Delete-free Reachability Analysis for Temporal and Hierarchical Planning (full version)

Arthur Bit-Monnot, David E. Smith, Minh Do

Delete-free Reachability Analysis for Temporal and Hierarchical Planning

Arthur Bit-Monnot
LAAS-CNRS, Université de Toulouse
Toulouse, France
arthur.bit-monnot@laas.fr

David E. Smith and **Minh Do**
NASA Ames Research Center
Moffet Field, CA, USA
{david.smith, minh.do}@nasa.gov

Abstract

Reachability analysis is a crucial part of the heuristic computation for many state of the art classical and temporal planners. In this paper, we study the difficulty that arises in assessing the reachability of actions in planning problems containing sets of interdependent actions, notably including problems with required concurrency as well as hierarchical planning problems. In temporal planners, this complication has been addressed by augmenting a delete-free relaxation with additional relaxations, but this can result in weak pruning of the search space. To overcome this problem, we describe a more sophisticated method for reachability analysis that uses Dijkstra's algorithm for propagation of times through a reachability graph, combined with a pruning mechanism that recognizes unachievable cycles.

We also extend our approach to handle hierarchical planning problems, in which an action and its subactions are naturally interdependent. Evaluations were conducted on a diverse set of temporal domains using FAPE, a constraint-based temporal and hierarchical planner.

1 Introduction

Reachability analysis is crucial in computing heuristics guiding many classical and temporal planners. This is typically done by relaxing the action delete lists and constructing the reachability graph. This graph is then used as a basis to extract a relaxed plan, which serves as a non-admissible heuristic estimate of the actual plan reaching the goals from the current state. Due to the relaxation, the reachability analysis is optimistic and can also provide a lower-bound on the actual "cost" of reaching the goals.

Temporal planning poses some additional challenges for reachability analysis due to the temporal objective function of minimizing the plan's makespan. This objective function leads to the requirements on the heuristic guidance to not only estimate the total cost but also the earliest time at which goals can be achieved. This can be accomplished on the reachability graph by labeling: (1) propositions by the minimum time of the effects that can achieve them; and (2) actions by the maximum time of the propositions they require as conditions. Since the reachability graph construction process progresses as time increases, when all *start* conditions are reachable, a given action a is eligible to be added to the graph. However, there is the additional problem that

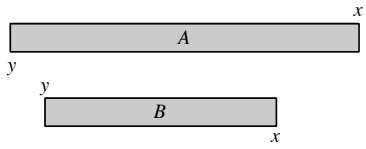


Figure 1: Two actions: A with a start effect y and an end condition x, and B with a start condition y and an end effect x.

a's end conditions must also be reachable, although they do not need to be reachable until the end time of a. To see why this is a problem for the conventional way of building the reachability graph, consider the two actions in Figure 1: action B achieves the end condition for action A, but requires a start effect of A before it can start. Thus, B cannot start before A, but A cannot end until after B has ended. This means that A is not fully reachable until B is reachable, but B is not reachable unless A is reachable. Whether this turns out to be possible depends on whether B fits inside of A. In this example, the reasoning is simple enough, but more generally, B might be a complex chain of actions.

Planners such as COLIN (Coles et al. 2012) and POPF (Coles et al. 2010) address this problem by splitting durative actions into instantaneous start and end events, and forcing a time delay between the start and end events. In our example, the start of A would be reachable, leading to the start of B being reachable, which leads to the end of B being reachable, and finally the end of A being reachable. This approach therefore concludes that A is reachable. Unfortunately, the same conclusion is reached in COLIN or POPF even when B does not fit inside of A, because this "action-splitting" approach allows A to "stretch" beyond its actual duration. While this is a satisfactory relaxation for problems with few interdependencies, it does not provide very good heuristic guidance for problems that involve a lot of action nesting, such as hierarchical style container actions that expand into subactions.

In this paper, we first present an approach to reachability analysis for durative actions that addresses the nesting problem described above. We use Dijkstra's algorithm for propagation of times through a reachability graph, together with a pruning mechanism that recognizes unachievable cycles. In the second part, we study some of the difficulties

that arise when performing reachability analysis in hierarchical planning and show that those challenges are similar to the ones encountered in temporal planning. We devise a compilation procedure that exposes the hierarchical features as additional conditions and effects in durative actions. The compiled problem is used as an input for reachability analysis of hierarchical planning problems.

2 Preliminaries

We first describe the temporal action model and other basic elements used throughout the rest of the paper.

2.1 Temporal Planning Model

In PDDL 2.2, a planning problem P is represented by a tuple $P \doteq \langle V, I, T, G, A \rangle$ where:

- V is a set of propositions.
- I is the initial state: a complete set of assignments of value `T` or `F` to all propositions in V.
- T is a set of timed-initial-literals, which are tuples $\langle [t]\ f := v \rangle$ with $f \in V$ and $t \in \mathbb{R}^+$ is the wall-clock time at which f will be assigned the Boolean value v.
- $G \subseteq V$ is a goal state: a set of propositions that need to be true when the plan finishes executing.
- A is a set of durative actions, each of the form $a \doteq \langle D_a, C_a, E_a \rangle$ where:
  - D_a is a set of constraints on the duration of the action. The actual duration of an action is referred to as d_a and takes a value in R^+ that is consistent with D_a.
  - C_a is the set of conditions. Each $p \in C_a$ is of the form $\langle (st_p, et_p)\ f = \mathtt{T} \rangle$ where st_p and et_p indicate the start and end time of the condition p relative to the action's start time. When $st_p = et_p = 0$ or $st_p = et_p = d_a$ then p is an instantaneous *at-start* or *at-end* condition. When $st_p = 0$ and $et_p = d_a$ then p is an *overall* durative condition. $f \in V$ is a proposition that must be true over the specified time period.
  - E_a is a set of instantaneous effects, each $e \in E_a$ is of the form $\langle [t_e]\ f := v \rangle$ where $t_e \doteq 0$ or $t_e \doteq d_a$ is the relative time at which the *at-start* or *at-end* effect e occurs.

A plan π of P is a set of tuples $\langle t_a, a, d_a \rangle$, in which an action $a \in A$ is associated to a wall-clock start time t_a and a duration d_a that satisfies the constraints in D_a. π is valid if it is executable in I and achieves all goals in G.

Beyond PDDL 2.2: We extend the temporal action model in PDDL2.2 to allow conditions expressed over sub-intervals of actions, and effects at arbitrary time points during an action. These features turn out to be particularly useful for encoding many temporal planning applications. We do this by allowing the times st_p and et_p of a condition p or t_e of an effect e to take an arbitrary value in $[0, d_a]$.

Discrete time model: Unlike PDDL 2.2, which assumes the continuous time model, we assume the discrete time model in which time changes in discrete steps. This is not essential to our approach, but simplifies the presentation. We therefore represent the durative conditions $\langle (st_p, et_p)\ f = \mathtt{T} \rangle$ in PDDL 2.2 as a sequence of consecutive instantaneous conditions $\langle [t]\ f = \mathtt{T} \rangle$ with $st_p \leq t \leq et_p$. For the rest of this paper, we will assume that all action conditions and effects occur at discrete time-steps t specified as either $t = \delta$ (at a constant duration δ after the start time of action a) or $t = d_a - \delta$ (at a constant duration δ before the end time of action a).

2.2 Delete-free Elementary Actions

To estimate when each fact can be achieved, our reachability analysis utilizes *elementary actions*, which are artificial actions created from the original temporal actions defined in the domain description. Elementary actions contain: (1) only a single 'add' effect and (2) the minimal set of conditions required to achieve that effect. Specifically, given a temporal action a, the set of elementary actions for a is created by:

1. Removing all 'delete' effects of a.
2. For each 'add' effect $e = \langle [t_e]\ f := \mathtt{T} \rangle$, creating a new elementary action a_e with e as the only effect of a_e.
3. Adding each condition $p \in C_a$ to a_e with an *optimistic* timing constraint on when p is needed. By *optimistic*, we mean requiring each $p \in C_a$ as late as possible with respect to the time at which e is achieved. Let lb_{d_a} and ub_{d_a} be the lower and upper bounds on the duration d_a of a and d_{a_e} be the duration of a_e, then this maximum lateness can be achieved by fixing the value of d_{a_e}:
   - If $t_e = \delta$, then set $d_{a_e} = ub_{d_a}$
   - If $t_e = d_a - \delta'$, then set $d_{a_e} = lb_{d_a}$

Figure 2 shows an example of a move action for a rover and its two elementary actions: $a_1 = moveFree(r, l, l')$ represents the start effect $e_1 = \langle [1]\ free(l) := \mathtt{T} \rangle$ and $a_2 = moveAt(r, l, l')$ represents the end effect $e_2 = \langle [d]\ at(l') := \mathtt{T} \rangle$. Since $t_{e_1} = 1$, we fix the duration of $moveFree(r, l, l')$ to be the upper-bound value of the duration d of the original action $move(r, l, l')$ and thus $d_{a_1} = 50$. On the other hand, d_{a_2} is set to be the lower-bound value 40 of d.

For easier illustration, an elementary action a is represented graphically by an action node a with (1) an outgoing effect edge $a \xrightarrow{X} f$ representing its effect $\langle [X]\ f := v \rangle$; and (2) one incoming condition edge $f \xrightarrow{-Y} a$ for each condition $\langle [Y]\ f = \mathtt{T} \rangle \in C_a$. The *reachability graph* is a pair $\langle N, E \rangle$ with N a set of action and fluent nodes and E a set of condition and effect edges for all elementary actions. The edges from fluents to actions encode the necessary delay between the conditions of an action and the action. The semantics is different for edges from actions to fluents, as each edge $a \xrightarrow{X} f$ represents one possible action choice a for achieving the fluent f.

move(r,l,l')

conditions:	$[0]\ at(r,l)$
	$[d{-}1]\ free(l')$
effects:	$[1]\ free(l)$
	$[1]\ \neg at(r,l)$
	$[d]\ \neg free(l')$
	$[d]\ at(r,l')$
duration constraints:	$40 \leq d \leq 50$

(a) Original action model

moveFree(r,l,l')

conditions:	$[0]\ at(r,l)$
	$[49]\ free(l')$
effects:	$[1]\ free(l)$

(b) Elementary action for the add effect $\langle [1]\ free(l) \rangle$

moveAt(r,l,l')

conditions:	$[0]\ at(r,l)$
	$[39]\ free(l')$
effects:	$[40]\ at(r,l')$

(c) Elementary action for the add effect $\langle [d]\ at(r,l') \rangle$

Figure 2: Action to move a rover r from location l to l' and its two elementary actions. The rover frees its original location one time unit after departing and requires its target location to be free one time unit before arriving.

3 Reachability Analysis

3.1 Definitions

An elementary action a is *applicable* once all of its preconditions are met. An action with an effect f is called an *achiever* of f. A fluent f becomes *achievable* after one of its achievers a becomes applicable, with the achievable time depending on the starting time of a and the time constraint on the effect of a that enables f. As a consequence of using the delete-free elementary actions, once a fluent is achievable at time t or an action is applicable at time t, it stays achievable/applicable at all subsequent time points.

Action a is applicable at time t (denoted by $applicable(a,t)$) if for all conditions $p = \langle [X] f = \mathtt{T} \rangle \in C_a$, p is achievable at time $t_p = t + X$ (denoted by $achievable(p,t_p)$). Similarly, a fact f is achievable at time t (i.e., $achievable(f,t)$) if there exist one achiever a of f such that a has an effect $e = \langle [X]\ f := v \rangle$ and a is applicable at time $t_a = t - X$ (i.e., $applicable(a,t_a)$).

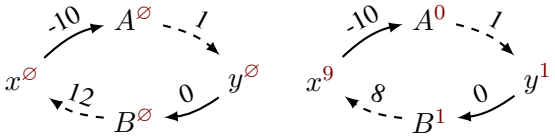
We define as the *earliest appearance* of action a (denoted by $ea(a)$), the smallest t for which $applicable(a,t) = \mathtt{T}$. Similarly, the *earliest appearance* of a fluent f is the smallest t for which $achievable(f,t) = \mathtt{T}$. The computation of $ea(a)$ and $ea(f)$ has traditionally been done by following the dynamic programming update rules below:

$$\begin{aligned}
\text{Initialization:}\quad & \forall f \in I : ea(f) = 0 \\
& \forall f \notin I : ea(f) = \infty \\
& \forall a \in A : ea(a) = \infty \\
\\
\text{Updating:}\quad & \forall a \in A : ea(a) = \max_{\langle [X]\ f=\mathtt{T} \rangle \in C_a} ea(f) - X \\
& \forall f \in V : ea(f) = \min_{\langle [X]\ f:=\mathtt{T} \rangle \in E_a} ea(a) + X
\end{aligned}$$

When the updating rules above are properly applied repeatedly, the collective values of $ea(f)$ and $ea(a)$ will reach a fix-point. We use $ea^*(f)$ and $ea^*(a)$ to denote the final values of $ea(f)$ and $ea(a)$ for all fluents f and actions a. If $ea^*(f) < \infty$ or $ea^*(a) < \infty$, we say that f or a is *reachable*, denoted by $reachable(f) = \mathtt{T}$ and $reachable(a) = \mathtt{T}$.

Intuitively, if an action or a fluent is not reachable by applying elementary actions, then it can not be achieved using the original action model. Therefore, a fluent cannot be true at a time earlier than $ea^*(f)$ and a reachable action A can never be executed at a time earlier than $ea^*(A)$.

3.2 Causal Loops in Temporal Planning



(a) (b)

Figure 3: Two reachability graphs built from the actions of Figure 1. A is an action with a duration of 10 time units, an end condition $\langle [10]\ x = \mathtt{T} \rangle$ and a start effect $\langle [1]\ y := \mathtt{T} \rangle$. B has the start condition $\langle [0]\ y = \mathtt{T} \rangle$ and the end effect $\langle [d_B]\ x := \mathtt{T} \rangle$. The duration of B, d_B, is respectively set to 12 and 8 in graphs (a) and (b). Each node is annotated in red with its earliest appearance time or $\varnothing$ if it is not reachable.

Let us consider what would happen when applying the dynamic programming rules to the reachability graphs of Figure 3. In Figure 3a, where B does not 'fit' into A, it would behave as expected: the positive cycle would maintain all the nodes with an infinite earliest start (i.e. unreachable). On the other hand, Figure 3b depicts a self-supporting causal loop where the effect y of A allows B to produce x early enough to achieve the end condition of A. Such self-supporting causal loops can be identified as cycles of negative or zero length in the reachability graph. If the cycle is of negative length, as in Figure 3b, the earliest appearances would be infinitely updated towards $-\infty$. In the case of a zero length cycle, the dynamic programming rules fail to detect that an update is needed to mark the node as reachable.

Temporal planning problems with such causal loops are identified by Cooper, Maris, and Régnier (2013) as *temporally-cyclic* problems and are characterized by sets of interdependent actions. The difficulty in handling them has been avoided in state of the art temporal planners by ignoring any condition that might be achieved through a self-supporting causal loop. In POPF (Coles et al. 2010), the reachability model is built by splitting durative actions into an instantaneous start-action and an instantaneous end-action, with the start-action using only the 'at-start' conditions. Applied to our example in Figure 2, this would

roughly result in ignoring the $\langle [49]\ free(l') = \mathtt{T} \rangle$ condition of $moveFree(r, l, l')$. This additional relaxation leads to reachability models that disregard any condition that could lead to a self-supporting causal loop.

Self supporting causal loops always contain an *after-condition* : a condition in C_a that appears at the same time or later than the effect of an elementary action a (Cooper, Maris, and Régnier 2013). To make the identification of after-conditions easier, we assume that a condition $\langle [X]\ f = \mathtt{T} \rangle$ is an after-condition iff $X > 0$. This restriction means that any negative edge in the reachability graph represents an after-condition. If necessary, this can be enforced by artificially shifting the start of all elementary actions to be one time unit before their effect.

3.3 Reachability Analysis with Causal Loops

To handle after-conditions during reachability analysis, as detailed in Algorithm 1, we alternate two steps: (1) a first step propagates achievement times while ignoring all after-conditions, performed by a Dijkstra pass on the graph limited to positive edges; then (2) a second step that enforces all after-conditions, represented by negative edges, that were ignored in the first step. Those two steps are complemented with a pruning mechanism that repeatedly detects nodes in positive cycles.

Algorithm 1 begins by selecting a set of *assumed reachable* nodes from which to start the propagation process (lines 2-10). The obvious candidates are fluents appearing in the initial state I and in timed initial literals T. We also optimistically select all actions that have no *before-conditions*, i.e., actions where every condition is an after-condition. Assumed reachable nodes are inserted into a priority queue Q of $\langle n, t \rangle$ pairs where n is a node of the reachability graph and t is a candidate time for its earliest appearance.

We then iteratively extend the initial set of assumed reachable nodes with all fluents that have an assumed reachable achiever and all actions whose every before-condition is assumed reachable. This is done by a Dijkstra-like propagation (line 13), that extracts the nodes in Q in the order of increasing appearance time. Those extracted nodes are marked as reachable and their successors are inserted into Q. The algorithm slightly differs from Dijkstra's as it ensures that an action node is enqueued only if all of its before-conditions have been already marked reachable (lines 36-38).

As a second step, we revise our optimistic assumptions by incorporating the ignored after-conditions:

- Line 16 removes from the graph any action a with an after-condition on an unreachable fluent f. More specifically, the RECURSIVELYREMOVE procedure marks its parameter as unreachable and removes it from the graph. This removal process is recursive: if a removed action a is the only achiever for a fluent f then f is removed as well (and as a consequence all actions depending on f will also be removed). Furthermore, if the first achiever of a fluent is removed from the graph and there is at least one other achiever for it, then the fluent is added back to Q with an updated earliest appearance.
- Line 18 takes an after-condition of an action a on a reachable fluent f and enforces the minimal delay δ between $ea(f)$ and $ea(a)$. If the current delay is not sufficient, a is added to Q and will be reconsidered upon the next Dijkstra pass.

Finally, *late nodes* are marked unreachable and removed from the graph (line 21). We say that a node n is *late* if for any *non-late* node n', $ea(n') + d_{max} < ea(n)$ where d_{max} is the highest delay on any edge of the graph. In practice, this means that nodes are partitioned into non-late nodes and late nodes, these two sets being separated by a temporal gap of at least d_{max}. The intuition, as demonstrated in the next section, is that the earliest appearance of a late node is being pushed back due to unachievable cycles.

Algorithm 1 Algorithm for identifying reachable nodes in a reachability graph and computing their earliest appearance.

```
1:  ⟨N, E⟩ ← Reachability Graph
2:  Q ← ∅      ▷ Priority queue of ⟨node, time⟩ ordered by increasing time
3:  for all n ∈ N do
4:      reachable(n) ← F
5:      if n is an action with no before-conditions then
6:          Q ← Q ∪ {⟨n, 0⟩}
7:  for all ⟨[t] f := T⟩ ∈ T do          ▷ timed initial literals
8:      Q ← Q ∪ {⟨f, t⟩}
9:  for all f := T ∈ I do                 ▷ initial state
10:     Q ← Q ∪ {⟨f, 0⟩}
11:
12: while Q non empty do
13:     DIJKSTRAPASS
14:     for all f --δ--> a ∈ after-condition edges do
15:         if ¬reachable(f) then
16:             ⟨N, E⟩ ← RECURSIVELYREMOVE(a)
17:         else if ea(a) < ea(f) + δ then
18:             Q ← Q ∪ {⟨a, ea(f) + δ⟩}
19:     for n ∈ N do
20:         if n is late then
21:             ⟨N, E⟩ ← RECURSIVELYREMOVE(n)
22:
23: procedure DIJKSTRAPASS
24:     while Q non empty do
25:         ⟨n, t⟩ ← pop(Q)
26:         if n already expanded in this pass then
27:             continue
28:         if reachable(n) ∧ ea(n) ≥ t then
29:             continue
30:         reachable(n) ← T
31:         ea(n) ← t
32:         if n is an action with an effect edge n --δ--> f then
33:             Q ← {⟨f, t + δ⟩}
34:         else
35:             for all a conditioned on n do
36:                 if all before cond. of a are reach. then
37:                     t' ← max_{f --δ--> a ∈ E}  ea(f) + δ
38:                     Q ← Q ∪ {⟨a, t'⟩}
```

The two-step process is repeated to take into account the newly updated reachability information. In the subsequent runs, the Dijkstra algorithm will start propagating the updated nodes from the previous run, with lines 28-29 making sure that the earliest appearance values $ea(n)$ are never decreased to an overly optimistic value. The algorithm detects a fix-point and exits if the queue is empty, meaning that after-conditions did not trigger any change.

3.4 Analysis and Related Models

We now explore some of the characteristics of Algorithm 1. The first Dijkstra pass acts as an optimistic initialization: it identifies a set of possibly reachable nodes and assigns them earliest appearance times. All operations after this first pass will only *(i)* shrink the set of reachable nodes; and *(ii)* increase the earliest appearance times.

Proposition 3.1. *If a node n is reachable, then $ea(n)$ converges towards $ea^*(n)$. If a node n' is not reachable then $ea(n')$ either remains at ∞ or diverges towards ∞ until it is removed from the graph.*

Proof (sketch). A node n is reachable if there is either a path from initial facts to n or n is part of a self-supporting causal loop (i.e. cycle of negative or zero length). Consequently repeated propagations will eventually converge. On the other hand, an unreachable node either depends on an unreachable node or is involved only in causal cycles of strictly positive length (such as the one depicted in Figure 3a). If the node was ever assumed reachable, its earliest appearance will thus be increased until it is removed from the graph. □

Proposition 3.2. *If a node is late, then it is not reachable.*

Proof (Sketch). The intuition is that the gap between non-late and late nodes appeared because late nodes are delaying each other due to positive causal cycles. We first show that any late node delayed to its current time is due to a dependency on another late node: because the temporal gap is bigger than all edges in the graph, a non-late node could not have influenced a late node. It follows that any late node depends on at least one other late node. Furthermore a late node necessarily participates in a positive cycle or depends on a late node that does. From there, one can show that at least one node n in this group is involved only in positive cycles. Any other possibility (path from timed initial literals or negative cycle) would have resulted in n being less than d_{max} away from a non-late node. □

It follows from propositions 3.1 and 3.2 that Algorithm 1 produces a reachability model R_∞ that contains a node n and its earliest appearance $ea^*(n)$ iff n is reachable in the relaxed problem. In the worst case, computing this model has a pseudo-polynomial complexity since there may be as many as d_{max} iterations of the algorithm (d_{max} being the highest delay in the graph). The cost of each iteration is dominated by the Dijkstra pass of $O(|N| \times \log(|N|) + |E|)$.

Discussion: One might consider computing various approximations of R_∞ by limiting the number of iterations to a fixed number K, making the algorithm strongly polynomial and producing a reachability model R_K. In the special case where $K = 1$, this is equivalent to performing a single Dijkstra pass and removing all actions with an unreachable after-condition. Increasing K would allow us to better estimate the earliest appearances and detect additional late nodes.

Another simplification is to ignore all negative edges of the reachability graph, which can be done by stopping Algorithm 1 after the first Dijkstra pass. In practice, this model simply ignores after-conditions and it has all the characteristics of the temporal planning graph of POPF: (1) the separation of durative actions into at-start and at-end instantaneous actions is done by the transformation into elementary actions; (2) the minimal delay between matching at-start and at-end actions is enforced by the presence of start conditions in the elementary actions representing the end effects; and (3) any end condition appearing in the elementary action of a start effect would be ignored because it would be an after-condition. Since it is a direct adaptation of the techniques used in POPF to our more complex action representation, we call this model R^{popf}.

It is interesting to note that R^{popf} and R_∞ are equivalent on all problems with no after-conditions. Classical planning obviously falls in this category as well as any PDDL model with no at-start effect or no at-end condition. In fact, on such problems R^{popf} and R_∞ are equivalent to building a temporal planning graph, with no significant computational overhead.

4 Extending to Hierarchical Models

While hierarchical planning is a dominant approach for modeling and solving real-world planning applications, it still mostly requires manual work to model and control its search space. In planners such as SHOP2 (Nau et al. 2003), this is done by manually annotating methods with additional preconditions that are checked before introducing a method into the plan. This approach allows for early dead-end detection and has proven to be extremely efficient for solving complex problems. Nonetheless, it does have important drawbacks. First, manual annotation requires significant domain modeling efforts and can easily lead to modeling errors and incomplete domain descriptions. Second, conditions on methods can only be efficiently tested on fully defined states. For that reason, HTN planners usually restrict themselves to finding totally-ordered plans, which do not work for planning problems with required concurrency. Reachability analysis thus constitutes a critical step to improve the performance of partially-ordered hierarchical planners, which can find plans with required concurrency, and also reduce the laborious domain engineering effort for HTN planners that only find totally-ordered plans.

The main difficulty of automated reachability analysis for hierarchical problems lies in the interactions between causal and hierarchical constraints. HTN planning has three main characteristics: (1) a method must eventually have all its subtasks achieved; (2) a method or operator can only appear in a plan if it is refining an existing task of the plan; and (3) all conditions of operators and methods must be causally supported by earlier effects. While there are known tech-

niques to check (3), integrating (1) and (2) causes interdependent actions and remains a difficult issue.

In this section, we propose a transformation of operators and methods to expose those hierarchical constraints as additional conditions and effects. This allows us to perform the reachability analysis proposed in the previous section on models integrating hierarchical and causal information.

4.1 Hierarchical Model

We extend the temporal planning model from Section 2.1 to support the definition of hierarchical problems. A temporal planning problem P is extended to contain:

- a set $\mathcal{T}$ of task symbols
- a set of goal tasks $G_{\mathcal{T}}$. A goal task $g_\tau \in G_{\mathcal{T}}$ is of the form $\langle [st_\tau, et_\tau]\ \tau \rangle$ where st_τ and et_τ are timepoints taking value in R^+ and $\tau \in \mathcal{T}$ is a task symbol. The goal task g_τ states that the plan should contain an action achieving the task τ and spanning the temporal interval $[st_\tau, et_\tau]$.

Each action $a \in A$ is associated to a task symbol $\tau_a \in \mathcal{T}$, representing the task achieved by a and a set of subtasks S_a. A subtask is denoted by $\langle [st_\tau, et_\tau]\ \tau \rangle$ where τ is a task symbol and st_τ and et_τ are timepoints. The intuition is that for each subtask $\langle [st_\tau, et_\tau]\ \tau \rangle \in S_a$, a requires an action achieving τ and executing over the interval $[st_\tau, et_\tau]$. This model does not make any distinction between methods and operators as it is usually done in HTN planning. To make this distinction, one could simply partition A into actions with no effects (i.e. methods) and actions with no subtasks (i.e. operators).

In addition to the requirements of Section 2.1, we consider the following conditions for a plan π to be valid:

- for any task $g_\tau = \langle [st_\tau, et_\tau]\ \tau \rangle$ appearing in $G_{\mathcal{T}}$ or as a subtask of an action in π, there is an action in π achieving g_τ. An action $\langle t_a, a, d_a \rangle \in \pi$ is said to achieve g_τ if $\tau = \tau_a$, $st_\tau = t_a$ and $et_\tau = t_a + d_a$.
- for any action $\langle t_a, a, d_a \rangle \in \pi$, there is a task g_τ appearing in $G_{\mathcal{T}}$ or as a subtasks of an action in π such that a achieves g_τ.

The latter condition is a consequence of the search mechanism of HTN planners in which every action is introduced to fulfill a given pending task. When combined with the former, it results in interdependencies as a method both requires the presence of actions fulfilling its subtasks and enables their presence.

4.2 Flattening Transformation

We now propose a compilation of a hierarchical problem into the temporal model of Section 2.1. This flattening procedure is meant to allow a reachability analysis on causal models that retain the hierarchical constraints from the original problem.

For each task $\tau \in \mathcal{T}$ from the hierarchical model, the set of propositions V is extended with three new propositions $started(\tau)$, $ended(\tau)$ and $required(\tau)$. They respectively represent that an action achieving τ starts, finishes or is required to fulfill a pending task τ.

put-on-table-from-stack(x)

task:	put-on-table(x)
conditions:	$\varnothing$
effects:	$\varnothing$
subtasks:	$[d_1, d_2]$ unstack(x)
	$[d_3, d_4]$ put-down(x)
constraints:	$0 < d_1 < d_2 < d_3 < d_4 < d$

(a) A high level action (or method) to move a block x from the top of a stack to the table.

put-on-table-from-stack(x) (flattened model)

conditions:	$[0]\ required(\text{put-on-table(x)})$
	$[d_1]\ started(\text{unstack(x)})$
	$[d_2]\ ended(\text{unstack(x)})$
	$[d_3]\ started(\text{put-down(x)})$
	$[d_4]\ ended(\text{put-down(x)})$
effects:	$[0]\ started(\text{put-on-table(x)})$
	$[d_1]\ required(\text{unstack(x)})$
	$[d_3]\ required(\text{put-down(x)})$
	$[d]\ ended(\text{put-on-table(x)})$
constraints:	$0 < d_1 < d_2 < d_3 < d_4 < d$

(b) The flattened action after compiling away its hierarchical properties.

Figure 4

A hierarchical action $a \in A$ is transformed into a 'flat' action a_{flat} with:

- all conditions, effects and constraints of a
- one additional condition $\langle [0]\ required(\tau_a) = \mathtt{T} \rangle$
- one additional at-start effect $\langle [0]\ started(\tau_a) := \mathtt{T} \rangle$ and one additional at-end effect $\langle [d_a]\ ended(\tau_a) := \mathtt{T} \rangle$
- for each subtask $\langle [st_\tau, et_\tau]\ \tau \rangle$ of a:
  - two additional conditions $\langle [st_\tau]\ started(\tau) = \mathtt{T} \rangle$ and $\langle [et_\tau]\ ended(\tau) = \mathtt{T} \rangle$
  - one additional effect $\langle [st_\tau]\ required(\tau) := \mathtt{T} \rangle$

For each goal task $\langle [st_\tau, et_\tau]\ \tau \rangle \in G_{\mathcal{T}}$, a timed initial literal $\langle [st_\tau]\ required(\tau) := \mathtt{T} \rangle$ is added to T and a goal $ended(\tau)$ is added to G.

It is important to note that the resulting 'flat' problem is a relaxation of the original one. Indeed, a given subtask $\langle [st_\tau, et_\tau]\ \tau \rangle$ yields two conditions $\langle [st_\tau]\ started(\tau) = \mathtt{T} \rangle$ and $\langle [et_\tau]\ ended(\tau) = \mathtt{T} \rangle$. Those two conditions can be fulfilled by distinct actions, thus ignoring temporal constraints on the unique action that should have achieved the subtask in the original model. This relaxed transformation is simply meant to expose hierarchical features of the problem to reachability analysis. Actions resulting from this compilation step can be split into elementary actions and added to a reachability graph (Section 3).

An example of this transformation is given in Figure 4b. The problem has interdependencies as the presence of the *put-on-table-from-stack* method both requires and allows

the presence of its *unstack* and *put-down* subactions. Indeed, an *unstack(x)* action would have a start condition $\langle [0]\ required(\text{unstack(x)}) = \texttt{T} \rangle$ which is achieved by the method *put-on-table-from-stack*. Concurrently, this method has the condition $\langle [d_1]\ started(\text{unstack(x)}) = \texttt{T} \rangle$ which would be achieved as a start effect of the *unstack(x)* action.

5 Empirical Evaluation

We implemented our reachability analysis technique within FAPE, a partial-order temporal planner (Dvorak et al. 2014a). FAPE takes problems modeled in ANML (Smith, Frank, and Cushing 2008). ANML natively supports: (1) conditions and effects at arbitrary time points and over arbitrary intervals within an action; and (2) hierarchical structures. FAPE supports most of the features of ANML and is capable of both hierarchical and generative planning.

Like other partial-order planners, FAPE searches for a plan by fixing flaws in partial plans until no flaws remain. Every time a partial plan p is extracted from the open queue, reachability analysis is performed and provides an updated set of impossible actions and fluents. If it can be verified that from p: (1) all goals are reachable, and (2) all unrefined tasks have a possible refinement, then we expand p by fixing one of its flaws. If this is not the case, p is a dead-end and is discarded. Reachability results are also used to filter out flaw resolvers involving impossible actions or fluents.

We evaluate our reachability analysis technique on several temporal domains with and without hierarchical decomposition, the former involving many instances of required concurrency and interdependent actions. The *satellite*, *rovers*, *tms*, *logisitcs* and *hiking* domains are direct translations of the eponymous domains from the International Planning Competition (IPC) into ANML. The domain files were manually translated while the translation of problem instances was automated. The *handover* domain is a robotics problem presented in (Dvorak et al. 2014b) and the *docks* domain is the dock worker domain from (Ghallab, Nau, and Traverso 2004). Hierarchical versions of the domains have their names appended with '-hier'. All experiments were conducted on an Intel Xeon E3 with 3GB of memory and a 30 minutes timeout.

Table 1 and Figure 5 present the number of problems solved using different reachability models. R_∞ outperforms the other configurations: solving the highest number of problems on all but one domain. R_5 and R_1 are respectively second and third best performers while R^{popf} does not provide significant pruning of the search space; the computational overhead makes it perform slightly worse than no reachability checks (denoted by $\varnothing$). As expected, on temporally simple problems (non-hierarchical domains in our test set), all configurations show similar performance.

Table 2 presents the percentage of actions detected as unreachable by different configurations. As expected, R_∞, R_5, R_1 and R^{popf} perform identically on temporally simple problems. However, R^{popf} is largely outperformed on all but one hierarchical domains.The good performance of R_1 with respect to R^{popf} shows that a single iteration is often sufficient to capture most of the problematic after-conditions. However, on more complex problems such as *hiking-hier* and *docks-hier*, more iterations are beneficial both in terms of detected unreachable actions and solved problems.

	R_∞	R_5	R_1	R^{popf}	$\varnothing$
satellite (20)	14	14	14	14	**15**
satellite-hier (20)	**17**	**17**	**17**	**17**	16
rovers (40)	**25**	**25**	**25**	**25**	**25**
rovers-hier (40)	**22**	**22**	**22**	**22**	**22**
tms-hier (20)	**7**	**7**	**7**	**7**	**7**
logistics (28)	**8**	**8**	**8**	**8**	**8**
logistics-hier (28)	**28**	**28**	**28**	6	9
hiking-hier (20)	**20**	17	16	15	17
handover-hier (20)	**16**	**16**	**16**	7	7
docks-hier (18)	**17**	13	12	7	7
Total (254)	**174**	167	165	128	133

Table 1: Number of solved tasks for various domains with a 30 minutes timeout. The best result is shown in bold. The number of problem instances is given in parenthesis.

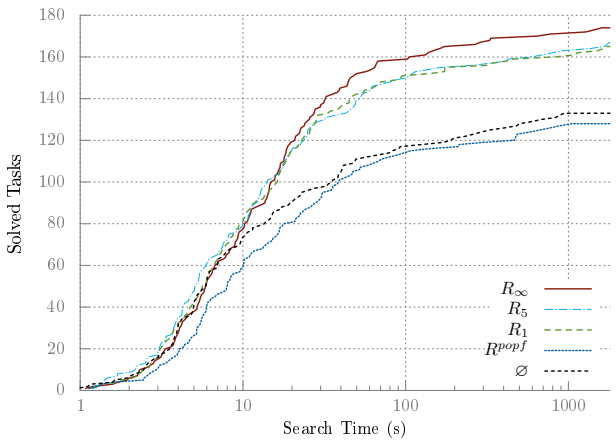


Figure 5: Number of solved tasks by each configuration within a given time amount.

	R_∞	R_5	R_1	R^{popf}	$\varnothing$
satellite	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
satellite-hier	**14.1**	**14.1**	**14.1**	**14.1**	0.0
rovers	**43.5**	**43.5**	**43.5**	**43.5**	0.0
rovers-hier	**72.6**	**72.6**	**72.6**	27.1	0.0
tms-hier	**87.9**	**87.9**	**87.9**	0.0	0.0
logistics	**34.5**	**34.5**	**34.5**	**34.5**	0.0
logistics-hier	**94.6**	**94.6**	**94.6**	15.5	0.0
hiking-hier	**38.1**	36.5	36.5	0.0	0.0
handover-hier	**99.2**	**99.2**	**99.2**	3.5	0.0
docks-hier	**85.2**	52.6	52.6	0.0	0.0

Table 2: Percentage of ground actions detected as unreachable from the initial state. For each problem instance, the percentage is obtained by comparing the number of ground actions detected as unreachable from the initial state with the original number of ground actions. Those values are then averaged over all instances of a domain.

	R_∞	R_5	R_1	R^{popf}	$\varnothing$
satellite	**100** (1)	**100**	**100**	**100**	–
satellite-hier	**100** (2)	**100**	**100**	**100**	–
rovers	**100** (1)	**100**	**100**	**100**	–
rovers-hier	**100** (4)	99.2	56.7	54.2	–
tms-hier	**100** (6)	69.3	14.2	14.2	–
logistics	**100** (1)	**100**	**100**	**100**	–
logistics-hier	**100** (2)	**100**	2.8	2.8	–
hiking-hier	**100** (9)	**100**	71.7	71.7	–
handover-hier	**100** (43)	98.2	5.7	5.7	–
docks-hier	**100** (37)	73.0	29.8	29.8	–

Table 3: Average admissible makespans for different reachability models. Those are computed by taking the earliest appearance of the latest satisfied goal from the initial state, and normalizing on the value computed for R_∞. For R_∞, we also indicate the average number of iterations needed to converge on the first propagation of each instance (in parenthesis).

Table 3 presents the value that would have been taken by the admissible h^{max} heuristic with different reachability models. On all but one hierarchical model, both R_1 and R^{popf} largely underestimate the makespan of a solution. Indeed, not propagating after-conditions makes them miss important causal aspect of the problems. Those can take as much as 43 iterations to be initially propagated by R_∞. The subsequent propagations are typically faster because they are made incrementally. As expected, a single iteration was needed to converge on all temporally simple problems.

Note that the current usage of our approach in FAPE is limited to restricting the search space. While it proves extremely useful on a wide variety of problems, one could contemplate using the available data structures to extract a heuristic value. The extraction of a relaxed plan has proven to be an effective heuristic in many planners. Earliest possible times $ea^*(a)$ and $ea^*(f)$ could also be used as admissible estimates when considering makespan optimization. However, FAPE's constraint-based algorithm, with POCL and lifted representation, is not yet suitable for this.

6 Related Work

The problem of required concurrency in temporal planning has been analyzed by Cushing et al. (2007). The authors distinguish temporally expressive problems that feature required concurrency from temporally simple problems that do not. Temporally expressive problems are further studied by Cooper, Maris, and Régnier (2013) who identify temporally-cyclic problems in which sets of actions can be interdependent. While the 7th and 8th International Planning Competitions (IPC) included problems with required concurrency, none of those had interdependent actions. In fact, even top performers in the temporal track of the IPC, including Temporal Fast Downward (Eyerich, Mattmüller, and Röger 2012) and YAHSP3 (Vidal 2014), cannot solve problems with interdependent actions.

In classical planners such as FF (Hoffmann and Nebel 2001), the most widely used reachability analysis involves building a Relaxed Planning Graph (RPG) from delete-free actions. CRIKEY3 (Coles et al. 2008), extended this technique to support temporal problems with interdependencies by splitting durative actions into at-start and at-end snap actions. The resulting Temporal RPG is used by CRIKEY3 and its successors POPF (Coles et al. 2010), COLIN (Coles et al. 2012) and OPTIC (Benton, Coles, and Coles 2012) both for reachability analysis and heuristic computation.

Cooper, Maris, and Régnier (2014) discuss another relaxation of temporal planning problems into monotone problems that can be solved in polynomial time. This relaxation is orthogonal to the delete-free relaxation and could also be used for reachability analysis. A key part of this relaxation is the removal of any condition that might be achieved by more than one action; this is likely to lead to poor performance on HTN planning problems where the difficulty is precisely to choose which action to support a given task.

HTN planning systems in the line of SHOP2 (Nau et al. 2003), avoid the need for reachability analysis by (1) manually annotating methods with conditions of applicability; and (2) requiring a total order between all operators, to ensure a method's conditions can be tested on fully defined states. While this technique has proven to be extremely useful on many practical problems, it increases the required domain-engineering effort as well as the risk of introducing modeling errors. Even temporal HTN planners such as SIADEX (Castillo et al. 2006) only partially remove the need for total-order between operators and cannot solve problems with required concurrency.

The recent development of hybrid hierarchical and generative planners such as PANDA (Schattenberg 2009) and FAPE (Dvorak et al. 2014a) has motivated the need for automated search guidance for hierarchical problems. Along these lines, Bercher, Keen, and Biundo (2014) and Elkawkagy et al. (2012) proposed techniques for evaluating the remaining search effort in hierarchical problems by exploiting landmarks and task decomposition graphs. However, hierarchical and causal constraints are still mainly considered independently, resulting in limited heuristic guidance.

Our work shares some conceptual similarities with Angelic Hierarchical Planning (Marthi, Russell, and Wolfe 2007), which performs automated analysis of sequential hierarchical problems in order to infer upper and lower bounds of the set of reachable states. Those sets are used in a hierarchical planner to detect when a task network is always refinable to a solution plan or when it is a dead-end. Those techniques are however only applicable to sequential hierarchical planning. While our technique only focuses on dead-end detection, we consider more general hierarchical problems, featuring concurrency and partial-ordering.

A translation of some hierarchical features of ANML into PDDL was proposed by Smith, Frank, and Cushing (2008) and a complete translation of a restricted class of HTN problems into PDDL was proposed by Alford, Kuter, and Nau (2009). Unlike the exact translations described in those works, we introduce a relaxed translation applicable to any HTN problem for the purpose of reachability analysis. This simpler transformation allows us to avoid the discovery by

Alford et al. (2014) that heuristics based on delete-free relaxation require further relaxation to allow tractability when dealing with hierarchical problems.

7 Conclusion

In this paper, we developed a technique to perform more accurate reachability analysis for temporal planning problems involving interdependent actions. This technique allows us to do a better job of recognizing impossible actions and estimating the earliest start times for actions and fluents. We also showed how interdependencies naturally arise in hierarchical planning problems and introduced a simple relaxation of those problems into temporal planning to enable reachability analysis on hierarchical planning problems.

Our method has been implemented in FAPE, a constraint-based temporal planner for the ANML language. We evaluated the effectiveness of the technique for pruning the search space in both hierarchical and generative planning problems. When compared to state of the art techniques, we showed that our algorithm provides notable improvements on temporally complex problems while having no computational overhead on temporally simple ones. This characteristic makes our method suitable for reachability analysis in a wide range of temporal as well as hierarchical problems.

Acknowledgements. We would like to thank Malik Ghallab and Félix Ingrand for their valuable comments on early versions of this paper. This work was supported in part by the EDSYS Doctoral School of the University of Toulouse, Stinger Ghaffarian Technologies (SGT) Incorporated, the EU MUMMER project funded by the H2020 program under grant agreement No 688147, the NASA Safe Autonomous Systems Operations (SASO) project, and the NASA Autonomous Systems and Operations Project.

References

Alford, R.; Shivashankar, V.; Kuter, U.; and Nau, D. S. 2014. On the Feasibility of Planning Graph Style Heuristics for HTN Planning. In *Proc. of the 24th Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

Alford, R.; Kuter, U.; and Nau, D. S. 2009. Translating HTNs to PDDL: A Small Amount of Domain Knowledge Can Go a Long Way. In *Proc. of the 21st Int. Joint Conf. on Artificial Intelligence (IJCAI)*.

Benton, J.; Coles, A.; and Coles, A. 2012. Temporal Planning with Preferences and Time-Dependent Continuous Costs. *Proc. of the 22th Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

Bercher, P.; Keen, S.; and Biundo, S. 2014. Hybrid Planning Heuristics Based on Task Decomposition Graphs. In *Proc. of the Seventh Annual Symp. on Combinatorial Search (SOCS)*.

Castillo, L. A.; Fernández-Olivares, J.; García-Pérez, Ó.; and Palao, F. 2006. Efficiently Handling Temporal Knowledge in an HTN Planner. In *Proc. of the 16th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 63–72.

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. Planning with Problems Requiring Temporal Coordination. *Proc. of the 22th AAAI Conf. on Artificial Intelligence*.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. In *Proc. of the 20th Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 42–49.

Coles, A. J.; Coles, A.; Fox, M.; and Long, D. 2012. COLIN: Planning with Continuous Linear Numeric Change. *Journal of Artificial Intelligence Research (JAIR)* 44:1–96.

Cooper, M.; Maris, F.; and Régnier, P. 2013. Managing Temporal Cycles in Planning Problems Requiring Concurrency. *Computational Intelligence* 29(1):111–128.

Cooper, M.; Maris, F.; and Régnier, P. 2014. Monotone temporal planning: Tractability, extensions and applications. *Journal of Artificial Intelligence Research* 50:447–485.

Cushing, W.; Kambhampati, S.; Weld, D. S.; et al. 2007. When is temporal planning really temporal? In *Proc. of the 20th Int. Joint Conf. on Artifical Intelligence*, 1852–1859.

Dvorak, F.; Barták, R.; Bit-Monnot, A.; Ingrand, F.; and Ghallab, M. 2014a. Planning and Acting with Temporal and Hierarchical Decomposition Models. In *Proc. of the 26th IEEE Int. Conf. on Tools with Artificial Intelligence, ICTAI*, 115–121.

Dvorak, F.; Bit-Monnot, A.; Ingrand, F.; and Ghallab, M. 2014b. A Flexible ANML Actor and Planner in Robotics. In *Planning and Robotics (PlanRob) Workshop (ICAPS)*.

Elkawkagy, M.; Bercher, P.; Schattenberg, B.; and Biundo, S. 2012. Improving Hierarchical Planning Performance by the Use of Landmarks. In *Proc. of the 26th AAAI Conf. on Artificial Intelligence*.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2012. Using the context-enhanced additive heuristic for temporal and numeric planning. *Springer Tracts in Advanced Robotics*.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory & Practice*. Elsevier.

Hoffmann, J., and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research (JAIR)* 253–302.

Marthi, B.; Russell, S.; and Wolfe, J. 2007. Angelic Semantics for High-Level Actions. In *Proc. of the 17th Int. Conf. on Automated Planning and Scheduling (ICAPS)*.

Nau, D.; Au, T.-c.; Ilghami, O.; Kuter, U.; Murdock, J. W.; Wu, D.; and Yaman, F. 2003. SHOP2: An HTN Planning System. *Journal of Artificial Intelligence Research (JAIR)* 20:379–404.

Schattenberg, B. 2009. *Hybrid planning & scheduling*. Ph.D. Dissertation, University of Ulm.

Smith, D. E.; Frank, J.; and Cushing, W. 2008. The ANML language. In *The ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*.

Vidal, V. 2014. YAHSP3 and YAHSP3-MT in the 8th International Planning Competition. In *8th International Planning Competition (IPC-2014)*.