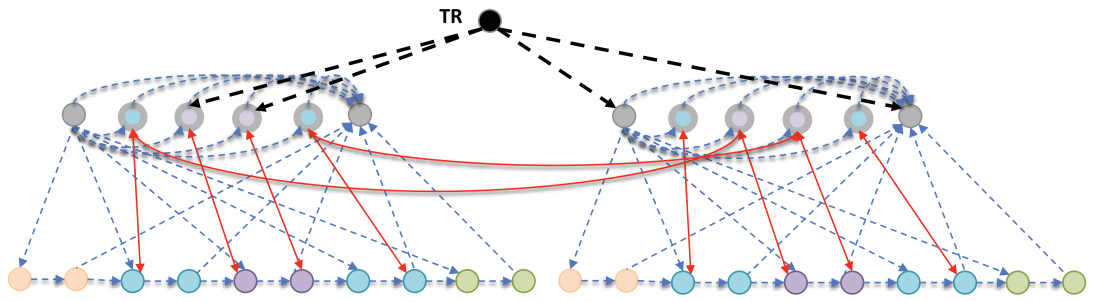
Our planning domain is based on a process that can be used to generate an Air Tasking Order (ATO) and is focused on military air missions with objectives including air interdiction (hit and destroy a target), reconnaissance (surveillance of an area), and providing tactical support (JSTARS aircraft with advanced sensor and communication capabilities). Each mission is composed of a set of activities, which can have sub-activities. We represent such hierarchical mission models as an HTN domain comprising 24 methods and 14 operators[1].

Figure 1 displays two missions for air interdiction (AI) and tactical support (JSTARS) and the first level of decomposition of each mission. The first level decomposition is the same for both missions, however all subtasks are composite tasks. The mission and target specific actions appear at later levels of decomposition affecting the navigation strategy and the core activity. For example, the core activity for an AI mission is a strike whereas for a JSTARS mission it is flying in an orbit. In this domain a typical AI mission has a decomposition tree with 58 tasks with a depth of 12. A JSTARS mission on the other hand involves less complex navigation

[1]The methods and operators are generated semi-automatically from the process models engineered for the JAGUAR system (Mulvehill, Rager, and Bostwick 2012).



(a) STN with start and end events



(b) STN with all events including milestones

Figure 2: (a) STN representing the constraints in Figure 1. This is not a sibling-restricted STN anymore due to the the domain-restriction constraints (shown as black thick-dotted lines) that limit the start and end time of an activity and synchronization constraints (shown as red solid lines). (b) HTNs with Temporal milestones. The nodes with gray thick border and blue center (second and fifth nodes for each top level task) are flight window milestones. The purple centered nodes (third and forth nodes for each top level task) are for the objective window. TR represents the temporal reference point, i.e., the start of the timeline.

thus has 33 tasks in its decomposition tree which has a depth of 9. Finally the decomposition tree size and depth for a reconnaissance (REC) mission are 55 and 12 respectively.

In an ATO problem, several missions, typically on the order of hundreds, have to be coordinated both in terms of time and resource usage. For example, it is common to have the JSTARS be in orbit during the time the AI mission is in flight, i.e., after the PreMission and before the PostMission activities. In Figure 1, the red dotted lines between the navigation tasks of the AI mission and the core activity of the JSTARS mission depicts this dependency. In the ATO domain the planning process is very mixed initiative and critical decisions are made by the human planner or requested by the end user (e.g., the commander). For our research in this domain, the inter-mission dependencies are stated as part of the goal task network (top level tasks and temporal constraints that are input to the planning problem) as opposed to being encoded as enabling conditions.

A key feature of this domain is the need to constrain the temporal windows of subtasks in addition to root tasks. For an AI mission the most important temporal window is when the target is going to be hit, thus the core activity window is constrained in the goal statement. Satisfying that constraint might impact other missions in the ATO. On the other hand, the start and end time of the mission are flexible.

Following the methodology outlined in the previous section we can represent the temporal relationships depicted in Figure 1 as the STN shown in Figure 2(a). The figure also shows the reference time point TR, the domain-restrictions enforced on the entire JSTARS mission, and the core activity of the AI mission. However, this graph is deceptively simple because the HTN in question is not fully decomposed. Once fully decomposed it will induce an STN with 183 nodes (one reference time point and two nodes for each of 58 tasks in the AI mission and 33 tasks in the JSTARS mission). Considering an ATO with hundreds of missions the size of the STN grows rapidly. Thus efficient reasoning is of utmost importance.

Castillo et al.(Castillo et al. 2006) proposed landmarks to support coordination between different task networks thus producing STNs such as in Figure 2(a). This extension however produces STNs with less desirable computational properties. More importantly it crosses the task abstraction boundaries that lie at the heart of the HTN paradigm. Specifically:

- The key in HTN planning is abstracting the tasks in a hierarchy and letting the user think in terms of top level tasks and their properties. In general the transformation from high level tasks to a plan, i.e., a partially ordered set of simple actions, is transparent to the user (meaning the intermediate decompositions are often invisible[2]). For ex-

[2]Obviously this is not the case for the domain expert who defined the task hierarchy along with the methods.

ample, in the ATO domain the operator needs to know the objective and flight window for each mission and how the objectives are related.

- Apart from the issue that in the temporal landmark approach a user needs to know subtasks, there is also the challenge of correctly referring to these subtasks. In a local context, such as within the same task network, unique labels can be used to differentiate multiple occurrences of the same task (or when there are recursive methods). However, if tasks in a decomposition are more globally accessible, then identifying the landmark variable associated with the correct instance of a reoccurring task is complicated. (Castillo et al. 2006) does not discuss how to handle this scoping problem in ATO-like domains.
- (Castillo et al. 2006) encode synchronization constraints using preconditions and effects. Both philosophically and practically this transformation seems unnecessary. One should be able to represent a temporal constraint without diving into state variables.
- Finally, when tasks from different networks are linked directly, the resulting STN becomes less decomposable, possibly requiring more constraint propagation thus yielding less efficient reasoning.

For all these reasons we propose a new extension to the HTN representation to support complex synchronization.

Temporal Milestones

As discussed earlier every task is associated with two special events: *start* and *end*. For complex tasks we will generalize this set to include an arbitrary number of events which we call *milestones*. We will use the notation $(event\, T)$ to denote an event associated with a task T including the start, end, and any milestone of the task. $Event(T)$ will denote the set of all events associated with T.

For complex tasks these milestones can represent certain stages in the execution of the task. Additionally, a milestone of a task can not be earlier than the start of the task or later than the end of the task. The mapping of milestones to execution stages is established in method definitions. Formally:

Definition 1: Let T be a complex task with a milestone labeled $midEvent$ and $M = < P, S, C >$ be a method that decomposes T into subtasks $S = S_1 \ldots S_n$ subject to constraints in C when the method precondition P is satisfied. M binds $midEvent$ if C contains a constraint $(midEvent\; T) = (event\; S_i)$ where $event \in E(S_i)$.

The milestone of a parent task can be bound to any event of its subtasks. It is also possible to generalize this definition to allow a fixed temporal distance from the subtask event such as $(midEvent\; T) = (event\; S_i) + \delta$ where $\delta \in \mathbb{R}$. The value of δ should be chosen carefully to avoid pushing the $midEvent$ beyond the parent task's execution window resulting an unsolvable STN.

Going from an HTN with milestones to STNs is straight forward: for every task there will be two nodes for the start and end time of the task and one node per milestone. The standard ordering arcs will ensure that the task start-node is before any other task event and the task end-node is after any other task event. The edges between parent start/end nodes and subtasks will be as before. In addition, milestone bindings will be represented by a pair of arcs between the milestone-node and a subtask event-node. In general an HTN with milestones will have an STN with $\sum_{t \in HTN} |Event(t)| + 1$ nodes.

task:	PerformMission
parameters:	missionType, objectiveArea
milestones:	flightBegin, flightEnd objectiveBegin, objectiveEnd

(a) Task

task:	PerformMission
subtasks:	S0:PreMission, S1:Navigate, S2:Core, S3:Navigate, S4:PostMission
constraints:	flightBegin=(start S1),

flightEnd=(end S3), objectiveBegin=(start S2)
objectiveEnd=(end S3), (end S0) ≤ (start S1)
(end S1) ≤ (start S2), (end S2) ≤ (start S3)
end S3) ≤ (start S4)

(b) Method

Figure 3: Example task and method with milestones.

For the ATO domain, we can augment the PerformMission task with four milestones in order to represent the time the aircraft is flying and performing the core mission. Figure 3 shows the task and method definition with these four milestones. For simplicity the preconditions of the method are not shown. The first two rows of the method constraints are for binding the task milestones and the last two rows enforce an ordering over subtasks.

Going back to the example in Figure 2(a); the STN with the milestone augmented HTN is shown in Figure 2(b). This STN has more nodes then Figure 2(a), however all arcs are between parent/child or sibling nodes. We will call an HTN domain *milestone-complete* if every method of every complex task binds every milestone. This is a desired property because it ensures that a task-milestone is always defined regardless of the way a task is achieved.

Theorem 1: Let T be a task with milestone $midEvent$ and N be the HTN for the task T. If N is generated using methods defined in a milestone-complete domain then there is a *source* task S in N such that $(midevent\, T) = (start\, S)$ or $(midevent\, T) = (end\, S)$.

This theorem follows from: 1) All milestones are bound (by milestone-complete domain); 2) A method sets a parent milestone to a subtask event (by Definition 1); and 3) For simple tasks the only task events are start and end. Basic tasks are not broken into stages thus don't have any milestones. Note that the source task does not have to be a simple task, (3) simply ensures that there will be a source task.

Using this theorem we can easily transform an STN with milestone events to a simpler one that only has the start and end events. The transformation involves:

- removing all milestone variables and arcs except the ones that come from top level constraints and,
- replacing the source/sink of any remaining arcs from/to a milestone-node with the start or end-node of the source task of the milestone.

Ultimately this transformation can be used to demonstrate that the milestone representation does not change the minimal domain for task execution windows.

Going from a two-event per task STN to a milestone version (simultaneously converting tasks and methods with milestones) is more complex. For every arc $a = (s_1, s_2)$ such that s_1 and s_2 are not siblings the transformation steps are:

1. Find the closest node pair t_1 and t_2 that are siblings and are ancestors of s_1 and s_2 respectively
2. Starting with t_1 create a milestone m_a (and a node for it) for each task that is on the path from t_1 to s_1 (not including s_1)
3. Connect each m_a nodes on the path finally to s_1
4. Do steps 2 and 3 for tasks from t_2 to s_2.
5. Create an arc between the m_a milestones of t_1 and t_2
6. Remove arc $a = (s_2, s_2)$.

This transformation will be incomplete if a task appears multiple times in the HTN that produced the original STN. In this case, the milestones from different instances need to be consolidated. Furthermore, if different methods are used to decompose the same task then in some of the sub-trees some of these milestones might be unbound. In this case, semantic intervention would be necessary to produce a milestone-complete HTN domain. Regardless of the incompleteness it is easy to verify that the minimal execution window of the tasks is equal.

Reasoning with Milestones

For efficient reasoning with milestone augmented HTNs a generalized version of the Sibling Restricted Propagation (SR-PC) algorithm (Yorke-Smith 2005) can be utilized. The steps that need generalization are:

- The distance matrix generation step should include milestones, i.e., extra rows and columns in the matrix. For algorithmic purposes STNs are represented as distance matrices.
- The local domain of a task is now the set of all pairwise distances between task events not just the distance between start and end nodes. The update and test steps should loop over all elements of the set.

We present the DelayedPropagation (D-PC) algorithm in Algorithm 1 which is an alternative but similar approach where the propagation of goal constraints is performed at a later stage. In D-PC the temporal domain of each top level HTN is computed separately and then combined with goal constraints to further restrict the domains of the top level events. Finally, the reduced domains are propagated down to the other tasks in the HTNs. The method for computing the minimal STNs is left unspecified leading to a range of algorithms. The choices range from naive PC (all-pair-shortest path (Floyd 1962)) to generalized SR-PC or even D-PC.

The main difference between SR-PC and D-PC is akin to a pre-order versus post-order traversal of a tree. A motivation of D-PC in delaying the propagation of top level constraints is providing a means to selectively propagate these constraints. This is a useful property for supporting conflict detection and deconfliction capabilities. For example, in a mixed initiative system when the constraints imposed yield no solution, a conflict detection step is needed to identify which of the constraints is unsatisfiable. A simple approach to this problem is to add the constraints incrementally (in some priority based order) and point to the first constraint that introduces inconsistency. SR-PC would unnecessarily recompute the basic temporal relationships between the tasks of an HTN.

Algorithm 1 DelayedPropagation: D-PC

```
Require: A set of hierarchical task networks $N$, a set of
    temporal constraints $C$ referring to events of top tasks
    in $N$, and a temporal reference point $TR$.
Ensure: Returns $false$ if the underlying STN is unsolvable
    o.w. returns $true$
 1: Let $S$ be an STN with only $TR$ node.
 2: for HTN $n \in N$ do
 3:     Let $t$ be the root task in $n$
 4:     Compute the minimal STN for $n$ with $TR$
 5:     if STN is inconsistent then
 6:         Return $false$
 7:     Add to $S$ all events of root task $t$ and edges between
        them and/or the $TR$
 8: Augment $S$ with constraints in $C$
 9: Compute the minimal STN for $S$
10: if $S$ is inconsistent then
11:     Return $false$
12: for HTN $n \in N$ do
13:     Compute the minimal STN for $n$ with $TR$ augment
        with event domains from $S$
14:     if STN is inconsistent then
15:         Return $false$
16: Return $true$
```

D-PC on the other hand, imposes the goal constraints on the fully connected event network of the top-level tasks thus allowing incremental propagation without recomputing the HTN induced STNs.

De-conflicting Goal Constraints

Using this property we can deconflict goal-constraints easily. We have applied this in the Mixed Initiative COA[3] Critic Agents (MICCA) framework to deconflict any goal-temporal constraints that are detected during the ATO planning process. Instead of an STN with hundreds of nodes the deconfliction step uses a simple STN that contains only events of the top-level tasks (Figure 4).

In the ATO domain, each task has six events, $start$, end, $objectiveWindowStart$, $objectiveWindowEnd$, $flightWindowStart$, and $flightWindowEnd$. The initial time domains and the distance between each event are computed separately for each mission. The goal constraints are incrementally added into the graph in the order of their importance. If at any point the network becomes inconsistent, the user is notified and the minimal windows (for events of the top tasks only) computed from the last-consistent network are displayed. The operator uses these windows to adjust

[3]Course of Action

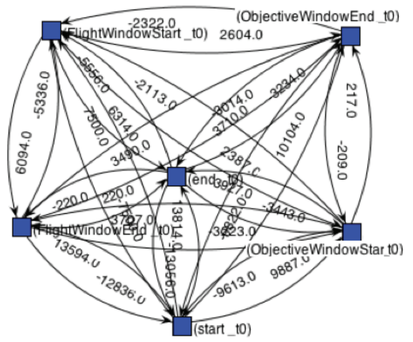
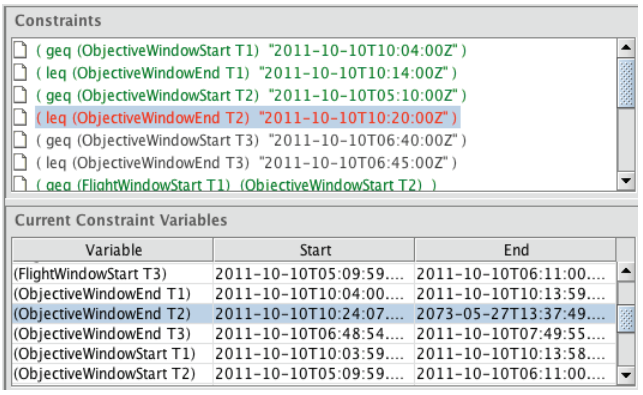


Figure 4: Simple STN with all events (milestones, start and end) of a single mission. The weights on the edges represent the temporal distance between events.

the constraint in question. Figure 5 displays a deconfliction session for a three mission scenario. The goal constraint in red states that the $objectiveWindowEnd$ for the task with label $T2$ should end by 10:20am. However adding this constraint leads to an unsolvable STN. The bottom window shows that the feasible values for this milestone lie between 10:24am and 13:37pm, suggesting that moving the deadline to 10:25am would resolve the problem.



Constraints

(geq (ObjectiveWindowStart T1) "2011-10-10T10:04:00Z")
(leq (ObjectiveWindowEnd T1) "2011-10-10T10:14:00Z")
(geq (ObjectiveWindowStart T2) "2011-10-10T05:10:00Z")
(leq (ObjectiveWindowEnd T2) "2011-10-10T10:20:00Z")
(geq (ObjectiveWindowStart T3) "2011-10-10T06:40:00Z")
(leq (ObjectiveWindowEnd T3) "2011-10-10T06:45:00Z")
(geq (FlightWindowStart T1) (ObjectiveWindowStart T2))

Current Constraint Variables

Variable	Start	End
(FlightWindowStart T3)	2011-10-10T05:09:59....	2011-10-10T06:11:00....
(ObjectiveWindowEnd T1)	2011-10-10T10:04:00....	2011-10-10T10:13:59....
(ObjectiveWindowEnd T2)	2011-10-10T10:24:07....	2073-05-27T13:37:49....
(ObjectiveWindowEnd T3)	2011-10-10T06:48:54....	2011-10-10T07:49:55....
(ObjectiveWindowStart T1)	2011-10-10T10:03:59....	2011-10-10T10:13:58....
(ObjectiveWindowStart T2)	2011-10-10T05:09:59....	2011-10-10T06:11:00....

Figure 5: The constraints in the top window are incrementally validated. The ones are green are safe, the ones in black have not been added into the STN yet, and the one in red leads to an unsolvable set. The bottom window displays the minimal domain of each task event.

Experimental Results

Our experiments are all based on our ATO domain. We have created several scenarios with 3 to 100 missions. All scenarios have solutions. We have implemented D-PC with the naive PC algorithm ($O(n^3)$ running time where n is the number of vertices). We then compared the time and computational space required by both D-PC and PC on the STNs from the scenarios. Figure 6 shows the ratio of the computational space and time required by PC to D-PC in log-scale (note that the X axis is equidistant). In terms of time (green line in Figure 6) the overhead introduced by the milestones enables quadratically better performance (as indicated by

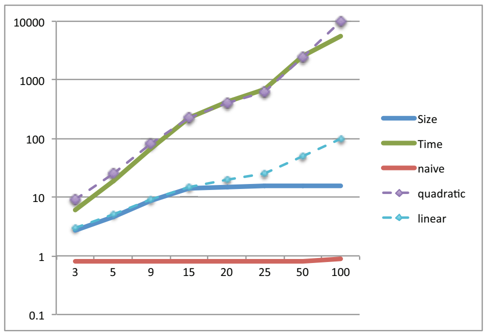


Figure 6: The ratio of computational space and time performance of PC over D-PC are linear and quadratic respectively.

quadratic function n^2). As for computational space (blue line), the linear advantage is seen up to fifteen missions and then the ratio of computational space requirement of D-PC to PC stays relatively constant. This is because after fifteen missions the dominating STN computation on the graph has all top-level events.

One can argue that the positive results from our experiments are partly due to some of the domain characteristics. For example, the number of milestone events introduced by each top task is only four and the source of these milestones are at the second level of decomposition thus introducing only four more milestone events bringing the total to eight. To give a better idea about the domain characteristics, for the 100 mission scenarios, we had 800 extra nodes due to milestones and over 600 goal constraints. Compared to the size of an STN per mission eight more nodes are not a significant overhead even for the naive PC algorithm (the red line in Figure 6). In the future we plan to investigate the how total number of milestones and the depth of the source nodes affect the performance.

Conclusions

This paper extends the HTN representation with temporal milestones that allows complex temporal task synchronization while respecting the task abstraction boundaries. Temporal milestones selectively expose intermediate events in a complex task. Temporal constraint networks derived from HTNs with milestones are decomposable which leads to efficient consistency checking algorithms. We presented our D-PC algorithm that takes advantage of the milestone induced structural properties. We also showed how D-PC can be used for conflict detection and deconfliction.

Our future work will investigate the application of this algorithm on other domains, build more efficient implementations of D-PC, and evaluate how this approach compares with other baseline algorithms such as incremental propagation as in land-mark variable approach. Further empirical evaluation will deepen our understanding of the generality and performance of milestone approach. Regardless of computational performance, the representational and practical advantages of milestones over landmarks will still hold.

References

Castillo, L. A.; Fernández-Olivares, J.; García-Pérez, Ó.; and Palao, F. 2006. Efficiently handling temporal knowledge in an HTN planner. In *Proceedings of the Sixteenth International Conference on Automated Planning and Scheduling, ICAPS 2006*, 63–72.

Cesta, A., and Oddi, A. 1996. Gaining efficiency and flexibility in the simple temporal problem. In *Proceedings of the 3rd Workshop on Temporal Representation and Reasoning (TIME'96)*, TIME '96, 45–. IEEE Computer Society.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Erol, K.; Hendler, J.; and Nau, D. S. 1994. Semantics for hierarchical task-network planning. Technical report, University of Maryland at College Park, College Park, MD, USA.

Floyd, R. W. 1962. Algorithm 97: Shortest path. *Commun. ACM* 5(6):345–.

Laborie, P., and Ghallab, M. 1995. Planning with sharable resource constraints. In *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 2*, IJCAI'95, 1643–1649.

Mohr, R., and Henderson, T. 1986. Arc and path consistency revisited. *Artificial Intelligence* 28(2):225–233.

Mulvehill, A.; Rager, D.; and Bostwick, R. 2012. Collaborative incremental model development for an adaptive model-driven planning system. In *Cognitive Methods in Situation Awareness and Decision Support (CogSIMA)*, 1 –6.

Myers, K. L.; Tyson, W. M.; Wolverton, M. J.; Jarvis, P. A.; Lee, T. J.; and desJardins, M. 2002. Passat: A user-centric planning framework. In *Proceedings of the Third International NASA Workshop on Planning and Scheduling for Space*.

Planken, L. 2008. Incrementally solving the stp by enforcing partial path consistency. In *Proceedings of the 27th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2008)*, 87–94. IEEE Computer Society.

Tate, A.; Drabble, B.; and Kirby, R. 1994. O-plan2: an open architecture for command, planning and control. In *Intelligent Scheduling*, 213–239. Morgan Kaufmann.

Wilkins, D. E. 1997. Using the sipe-2 planning system - a manual for sipe-2, version 4.14.

Yaman, F., and Nau, D. S. 2002. Timeline: An HTN planner that can reason about time. In *AIPS Workshop on Planning for Temporal Domains*, 75–81.

Yorke-Smith, N. 2005. Exploiting the structure of hierarchical plans in temporal constraint propagation. In *Proceedings of the 20th national conference on Artificial intelligence - Volume 3*, AAAI'05. AAAI Press.