Planning in Dynamic Environments: Extending HTNs with Nonlinear Continuous Effects

Matt Molineaux[1], Matthew Klenk[2], and David W. Aha[2]
[1]Knexus Research Corporation; Springfield, VA 22153
[2]Navy Center for Applied Research in Artificial Intelligence;
Naval Research Laboratory (Code 5514); Washington, DC 20375
matthew.molineaux@knexusresearch.com | {matthew.klenk.ctr,david.aha}@nrl.navy.mil

Abstract

Planning in dynamic continuous environments requires reasoning about nonlinear continuous effects, which previous Hierarchical Task Network (HTN) planners do not support. In this paper, we extend an existing HTN planner with a new state projection algorithm. To our knowledge, this is the first HTN planner that can reason about nonlinear continuous effects. We use a *wait* action to instruct this planner to consider continuous effects in a given state. We also introduce a new planning domain to demonstrate the benefits of planning with nonlinear continuous effects. We compare our approach with a linear continuous effects planner and a discrete effects HTN planner on a benchmark domain, which reveals that its additional costs are largely mitigated by domain knowledge. Finally, we present an initial application of this algorithm in a practical domain, a Navy training simulation, illustrating the utility of this approach for planning in dynamic continuous environments.

1 Introduction

Hierarchical Task Network (HTN) planning is a proven technique for quickly generating large plans to solve problems in challenging environments (e.g., strategy simulations). These planners often represent changes to the environment as discrete and instantaneous, even though aspects of these environments change *continuously* over time, frequently as the result of exogenous events. We show that this limiting assumption can be problematic for HTN planners. Therefore, we developed SHOP2$_{PDDL+}$, an extension of SHOP2 (Nau *et al.* 2003) that can reason about nonlinear continuous effects. In particular, it reasons explicitly about *fluents*, which are values that change over time (e.g., the amount of fuel on a ship), *processes*, which describe when and how they change, and *events*, which describe instantaneous occurrences resulting from fluent changes. SHOP2$_{PDDL+}$ uses a novel state projection algorithm to predict the continuous and discrete state of the environment at any arbitrary point in the future, and a *wait* action to control the passage of time during planning.

In addition to extending SHOP2, we also: (1) introduce a motivating, paradigmatic stunt car planning domain that requires explicit reasoning about nonlinear continuous effects; (2) empirically compare (on benchmark tasks) the performance of SHOP2$_{PDDL+}$ with SHOP2 and COLIN (Coles *et al.* 2009), a planner that can reason about linear continuous effects; and (3) describe an application of SHOP2$_{PDDL+}$ to tasks involving a Navy training simulation, which demonstrates that our algorithm can be applied to practical problems.

We begin by discussing this training simulation and our planning representation. Next, we introduce the wait action, the state projection algorithm, and its integration with SHOP2. Then we empirically assess our planner's performance on a range of planning problems and close with a discussion of related and future work.

2 TAO Sandbox and PDDL+

The TAO Sandbox is a strategy simulator used by the US Navy for training Tactical Action Officers in anti-submarine warfare (Auslander *et al.* 2009). Using it, trainees accomplish their objectives by giving orders to naval ships, planes, and helicopters. Vessel positions, fuel levels, heading, and speed are important fluents in this domain. The trainee's actions are orders, which occur instantaneously. The effects of these orders may be instantaneous (e.g., launch a helicopter), of fixed duration (e.g., move to a specific location), or of indefinite duration (e.g., follow another vessel). Therefore, agents interacting with the TAO Sandbox must reason about instantaneous changes and continuous effects.

The HTN extensions that we present are motivated by interest in applying a continuous planning (desJardins *et al.* 1999) agent in the TAO Sandbox domain. It must generate plans *and* monitor their execution for opportunities and failures. Because the TAO Sandbox is partially observable, this agent must monitor both the discrete and continuous state of the environment during plan execution. Frequently, knowing the value of fluents at individual time points is insufficient. Consider a vehicle that should have five gallons of fuel when reaching its destination. If the vehicle's fuel level is changing more rapidly than expected, perhaps due to a leak, then the agent should detect this while the vehicle is moving and not wait until after the action completes.

PDDL+ (Fox and Long 2006) was designed to support representations of mixed discrete-continuous planning domains. In PDDL+, changes to the discrete state occur as the result of both the agent's *actions* and *events* occurring in the environment. As mentioned in Section 1, *processes* describe changes to fluents occurring over time, while *events* describe discrete state changes occurring at an instantaneous point in time. Each PDDL+ planning domain includes *definitions* for its processes and events in terms of their *participant types*, *conditions*, and *effects*. A process is *active* for each set of objects that satisfies a process definition's participant types and conditions at a given time. All of the effects of processes are continuous and represented using algebraic functions, which describe the values of fluents with respect to time. Similarly, an event *occurs* when a set of objects satisfy the participant types and conditions of an event definition. The effects of an event describe discrete changes to the state; as a result, the set of active processes may be altered. In Section 3, we present a method for projecting future states that is useful for HTN planning in environments described using PDDL+.

3 Extensions for Continuous Effects

SHOP2$_{\text{PDDL+}}$'s extensions to SHOP2 (Nau *et al.* 2003) include the addition of a *wait* action, which allows the planner to reason about continuous effects, and a state projection algorithm, which projects the effects of active processes and occurring events. Section 3.3 presents an example of this algorithm.

3.1 Wait Action

SHOP2 is a state space planner; at each step in its search, applicable actions are considered to extend an existing plan, and a new state is extrapolated. We introduce a special *wait* action that, for the purposes of search, appears to function as any other action. The wait action takes one argument, *wait-time*, representing the action's duration.[1] It is always applicable, and because time passes during this action, it is necessary to compute the effects of the active processes and events that occur. All other actions are instantaneous and added sequentially with any number of actions occurring between waits. Although this is similar to a mechanism described by McDermott (2003), our wait action gives an agent the ability to wait for *any* amount of time. However, this freedom requires the agent to reason about how much time is appropriate to wait.

3.2 State Projection Algorithm

Changes in the environment during the wait action are projected using the state projection algorithm summarized

[1]This could easily be extended to include an arbitrary condition. In this case, the duration would be until the condition was satisfied in the environment.

Legend: S=State P=Planning domain, *ce*=continuous effect, *c*=continuous condition, *f*=fluent, $u_f(t)\in F$=fluent update functions, $th_c(t)\in E$=event threshold functions, t_{ee}=earliest event time, t_{init}=start time, t_{end}=wait end

```
State **ProjectState**($t_{init}$, $t_{end}$, $S$, $P$)
//Step 1: Build fluent-update-table F
  For each process definition $p \in P$
    For each $p_i$ in instantiations($p$)
      For each ce in $p_i$, add $u_f(t)$ to $F$
  //Step 2: Determine eligible-events
  For each event definition $e \in P$
    For each $e_i$ in instantiations($e$)
      Add $e_i$ to eligible-events
      Add $th_c(t)$ to $E$ for each $c$ in $ei$
  //Step 3: Update state to after next event
  do until ($\exists e_i$ , occurs($e_i$ ) or $t_{ee}$=$t_{end}$)
    For each $th_c(t)$ in $E$, $t_{ee}$=minRoot($t_{init}$ ,$t_{end}$, $th_c(t)$))
    For each $u_f(t)$ in $F$, $S.f$=$u_f(t_{ee})$
    For each $e_i$ in eligible-events
      if(occurs($e_i$)), S=updateState($e_i$,$S$)
  if($t_{ee}$!=$t_{end}$)
    then ProjectState ($t_{ee}$, $t_{end}$, $S$, $P$)
    else return($S$)
```

Figure 1: State projection algorithm

in Figure 1. It takes as inputs: the starting time (t_{init}) and ending time (t_{end} = t_{init} + *wait-time*) of the wait action, the current state, and the planning domain. This algorithm has three steps: building the *fluent update table*, determining *eligible events*, and updating the state until the time of the next event. This is repeated until no events are left to occur within the duration of the wait action.

The fluent update table F describes all continuous changes occurring after time t_{init}. To build it, the algorithm iterates over each process definition p from the planning domain. For each set of objects that satisfy p's participant types and conditions, an active process p_i is created. For each fluent f that is affected by a continuous effect of p_i, a *fluent update function* $u_f(t)$ is created and added to F. (If there are multiple fluent update functions referring to the same fluent, they are combined through summation into a single function in F.)

In step 2, for each instantiation e_i of each event definition e, this algorithm inserts e_i into the set of *eligible events* (i.e., that may occur as a result of the active processes). Also, it uses F to create an *event threshold function* $th_c(t)$ for each continuous condition c of each e_i, where $th_c(t)$=0 when c is satisfied at time t.

Because $th_c(t)$ may be defined using other fluents, this algorithm (recursively) replaces them with their fluent update functions $u_f(t)$ until only constants and the time variable t remain.

Step 3 determines when the next event will occur and updates the state accordingly. It begins by initializing the earliest potential event time t_{ee} to the latest time to be considered, t_{end}. Then, for each event threshold function $th_c(t)$, it searches for a root (i.e., where $th_c(t)$=0) in the range [t_{init}, t_{ee}). If one is found, the algorithm updates t_{ee} to the minimum root $t_{minRoot}$ in this range. That is,

$th_c(t_{minRoot})=0$ and $\forall t\ th_c(t)=0,\ t \geq t_{minRoot}$. Thus, $t_{minRoot}$ is the next time point at which event e_i may potentially occur. To find the minimum root of $th_c(t)$, recursive searches are made for extrema (i.e., maxima and minima) using golden section search and roots using bisection (Press *et al.* 2007). When a root is found at time t_{root}, search continues for the smaller range $[t_{init}, t_{root})$ until all extrema in the search range are either all negative or all positive. This procedure will find the first root in the range provided the function is continuous and smooth over $[t_{init}, t_{ee})$.At this point, we know $t_{minRoot}=t_{root}$ for $th_c(t)$. After iterating through each event threshold function, the resulting t_{ee} is the next time point at which any event may occur.

Next, the algorithm updates the continuous state and determines if any events occur. For each fluent update function $u_f(t)$, the algorithm updates fluent f to $u_f(t_{ee})$. Then, if the conditions of any of the eligible events e_i are satisfied, the discrete state is updated accordingly. Otherwise, Step 3 is repeated. However, if an event did occur, then the set of active processes may have changed. Therefore, the entire procedure is repeated until no event occurs prior to t_{end}, at which time all fluents are updated to $u_f(t_{end})$. We next present an example to illustrate this.

3.3 Example

Table 1: Process for projecting a ship's position, where #t is a special variable that denotes time in PDDL+

Process Name	`ShipMovement`
Participants	`?ship` type = `NavalShip`

Conditions (Discrete Only)	`(movingTo ?ship ?x ?y)`
	`(speedOf ?ship ?speed)`

Effects (Continuous Only)	`(increase (atX ?ship)`
	`   (* (cos (headingOf ?ship))`
	`      ?speed #t))`
	`(increase (atY ?ship)`
	`   (* (sin (headingOf ?ship))`
	`      ?speed #t))`

Consider the following example of a ship moving in the TAO Sandbox environment. To represent a ship's motion, we define the ship movement process shown in Table 1. This process has one participant of type `NavalShip` representing the moving ship. Its conditions include the ship's destination and speed. The continuous effects define functions that update the ship's location with respect to time. The update functions are defined in terms of the fluent `headingOf`, which could vary continuously at the same time as a result of another active process.

When the ship reaches its destination, its new position will trigger the end of movement event defined in Table 2. Like the `ShipMovement` process definition, the `EndOfMovement` event definition has one participant of type `NavalShip`. It has two discrete conditions and one continuous condition relating the fluents representing the ship's coordinates to its destination. The effect of this event is that the ship is no longer moving toward its destination.

Table 2: Event definition for the end of a ship's movement

Event Name	`EndOfMovement`
Participants	`?ship` type = `NavalShip`

Conditions (Discrete and Continuous)	`(movingTo ?ship ?x ?y)`
	`(speedOf ?ship ?speed)`
	`(<=(dist(atX ?ship)(atY ?ship)`
	`         ?x ?y)`
	`     0.5)`

Effects (Discrete and Continuous)	`(not (movingTo ?ship ?x ?y))`
	`(not (speedOf ?ship ?speed))`
	`(speedOf ?ship 0)`

Consider a situation in which the agent has just taken the action of ordering a ship, `Ship1`, to move to location `(5.6, 7.8)`. Figure 2 shows the current state. The continuous state includes fluents describing the location and heading of the ship and the discrete state includes the ship's speed and its current orders.

From this state, the planner applies the `wait(2)` action. To determine the state after 2 time units, the planner uses the state prediction algorithm with the above event and process definitions. There is only one possible set of participants for the `ShipMovement` process definition:

```
(atX Ship1) = 3.4 (Continuous)
(atY Ship1) = 2.3 (Continuous)
(headingOf Ship1) = 68.2 (Continuous)
(speedOf Ship1 20) (Discrete)
(isa Ship1 NavalShip) (Discrete)
(movingTo Ship1 5.6 7.8) (Discrete)
```

Figure 2: Initial TAO Sandbox state (simplified for clarity)

`?ship = Ship1`. The conditions are satisfied for this binding, resulting in one active process with two fluent update functions:

```
(atX Ship1), uf(#t) =
   (+ 3.4 (*(cos (headingOf Ship1)) 20 #t))
(atY Ship1), uf(#t) =
   (+ 2.3 (*(sin (headingOf Ship1)) 20 #t))
```

The fluent update table *F*, consisting of these two functions, determines the ship's location until the next event. To project the state forward, we generate a set of eligible events. In this situation, there is only one set of objects that satisfies the EndOfMovement event's types and discrete conditions: {`?ship = Ship1`}. This event has one continuous condition:

```
(<= (dist (atX Ship1) (atY Ship1) 5.6 7.8)
     0.5)
```

From this, the algorithm creates the following event threshold function:

```
  thc(#t)=
    (- (dist (+ 3.4 (* (cos 68.2) 20 #t))
             (+ 2.3 (* (sin 68.2) 20 #t))
             5.6 7.8)
       0.5)
```

Step 3 of the state projection algorithm begins by finding the minimum root for $th_c(\#t)$. In this case, $th_c(\#t) = 0$ when $\#t = .271$. To determine if any events occur at this point, the algorithm updates each fluent $f \in F$ to the value given by $u_f(.271)$. In this case, the values for `(atX Ship1)` and `(atY Ship1)` are found to be 5.41 and 7.34, respectively. These satisfy the continuous condition for the eligible `EndOfMovement` event. Therefore, this event occurs, which updates the discrete state by deleting the movement statement, `(movingTo Ship1 5.6 7.8)`, and updating the ship's speed, `(speedOf Ship1 0)`. No other events are eligible to occur. Because .271 is less than 2, another iteration of the algorithm is performed. No processes are now active, as no set of objects satisfies the `ShipMovement` process definition conditions. Therefore, the state remains the same until wait action ends.

3.4 Incorporation into HTN planning

```
(defMethod
  (movingShipAndWait ?ship ?to-x ?to-y)
  :preconditions
  (durationOfMovement ?ship ?to-x ?to-y
                      ?duration)
  :subtasks
  (move ?ship ?to-x ?to-y)
  (wait ?duration))

(defAction (move ?unit ?x ?y)
  :preconditions
  (not (movingTo ?unit ?x ?y))
  :effects (movingTo ?unit ?x ?y))
```

Figure 3: HTN method and action for moving a ship

Our agent uses the SHOP2 (Nau *et al.* 2003) planner. SHOP2 takes as input a *task* to be performed and produces a sequence of *actions*. The *task* is decomposed into *subtasks* and actions using *methods*. We use this hierarchical structure to determine when to apply a wait action and for how long. Consider the ship movement example above. The method and action in Figure 3 decompose the `movingShipAndWait` method into a move action, which issues a move order to a single ship, and a wait action, which allows the movement process to update the state over time. The `durationOfMovement` precondition determines the predicted amount of time required for the ship to reach its destination. Therefore, the planner can predict the state after the ship's movement and continue planning.

4 Evaluation

To evaluate the utility of our approach, we performed a series of case studies with the following objectives:

- Illustrate the class of problems for which our method is capable of generating plans
- Determine the comparative costs of our approach vs. existing methods on benchmark problems
- Ascertain its applicability in strategy simulations

4.1 Illustrative Domains

First, we tested SHOP2$_{PDDL+}$ on Howey and Long's (2003) generator problem, which has linear continuous effects. This problem requires a generator to run for 100 time units. Its fuel tank has a 90 unit capacity, and it consumes one unit of fuel per unit of time. A refill action is available that adds two units of fuel per unit of time. We added an HTN method to the planning domain that determines when, after the generator starts running, to begin the refueling process. Our state prediction algorithm accurately projects the continuous effects of the refueling process, and the planner creates a successful plan.

To illustrate the importance of reasoning about nonlinear continuous effects, we introduce a stunt car domain inspired by Hollywood action movies. This domain includes a stunt car and a wall. Its two processes, acceleration and changing position, affect the car's velocity and position using the following continuous effects:

```
(increase (vel ?obj) (* (accel ?obj) #t))
(increase (pos ?car) (* (vel ?car) #t))
```

As a result, the position of the car is a nonlinear function of time. This domain's only action is to apply the brake, which begins the acceleration process with an acceleration of -14m/s^2. In this domain, a crash, described by an event definition, occurs when the positions for the car and wall are the same. If the crash occurs at over 13m/s, the driver dies. If the crash occurs below 9m/s, the crash is not spectacular enough. If the crash occurs between 9m/s and 13m/s, then there is spectacular crash and the driver survives, accomplishing the goal. In our simple test problem, the stunt car begins 100m away from the wall, traveling at a speed of 44m/s. Using an HTN method, SHOP2$_{PDDL+}$ is able to generate a successful plan, indicating that it can reason about nonlinear continuous effects.

4.2 Empirical Comparison: Rover Domain

To assess the additional costs of SHOP2$_{PDDL+}$, we compared its performance on a modified single rover domain[2] that was used to test COLIN (Coles *et al.* 2009), a linear continuous planner. Specifically, we compare SHOP2$_{PDDL+}$'s plan generation times on 20 planning problems from this domain to COLIN's previously reported plan generation times. These problems were generated using the IPC3 parameters permitting a comparison to SHOP2's performance in the competition (Long and Fox 2003). Given that the nature of these comparisons (i.e., domain-independent versus HTN planners running on different hardware) prevents a quantitative analysis, we focus on a qualitative assessment of the additional costs of SHOP2$_{PDDL+}$.

[2]PDDL domain and planning files were downloaded from the COLIN website at: personal.cis.strath.ac.uk/~amanda/ContinuousPlanning/

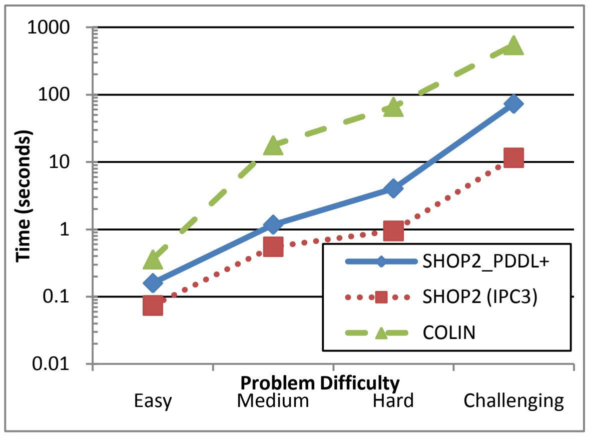

Figure 4: Comparison of plan generation times for SHOP2$_{PDDL+}$, SHOP2, and COLIN on rover problems, sorted by problem difficulty

The results, shown in Figure 4, compare the average planning time of SHOP2$_{PDDL+}$ to the other two planners. To better illustrate scaling issues, we group the problems by their difficulty as determined by the IPC3 parameters. Both SHOP2 and SHOP2$_{PDDL+}$ solved all the planning problems, while COLIN failed on three of the "challenging" and one of the "medium" problems.

Our analysis of these results identifies two key observations. First, the expected large performance gains of HTN planning vs. domain-independent approaches occur. Second, our comparison with SHOP2 indicates that there is significant overhead in projecting all continuous values through all states. An important aspect of future work is to explore dynamic programming methods for improving efficiency, but this result establishes a baseline performance for an HTN planner reasoning about continuous effects.

4.3 Application to the TAO Sandbox

In the TAO Sandbox, our planner can generate plans for 6 distinct scenarios. Our domain model contains 7 process definitions and 30 event definitions. We tested it in scenarios containing up to 12 distinct moving vessels. Typically, SHOP2$_{PDDL+}$ generates solution plans of 50+ actions in about 1 second. Our future work will explore the quality of the generated plans and the utility of state projection by assessing results of plan execution in this environment.

4.4 General Discussion

The generator and stunt car problems demonstrate that our planner can reason about domains with linear and non-linear continuous effects. Furthermore, results on the rover domains indicate that, while there is a significant cost in plan generation time, it is largely mitigated by domain knowledge required for HTN planning. The results from the rover domain and TAO Sandbox indicate that the overhead of our approach for planning with nonlinear continuous effects is sufficiently low to encourage its further application to strategy simulation tasks.

5 Related work

Within the HTN planning community, several efforts have focused on reasoning about temporal domains. The majority of this work focused on using the structure of HTNs to propagate temporal constraints (e.g., Yorke-Smith 2005, Castillo *et al.* 2006). We are not aware of any HTN planner that reasons about the (linear or nonlinear) continuous effects of actions or exogenous events.

Work exploring planning with continuous processes began with Zeno (Penberthy and Weld 1994), which modeled processes using differential equations. While an important first step, Zeno cannot reason about multiple processes that affect a single fluent.

More recently, a common approach to temporal planning problems has been to sidestep continuous reasoning by transforming each temporal action into an instantaneous action (Cushing *et al.* 2007). Unfortunately, this approach is insufficient for domains with rich temporal interactions between actions, processes, and events (e.g., domains in which concurrent actions are required to achieve a solution).

Recognizing this limitation, several planners have been developed for temporal reasoning about continuous effects. The Optop estimated-regression planner (McDermott 2003) uses a similar strategy to ours for temporally projecting the continuous state at the time of a wait action. Unlike our approach, it requires the planner to plan after the next event occurs instead of from an arbitrary point in the future. As indicated previously, COLIN can represent the continuous effects of durative actions (Coles *et al.* 2009). While useful in some environments, representations using durative actions, rather than processes, do not allow a planner to reason about the long-term consequences of exogenous events.

COLIN and Optop both focus solely on *linear* changes to fluents. SHOP2$_{PDDL+}$ instead reasons about processes whose effects contain *arbitrary* continuous functions. Similar to our work, Kongming (Li and Williams 2008) can project nonlinear continuous effects of durative actions using *flowtubes*. It encodes undersea navigation problems in a mixed logic linear/non-linear program solvable using standard techniques. While a promising approach, there is considerable difficulty in extending Kongming's approach to reason about actions with variable durations. However, actions with variable duration are common in strategy simulations (e.g., consider the ship-moving process for the TAO Sandbox). VAL (Howey *et al.* 2004) addresses polynomial continuous effects for plan validation and repair in mixed-initiative planning with application to another practical domain (i.e., space missions). In contrast, our focus is on *arbitrary* continuous effects in the context of *automated* planning.

Baral *et al.* (2002) define a similar state projection algorithm using an alternative formalization for planning with continuous effects. By using the PDDL+ formalization, our approach considers exogenous changes in the environment.

6 Conclusions

We presented a novel state projection algorithm for use in continuous temporal environments and illustrated its implementation in SHOP2$_{\text{PDDL+}}$, an HTN planner. Unlike previous approaches, our algorithm allows for continuous effects to contain arbitrary continuous functions, and our wait operator allows the planner to consider future actions at any time.

Our focus is on the projection of fluents within individual states with respect to time. Therefore, SHOP2$_{\text{PDDL+}}$ needs to predict the duration of actions *and* their continuous effects. This is necessary for planning, as shown above, *and* for plan-monitoring agents. SHOP2$_{\text{PDDL+}}$ was developed in the context of a continuous planning agent, which reasons about its goals while performing tasks in the TAO Sandbox simulation (Molineaux *et al.* 2010).

In future work, we intend to empirically evaluate SHOP2$_{\text{PDDL+}}$ to determine the scalability and effectiveness of its state projection algorithm across a variety of challenging tasks. We expect this algorithm to allow agents to generate plans for these scenarios and quickly identify potential problems and opportunities. We believe that continuous planning will improve the performance of intelligent agents in these strategy simulations, resulting in more flexible opponents and intelligent teammates.

Acknowledgements

This research was supported by DARPA IPTO (MIPRs 09-Y213 and 09-Y214). Matthew Klenk is supported by an NRC postdoctoral fellowship.

References

Auslander, B., Molineaux, M., Aha, D.W., Munro, A., & Pizzini, Q. (2009). *Towards research on goal reasoning with the TAO Sandbox* (Technical Report AIC-09-155). Washington, DC: Naval Research Laboratory, Navy Center for Applied Research on AI.

Baral, C., Son, T., and Tuan, L. (2002). A transition function based characterization of actions with delayed and continuous effects. In *Proceedings of KR-02.* p. 291-302

Castillo, L.; Fdez-Olivares, J.; Garcıa-Perez, O.; and Palao, F. (2006). Efficiently handling temporal knowledge in an HTN planner. In *Sixteenth International Conference on Automated Planning and Scheduling*.

Coles, A., Coles, A., Fox, M., and Long, D. (2009). Temporal planning in domains with linear processes. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. Pasadena, CA.

Cushing, W., Kambhampati, S., Mausam, and Weld, D. (2007). When is temporal planning *really* temporal planning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. p. 1852-1859. Hyderabad, India.

desJardins, M., Durfee, E., Ortiz, C., & Wolverton, M. (1999). A survey of research in distributed, continual planning. *AI Magazine*, 20(4), 13–22.

Fox, M. & Long, D. (2006). Modelling mixed discrete-continuous domains for planning. *Journal of Artificial Intelligence Research*, 27:235-297.

R. Howey and D. Long. (2003). Validating plans with continuous effects. In *Proc. of the 22nd Workshop of the UK Planning and Scheduling Special Interest Group*.

Howey, R., Long, D., and Fox., M. (2004). VAL: Automatic Plan Validation, Continuous Effects and Mixed Initiative Planning using PDDL. *The 16th IEEE International Conference on Tools with Artificial Intelligence*. pp. 294-301.

Li, H. & Williams, B. (2008). Generative systems for hybrid planning based on flowtubes. In *Proceedings of the 18th International Conference on Automated Planning and Scheduling (ICAPS)*.

Long, D. & Fox, M. (2003). The 3rd International Planning Competition: Results and analysis. *Journal of Artificial Intelligence Research*. 20: 1-59.

McDermott, D. (2003). Reasoning about autonomous processes in an estimated regression planner. In *Proceedings of the 13th International Conference on Automated Planning and Scheduling (ICAPS)*.

Molineaux, M., Klenk, M. & Aha, D. (2010). Goal-driven autonomy in a navy strategy simulation. In *Twenty-Fourth AAAI Conference on Artificial Intelligence* (AAAI-10). Atlanta, Georgia.

Nau, D., Au, T.-C., Ilghami, O., Kuter, U., Murdock, J. W., Wu, D., & Yaman, F. (2003). SHOP2: An HTN planning system. *Journal of Artificial Intelligence Research*. 20:379–404.

Penberthy, J. & Weld, D. (1994). Temporal Planning with Continuous Change. *Proceedings of AAAI-94*. Seattle, WA.

Press, W., Teukolsky, S., Vetterling, W., and Flannery, B. (2007). *Numerical Recipes 3rd Edition: the Art of Scientific Computing*. Cambridge University Press.

Yorke-Smith, N. (2005). Exploiting the structure of hierarchical plans in temporal constraint propagation. In *Proceedings of the 20th National Conference on Artificial Intelligence*. pp. 1223-1228. Pittsburgh, PA.