Extracting landmarks in temporal domains

Laura Sebastia, Eliseo Marzal and Eva Onaindia
Technical University of Valencia
Camino de Vera s/n - 46022 - Valencia Tel. +34963877000 - Fax. +34963877359 Spain
Email: lstarin, emarzal, onaindia@dsic.upv.es

CONFERENCE: ICAI - International Conference on Artificial

Keywords: planning, temporal, landmark

Author presenting the paper: Laura Sebastia
Abstract—In this paper, we present a new technique to extract landmarks in a temporal context. This technique works iteratively over a temporal landmarks graph in order to obtain the skeleton of the final plan.

Our approach consists in four stages: first, a temporal planning graph is built; then, a set of initial temporal landmarks is computed from this temporal planning graph and we build the initial temporal landmarks graph; the next step activates each landmark in the most appropriate time point by taking into account mutex relationships and dependencies; the final step computes necessary and reasonable orders between landmarks which enrich the skeleton of the plan.

The results will show that our technique is very interesting because it is able to obtain a temporal landmarks graph whose structure is very similar to the final plan.

I. INTRODUCTION

Research in AI planning is more and more concerned with the introduction of more expressive languages and development of new techniques to deal with more realistic problems. A crucial element in this approach to reality is time. In the last years, several extensions in the standard language PDDL have represented a step forward in the specification of temporal planning problems, as the introduction of durative actions in PDDL2.1[1] or timed initial literals in the most recent PDDL2.2[2].

Researchers have made a great effort to extend and improve the capabilities of temporal planners. The last decade has seen many new works: ZENO [3] and IxTeT [4], which cope with temporality on actions and temporal constraints, etc. Graphplan success has allowed the development of more modern temporal planners: TGP [5], whose ideas are very valuable in temporal environments, TP4 [6] and Sapa [7], which handle concurrent durative actions and use heuristic metrics to deal with resources in planning. TPSYS [8], although inspired by the ideas of TPG, handles a richer model of actions and uses heuristic techniques to improve the search process. LPG [9], Mips [10] and SGPlan [11] have showed an outstanding performance in different planning competitions, demonstrating that heuristic and local search are very useful in planning and especially in temporal planning.

On the other hand, the extraction and analysis of landmarks have proved to be a useful approach to solve classical planning problems. A landmark in a STRIPS setting is defined as a literal that must be achieved in any solution plan. In general, these landmarks are studied and used to decompose the planning problem in smaller subproblems or, simply, as a heuristic to guide the search process, as we will explain later on.

Our aim is to extend the concept of landmark to the temporal context, in order to build an skeleton of the solution plan. This skeleton will permit us obtain a short plan, that is, our goal is to incorporate to the landmark definition those literals that will lead to the shortest plans. This paper introduces an approach to extract this set of landmarks. It also compares the structure of the skeleton that we obtain with the structure of the optimal plan for each problem. As we will show, our method is capable of obtaining a skeleton that, after adding the corresponding actions, will correspond to the optimal plan.

Section II formally defines what a temporal problem is. Section III summarizes previous approaches used to extract landmarks in a STRIPS setting. Section IV introduces the main aspects of our approach, which are detailed in sections V to VIII. Section IX shows the experiments we have performed in order to test our technique and, finally, section X draws some conclusions and further work.

II. TEMPORAL PROBLEM

The components of a durative action a are the following:

- **Conditions.** The two types of conditions of a durative action are $S\text{Cond}(a)$, the set of conditions to be guaranteed at the start of the action and $Inv(a)$, the set of invariant conditions to be guaranteed over the execution of the action $^1$.
- **Duration.** The duration of the action is a positive value represented by $dur(a) \in \mathbb{R}^+$.
- **Effects.** The two types of effects of a durative action are $SEff(a) = \{SAdd(a) \cup SDel(a)\}$, with the positive and negative effects respectively to be asserted at the start of the action and $EEff(a) = \{EAdd(a) \cup EDel(a)\}$, with the positive and negative effects respectively to be asserted at the end of the action.

The following is an example of a durative action in the **driverlog** domain [12]:

```plaintext
{ (:durative-action LOAD
  :parameters
  (?obj - obj
  ?truck - truck
  ?loc - location)
  :duration (= ?duration 2)
  :condition
    (and
      (over all (at ?truck ?loc))
      (at start (at ?obj ?loc)))
  :effect
    (and
      (at start (not (at ?obj ?loc)))
      (at end (in ?obj ?truck)))
)
```

A temporal planning problem $P = \langle A, I, G \rangle$ is a triple where $A$ is the set of ground durative actions that can be applied in the domain and $I$ (initial state) and $G$ (goal state) are sets of facts.

Along this paper, we will use an example of the **driverlog** domain to illustrate our approach. The corresponding initial state and goal state are shown in Table I. In this example, we find two drivers **driver1** and **driver2** who can drive two trucks **truck1** and **truck2**. Both **driver1** and **truck1** are initially at **s1**, whereas **driver2** and **truck2** are at **s0** and **s2**, respectively. Moreover, we have two packages **package1** and **package3** that must be transported to **s1** and to **s2**, respectively, by means of these trucks and drivers.

$^1$In PDDL 2.1, we can also find another type of conditions: $ECon\text{ond}(a)$, the set of conditions to be guaranteed at the end of the action, but we do not consider them in our approach.
III. PREVIOUS WORK BASED ON EXTRACTION OF LANDMARKS

As we said before, a landmark is defined as a literal that must be achieved in any solution plan. Landmarks have been successfully used to plan in a STRIPS setting. Some approaches use them as another source of information to take into account in the search process. For example, GRT [13] is a planner that, besides using a heuristic to improve the search process, uses a subgoal (landmarks) extraction to decompose the whole problem in smaller ordered subproblems, which are easier to solve. A similar technique is used by SGPlan [11], which partitions a planning problem into subproblems, orders these subproblems according to a sequential resolution of their subgoals, finds a feasible plan for each goal and combines the solutions to obtain the final plan. Zhu and Givan [14] have developed a technique which uses landmarks as a source for computing a heuristic to guide the plan search.

The resolution technique of some other approaches is completely dependent on the extraction of landmarks. For example, Porteous, Hoffmann and Sebastia [15] build a landmarks graph (LG) which is used to obtain a set of subproblems, each of them resulting from considering the leaf nodes of the current LG, and when a subtask has been processed, the LG is updated by removing the achieved leaf nodes. The solution plan is formed by concatenating the subplans obtained from each subproblem. On the other hand, Sebastia, Onaindia and Marzal [16] use the LG to perform a chronological decomposition and obtains a sequence of subproblems that can be solved concurrently, thus obtaining important time savings.

In the development of our approach, we are more interested on how landmarks can be extracted rather than on how to use them in the search process. We can find two different approaches: one based on the planning graph (PG) [17] and the other based on the relaxed planning graph (RPG) [18].

The technique of landmarks extraction developed in [14] is based on the PG. Two types of landmarks are distinguished: proposition and action landmarks, which are literals and actions (respectively) that must occur in any correct plan. This approach performs a propagation process over the PG. The propagated labels are actions or propositions. Action or proposition \(N\) is labeled with label \(L_i\) at level \(i\) in the PG to represent the claim that any \(i\)-step parallel plan achieving \(N\) must contain an occurrence of \(L_i\). In the initial graph level, which is an action level, every action is labeled with itself and no other labels. Each proposition node in the graph is labeled with the intersection of the labels on its predecessor action nodes (because a landmark label applies to a proposition only if it applies to all actions that add that proposition). Each action node in the graph, after the first level, is labeled with the union of the labels on its predecessor proposition nodes (because a landmark label applies to an action if it applies to any of its predecessors). When the propagation is complete, any label on a goal proposition in the final level is a landmark for plans achieving the goal.

On the other hand, the landmarks extraction based on the RPG [15] proceeds in two steps:

1) First, landmark candidates are computed by means of a backwards search on the RPG. The process starts by adding all top level goals to the set of landmarks candidates (LC) and they are also posted as goals in the first level where they appeared in the RPG. Then, each goal is solved in the RPG starting from the last level. For each goal \(g\) in a level, all actions achieving \(g\) are grouped into a set and the intersection \(I\) of their preconditions is computed. For all facts \(p\) in \(I\), \(p\) is posted as a (sub-)goal in the first RPG level where it is achieved and \(p\) is inserted into the LC. When all goals in a level are achieved, the next lower level is processed. The process stops when the first (initial) level is reached.

2) Once the landmark candidates have been extracted, each of them is checked to be a provably landmark by means of the following method. A new planning problem is constructed where the actions adding the literal \(I\) to be checked are eliminated. Then, a RPG is built; if the goals are reached, the literal is discarded as a landmark.

These landmarks are then ordered according to three types of orders: necessary, reasonable and obedient orders. As a result, a landmark generation graph is obtained.

IV. OVERVIEW OF OUR APPROACH

Our goal is to build a skeleton of the solution plan by means of the extraction of a set of landmarks in a temporal context. But, what type of literals can be considered as landmarks in this context?

Landmarks have been used in a STRIPS context to help in obtaining a good solution of the planning problem thanks to the construction of the landmarks graph. This involves additional reasoning about the convenience of achieving the literals in a certain order, which gives the planner very useful information when building the plan.

When we move to the temporal setting, time plays a central role and, therefore, it must be taken into account when reasoning about landmarks, that is, the definition of a landmark in a STRIPS context must be extended to the temporal context.

Informally, a temporal landmark is a literal that will permit us to build the skeleton of a short plan, that is, we focus on obtaining (perhaps not the optimal but) one of the shortest plans.

<table>
<thead>
<tr>
<th>Initial state</th>
<th>Goal state</th>
</tr>
</thead>
<tbody>
<tr>
<td>at driver1 s1</td>
<td>at package1 s1</td>
</tr>
<tr>
<td>at driver2 s0</td>
<td>at package2 s0</td>
</tr>
<tr>
<td>at truck1 s1</td>
<td>at package3 s2</td>
</tr>
<tr>
<td>at truck2 s2</td>
<td></td>
</tr>
<tr>
<td>at package1 s0</td>
<td></td>
</tr>
<tr>
<td>at package3 s1</td>
<td></td>
</tr>
<tr>
<td>empty truck1</td>
<td></td>
</tr>
<tr>
<td>empty truck2</td>
<td></td>
</tr>
</tbody>
</table>

TABLE I
INITIAL AND GOAL STATE FOR OUR EXAMPLE
we have developed a technique that works iteratively over a stored in

Figure 1. Scheme

Consequently, the set of temporal landmarks will include the STRIPS landmarks plus some other literals that will belong to the shortest plans. In order to identify the temporal landmarks, we have developed a technique that works iteratively over a temporal landmarks graph. A scheme of this approach is shown in Figure 1.

First, a temporal planning graph (TPG) is built. This TPG will be the basis of the whole process. It is used to extract the temporal landmarks and to build the landmarks graph. Then, we compute a initial set of temporal landmarks (LMS). These landmarks and their dependencies compound the initial temporal landmarks graph.

Afterwards, we enter in an iterative process that activates each landmark in the most appropriate time point in the TPG. In this step, some temporal landmarks may disappear as the process goes on, whereas some other may appear, too. Moreover, a set of necessary, reasonable and landmark orders is computed between the landmarks obtained in the previous steps, to enrich the landmarks graph. Steps ACTIVATE and ORDERS can be executed several times, until no changes (new landmarks or new orders) are found. The following sections detail the whole process.

V. BUILDING THE TEMPORAL PLANNING GRAPH

A temporal planning graph consists of a directed, layered graph which alternates proposition levels \( P[t] \) and action levels \( A[t] \) with the propositions and actions, respectively, which are present in time \( t \). Each level represents a real time stamp (action durations are real values) instead of representing a simple planning step as in an STRIPS planning graph[17]. Consequently, time is explicitly stored in the levels of the graph which are chronologically ordered by their value.

Each level is splitted into two parts: end-part and start-part. The end-part (start-part) analyses all the actions that end (start) at that time and all their conditions/effects. Actions that end (start) at an action level \( A[t] \) are stored in \( A[t]_{end} \) (\( A[t]_{start} \)). Analogously, propositions achieved as final (initial) effects are stored in \( P[t]_{end} \) (\( P[t]_{start} \)).

An important part when building the temporal planning graph is the computation of mutexes between propositions and actions. We can distinguish two types of mutex relationships: static and dynamic mutexes.

Formally, we say that:

- (Static PP mutex) Two propositions \( p, q \) are statically PP mutex iff \( p \) is the negation of \( q \).
- (Static PA mutex) A proposition \( p \) is statically PA mutex with action \( a \) iff \( p \in \{ SDel(a) \cup EDel(a) \} \).
- (Static AA mutex) There exist four types of action/action static mutexes:
  1. Case 1 (AA\(_{start-start} \)) represents the situation when two actions cannot start together because: i) initial effects are PP mutex, or ii) initial effects and invariant conditions are PP mutex.
  2. Case 2 (AA\(_{end-end} \)) represents the situation when two actions cannot end together because final effects are PP mutex.
  3. Case 3 (AA\(_{end-start} \)) represents the situation when an action cannot end when another action starts because the final effects of the former are PP mutex with the invariant conditions/initial effects of the latter.
  4. Case 4 (AA\(_{during-during} \)) represents the situation when an action cannot start/end during the execution of another action because the start/end effects of the former are PP mutex with the invariant conditions of the latter.

The dynamic mutexes to be calculated in time \( t \) in the temporal planning graph are the action/action mutex \( AA[t] \), the proposition/action mutex \( PA[t] \) and the proposition/proposition mutex \( PP[t] \). We use the notation \( AA[t] \), \( PA[t] \) and \( PP[t] \) for the dynamic mutexes which are time-dependent and can disappear along the extension of the graph, whereas AA, PA and PP are used for the static mutexes, which always hold.

We organise the mutexes into three different groups:

1) Group 1 includes the mutexes to be calculated between ending actions and propositions: \( AA[t]_{end-end} \) (actions that are mutex ending in \( t \)), \( PA[t]_{end-end} \) (propositions that are mutex with actions ending in \( t \)), and \( PP[t]_{end-end} \) (propositions that are mutex in \( t \) after the ending of their supporting actions).
2) Group 2 includes the mutexes to be calculated when dealing with ending and starting actions: \( AA[t]_{end-start} \) (mutex between actions which end and start in \( t \)) and \( PP[t]_{end-start} \) (propositions that are mutex in \( t \) and are generated by actions which end/start in \( t \)).
3) Group 3 includes the mutexes to be calculated between starting actions and propositions: \( AA[t]_{start-start} \) (actions that are mutex starting in \( t \)), \( PA[t]_{start-start} \) (propositions that are mutex with actions starting in \( t \)), and \( PP[t]_{start-start} \) (propositions that are mutex in \( t \) after the starting of their supporting actions).

Intuitively, \( AA[t] \) mutex indicates the impossibility of two actions ending, starting or abutting together at the same time \( t \). \( PA[t] \) mutex indicates the impossibility of having a
proposition and an action starting or ending at time \( t \). \( PP[t] \)
mutex indicates the impossibility of having two propositions
together at time \( t \). This calculation of the mutex relationships
obtains the same mutexes as Graphplan, but now in a temporal
setting.

The temporal planning graph is incrementally generated by
means of the same forward chaining process of Graphplan.
Specifically, the process consists in generating all the actions
\( \{a_i\} \) in action level \( A[t] \) of the graph as soon as their
start and invariant conditions are non-pairwise mutex in the
proposition level \( P[t] \), generating their start and end effects
in the proposition levels \( P[t] \) and \( P[t+\Delta w(a_i)] \), respectively
(action \( a \) will end after \( \Delta w(a) \) time units).

Note that the persistence of the propositions/actions is
easily managed since they implicitly persist in time: if a
proposition/action is present at time \( t \), it will be present at any
time \( t' > t \). An important point to be considered when dealing
with actions with local conditions/effects in a Graphplan-based
approach is the condition to terminate the extension of the
temporal graph. In Graphplan, this condition holds once all
the problem goals are non-pairwise mutex. Now, actions may
assert start effects which might satisfy these goals before these
actions end. This implies that the temporal graph extension
must end at time \( t \) when all propositions in \( G \) are not dy-
namically mutex and their corresponding actions have ended.
This termination condition guarantees that a feasible plan will
never be shorter than \( t \). This termination condition, called
first termination condition, is necessary but not sufficient for
finding a plan, as it also happens in Graphplan.

The implementation used in our system is based on the TPG

Table II shows a part of the TPG generated for the problem
introduced in Section II. It is represented in two rows: the
first row includes from \( t = 1 \) until \( t = 56 \) and the second
row goes from the latter to the end of the TPG at \( t = 89 \).
The first component of each row denotes to the evolution
of the propositions. Notice that when a literal is added to
the TPG, it is persists until the last time point (we represent
this persistence by \( \rightarrow \)). The second component of each row
shows the starting of an action (ground action as the name
and its correponding parameters), the interval while it is
executed (represented by \( \rightarrow \)) and the ending of its execution
(represented by \( \downarrow \)).

The first level (column) contains all the literals in the initial
state. One of the first applicable actions is \texttt{Board driver1
truck1 s1} which generates literal \texttt{(driving driver1
truck1)} in the next time point. The process goes on as
explained above.

Also, the mutexes in each level are computed. For example,
at \( t = 1 \), literals \texttt{(empty truck1)} and
\texttt{(driving driver1 truck1)} are statically PP mutex
and actions \texttt{Drive-truck truck1 s1 s2 driver1}
and \texttt{Drive-truck truck1 s1 s0 driver1} are sta-
tically AA mutex.

Literals \texttt{(in package3 truck1)} and \texttt{(at truck1
s2)} are dynamically PA mutex at \( t = 56 \) because:

- literal \texttt{(at truck1 s2)} is PA mutex with the action
  \texttt{Load-truck package3 truck1 s1} that achieves
  \texttt{(in package3 truck1)} and
- literal \texttt{(in package3 truck1)} is PA mutex with the
  action \texttt{Drive-truck truck1 s1 s0 driver1}
  that achieves \texttt{(at truck1 s2)}

At \( t = 57 \), this restriction dissapears due to the fact that it
is possible to execute these actions sequentially and, therefore,
action \texttt{Unload-truck package3 truck1 s2} can start.

Another remarkable situation in the TPG occurs at \( t = 87 \).
At this moment, action \texttt{Unload package1 truck1 s1}
starts, whose preconditions are \texttt{(at truck1 s1)} and \texttt{(in
package1 truck1)}. These two literals are present in the
TPG from \( t = 45 \). However, they are PP mutex until \( t =
87 \), and this is the reason why action \texttt{Unload package1
truck1 s1} cannot start until this time point.

VI. LANDMARKS EXTRACTION

This section explains how landmarks are extracted from the
TPG computed in the previous step of the algorithm.

We use a propagation method similar to Zhu’s process [14].
Our propagation technique works over a TPG extended until
the first termination condition is satisfied (let \( T \) be this time
point) and only labels proposition nodes (action nodes are used
to compute these labels, but are not labeled). In the initial
graph level, every proposition is labeled with itself and no
other labels. In the following levels, each proposition node \( p \)
in the graph is labeled with the intersection of the labels on the
preconditions of the actions that add \( p \). When the propagation
is complete, any label on a goal proposition in the final level
is a temporal landmark for plans achieving the goal in \( T \) time
units.

It is important to remark that, with this process, we do
not only extract the STRIPS landmarks, but also some extra
literals that will be necessary to achieve the goal in \( T \) time
units. If a plan exists in \( T \) time units, this will be the optimal
plan. Furthermore, since these extra literals must be achieved
to obtain the goal in \( T \) time units, these literals are leading to
the optimal plan.

In case the optimal plan is contained in a TPG which ends
later than \( T \), the TPG must be extended. As we will show
below, the activation phase may require to extend the TPG
and, in this process, some of the landmarks extracted at this
point may dissappear, whereas some others may appear as
well.

Once the landmarks have been extracted, we obtain a
labeled TPG (LTPG), where all the propositions at each time
point \( t \) have a label with the literals that would be necessary
to obtain that proposition at time \( t \). This LTPG will be used
in the following steps of our approach.

Let’s illustrate this method with the extraction of the
landmarks of our example, whose partial TPG is shown
in Table II. First, all the literals in the initial state
are labeled with themselves. At \( t = 1 \), when literal
\texttt{(driving driver1 truck1)} is introduced in the TPG,
we compute the landmarks as the intersection of the labels of the preconditions of the actions achieving this literal. There is only one action, Board driver1 truck1 s1, whose preconditions are (at driver1 s1), (at truck1 s1) and (empty truck1). Therefore, literal (driving driver1 truck1) is labeled with these three literals plus itself. Let’s move later on the time line. At t = 56, literal (at truck1 s2) is added to the TPG due to the fact that action Drive truck1 s1 s2 driver1 finishes. The landmarks for the literal (at truck1 s2) will be the landmarks of the preconditions of the action (Drive truck1 s1 s2 driver1), which consist in (driving driver1 truck1) and (at truck1 s1), plus itself.

At the end of the process, the landmarks for the literals in the goal state ((at package3 s2) and (at package1 s1)) are the temporal landmarks for the problem. In this case, these temporal landmarks are:

<table>
<thead>
<tr>
<th>Initial State</th>
<th>Goal</th>
<th>Temporal landmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>at driver1 s1</td>
<td>at package3 s2</td>
<td>at truck1 s2</td>
</tr>
<tr>
<td>at truck1 s1</td>
<td>at package1 s1</td>
<td>in package3 truck1</td>
</tr>
<tr>
<td>at package1 s0</td>
<td>empty truck1</td>
<td>driving driver1 truck1</td>
</tr>
</tbody>
</table>

It is important to remark that none of the temporal landmarks correspond to a STRIPS landmark. This means that these literals will not appear in all solution plans but, in case there is a plan executable in 89 time units, they will lead to the optimal plan. Otherwise, they still may lead to this optimal plan if the TPG needs to be extended in the next phase.

VII. ACTIVATION OF LANDMARKS

After the extraction of landmarks, we have a list of literals that need to be achieved in the final plan. In order to build the skeleton of this solution plan, it is necessary to establish an order between these literals, thus obtaining a temporal landmarks graph.

A first temporal planning graph is built by establishing an order between two landmarks \( l \) and \( l' (l \leq l') \) when \( l \) appears in the label of \( l' \) in the LTPG at the first time point where \( l' \) is added to the TPG. This represents that \( l' \) depends on \( l \).

Then, this temporal planning graph is iteratively refined at the activation step and at the orders step (this step will be explained in the next section). The aim of this activation process is to place each landmark at the most appropriate time point in the LTPG. Since a literal added in the TPG persists “forever”, the objective of this process is to select which appearance (time point) of each landmark is the most appropriate for obtaining the final plan. This selection is carried out by studying two types constraints: mutex relationships and landmark dependencies.

Before introducing the algorithm in detail, we need some definitions:

- **An activated landmark** is a landmark whose most appropriate occurrence has been activated in the LTPG.
A pending landmark is a landmark that has not been activated at any time point.

$Lms_{t}(l)$ is the list of landmarks of a literal $l$ at a time point $t$, that is, the labels of $l$ in the LTPG at $t$.

$AP_{t}$ is the list of activated propositions at a time point $t$ of the LTPG; it follows that $AP_{t} \subseteq P_{t}$

$AA_{t}$ is the list of activated actions at a time point $t$ of the LTPG; it follows that $AA_{t} \subseteq A_{t}$

The first step in this process is to activate the literals whose dependencies (landmarks) have been previously activated. This process is shown in Algorithm 1. First, all the literals in the initial state are activated and removed from the set of pending landmarks (steps 1 and 2). The remaining pending landmarks are processed according to the time point where they are first added to the TPG (steps 3 and 4). Let $l$ be a landmark to be processed. In case all the landmarks for $l$ in the first time point $t''$ where $l$ is not mutex with any other activated landmarks, have been already activated (step 5), $l$ is activated in $t''$ and removed from the set of pending landmarks (steps 6 and 7). Moreover, all the actions supporting $l$ in $t''$ are activated, too (steps 8 to 10).

With this algorithm, we activate the “easy” landmarks, that is, those landmarks whose dependencies were already activated. The remaining landmarks (pending landmarks) need a more sophisticated process, that consists in checking whether the landmarks they depend on can be activated. Specifically, we recursively decide whether $l$ can be activated at a certain time point by checking that its dependencies have been already activated or that they, in turn, could be activated (because, again, its dependencies have been already activated or could be activated, etc.).

The process to activate the pending landmarks is shown in Algorithm 2. Again, we process the pending landmarks in order according to the time point where they are first added in the TPG (steps 1 and 2). Let $t'$ be the first time point where $l$ (a pending landmark) could be activated (step 3). We compute $t_{m}$ as the first time point where $l$ is not mutex with any other activated literal (step 4) and $t_{d}$ as the time point where $l$ would be first added by an action whose preconditions have been activated before $t'$ (step 5).

If $t_{d}$ or $t_{m}$ cannot be obtained, it implies that the TPG must be extended (steps 6 and 7) because it is not possible to achieve $l$ before $T$ due to mutex relationships or dependencies. Otherwise, $l$ is activated at $t''$, computed as the maximum between $t_{d}$ and $t_{m}$ (steps 9 and 10). $l$ is removed from the set of pending landmarks, but the landmarks corresponding to $l$ at $t''$ (which can also be activated) are added to the pending landmarks (step 11). At this point, new landmarks may appear because all the dependencies of $l$ at $t''$ are considered as temporal landmarks. The actions that support $l$ at $t''$ are activated if they are no mutex with any other previously activated action.

Now, we illustrate how these algorithms work with our example. First, landmarks in the initial state are activated. The remaining landmarks are considered as pending landmarks. The first literal to be studied is (driving driver1 truck1). At $t = 1$ (when it is first added to the TPG), its landmarks are literals from the initial state, already activated. Therefore, this literal is activated at $t = 1$. The same occurs with literal (in package3 truck1), which is activated at $t = 2$.

Next proposition is (in package1 truck1), whose landmarks are literals from the initial state, (driving driver1 truck1) (all of these already activated) and (at truck1 s0), which has not been activated at this moment. Therefore, literal (in package1 truck1) remains as a pending landmark. It is important to remark that literal (at

---

**Algorithm 1** Easy activation of literals

1: $AP_{0} = AP_{0} \cup \{T\}$
2: pending landmarks=all extracted landmarks - $\emptyset$
3: for all time point $t$ from 0 to $T$
4: for all $l$ in pending landmarks that appears at $t$ for first time
5: if $\exists t' > t : \neg \exists l' \in AP_{t'} : mutex(l,l') \land \forall l'' \in Lms_{t'}(l), \exists l'' < t : l'' \in AP_{t''} \text{ then}$
6: $AP_{t} = AP_{t} \cup \{l\}$
7: pending landmarks=pending landmarks - $\{l\}$
8: $Actions = \{a \in A[t'_\text{end}] \mid l \in EAdd(a)\} \cup \{a \in A[t'_\text{start}] \mid l \in SAdd(a)\}$
9: for all $a \in Actions \land \neg \exists a' \in AA_{t'} : mutex(a,a')$
10: $AA_{t'} = AA_{t'} \cup \{a\}$
11: end for
12: end if
13: end for
14: end for

**Algorithm 2** General activation of literals

1: for all time point $t$ from 0 to $T$
2: for all $l$ in pending landmarks that appears at $t$ for first time
3: if $\exists t' > T : t'$ is the first time point where $l$ can be activated
4: $t_{m}$ = first time point where $\neg \exists l' \in AP_{t_{m}}$ : $mutex(l,l')$
5: $t_{d} = t' + \max \{0 \text{ if } \exists a \in A[t'_\text{start}] : a \in SAdd(a)\}
6: if $t_{d} = \infty \lor t_{m} = \infty$ then
7: extend TPG
8: else
9: $t'' = \max(t_{d}, t_{m})$
10: $AP_{t''} = AP_{t''} \cup \{l\}$
11: pending landmarks=pending landmarks - $\{l\}$ \cup $Lms_{t''}(l)$
12: Activate actions (see steps 8 to 11 Algorithm 1)
13: end if
14: end if
15: end for
16: end for
truck1 s0) was not extracted as landmark, but now is considered because it is necessary to achieve (in package1 truck1) at $t = 45$.

Then, we process literals (at truck1 s2) and (at package3 s2), which are activated. The last proposition (at package1 s1) cannot be activated because it depends on (in package1 truck1), that could not be activated before, as shown above.

Next step (Algorithm 2) in the activation process tries to activate literals (in package1 truck1) and (at package1 s1), which are still pending landmarks. First, we check whether literal (in package1 truck1) can be activated. As seen above, all the dependencies of literal (in package1 truck1), except (at truck1 s0), have been already activated. Therefore, we have to check whether (at truck1 s0) can be activated and at what time point:

1. The first time point where all its dependencies have been already activated is $t = 43$, so it can be activated at this time point.
2. Now, we need to compute $t_m$ for this literal, that is, the first time point where it is not mutex with any other literal or action:
   - At $t = 43$, it is mutex with action Drive truck1 s1 s2 driver1 which have been already activated. It is necessary to wait until this action finishes at $t = 56$.
   - At $t = 56$, it is mutex with literal (at truck1 s2) until $t = 59$.
   - At $t = 59$, it is mutex with literal (at package3 s2) until $t = 79$.

After all checkings, (at truck1 s0) can be activated at $t = 79$. This means that now (in package1 truck1) can be activated. We compute $t_m = 79$ and $t_a = 81$ for this literal and, finally, this literal is activated at 81. Therefore, literal (at truck1 s0) which was not a temporal landmark initially, is added to the list of pending landmarks (to be activated).

The last proposition is (at package1 s1) whose dependencies have been already activated at $t = 89$ and, consequently, it is activated at this time point.

VIII. Orders

At this moment, we have a temporal landmarks graph where each landmark has been activated at the most appropriate time point. However, there are some relationships between them that still can be extracted in order to obtain a more accurate skeleton of the final plan.

Each edge of the landmarks graph is called a landmark order. We distinguish two types of landmarks orders:

- There is a permanent landmark order between $l$ and $l'$ ($l <_p l'$) if $l'$ is missing from a RPG built for a new planning problem where the actions adding the literal $l$ are eliminated. This means that, in order to satisfy $l'$ in a plan, it is always necessary to achieve $l$ firstly.
- The remaining landmark orders are considered as non-permanent landmark orders.

A permanent landmark order implies that, in order to obtain $l'$ from $l$, we need a path of actions. In case this path contains only one action, we say that this is a necessary order. More precisely, an existing edge between $l$ and $l'$ (added at $t'$) is converted into a necessary order ($l <_n l'$) if $\forall a \in A[t'_\text{end}] : l' \in E\text{Add}(a) \cup a \in A[t'_\text{start}] : l' \in S\text{Add}(a) \rightarrow l \in S\text{Cond}(a) \cup \text{Inv}(a)$.

From the set of necessary orders we can compute a new type of orders called reasonable orders [15]. A reasonable order between $l$ and $l'$ states that it is more reasonable to achieve $l$ first, because otherwise $l'$ would be achieve twice in the plan. The introduction of this type of orders will give us additional relationships between the landmarks in the graph, which can be exploited to enrich the skeleton of the final plan.

These orders are propagated along the graph, which may imply a modification of the activation time of some landmarks. In this case, the activation step is performed again.

Figure 2 shows the temporal landmarks graph obtained for our example after the computation of necessary and reasonable orders. For example, there is a necessary order between (at driver1 s1) and (driving driver1 truck1) because the only activated action that generates the latter needs the former. On the other hand, there is a reasonable order between (in package3 truck1) and (at truck1 s2) indicating that before driving the truck it is necessary to load the package.

As it can be observed, a cycle between some landmarks appears. At this moment, we are developing a technique to deal with cycles in order to extract additional information to improve the skeleton of the final plan.

IX. Results

This section shows the experiments we have performed in order to test our approach. We are interested on two aspects. First, we want to know whether our iterative process is able to obtain more than only STRIPS landmarks. Second, we want to compare the structure of the skeleton obtained from our technique with the structure of an optimal plan2. The main reason is that if we obtain more than STRIPS landmarks and all these landmarks appear in the optimal plan, this implies that we would be able to obtain an (near to) optimal plan from this temporal landmarks graph.

Table III shows the results obtained for three domains, driverlog, zeno and depots, which were introduced in the 3rd International Planning Competition [12]. For each problem, we show two values: (1) the percentage of no-STRIPS landmarks with respect to the total number of landmarks in the temporal landmarks graph and (2) the percentage of landmarks in the temporal landmarks graph with respect to the literals in the optimal plan (literals that are generated and used in the plan plus the literals from $I$ and $G$).

As it can be observed, the percentage of no-STRIPS landmarks varies from 0 to 50% across the problems, but in average is approximately 21%. This means that we are obtaining a

2We have used LPG [9] to calculate the optimal plan.
good number of no-STRIPS landmarks, which enrich the structure of the skeleton of the final plan. Moreover, only the temporal landmarks graphs of four problems have one or two landmarks that are not in the optimal plan.

On the other hand, the obtained temporal landmarks graphs contain approximately the 70% of the literals in the final plan, which is a very good rate. The missing literals correspond to situations where there are, for example, more than one truck to use and there is no difference if one or the other is used.

## X. Conclusions

In this paper, we have presented a new technique to extract landmarks in a temporal context. This technique works iteratively over a temporal landmarks graph in order to obtain the skeleton of the final plan.

Our approach consists in four stages: first, a temporal planning graph is built; then, a set of initial temporal landmarks is computed from this TPG and we build the initial temporal landmarks graph; the next step activate each landmark in the most appropriate time point by taking into account mutex relationships and dependencies; the final step computes necessary and reasonable orders between landmarks which enrich the skeleton of the plan.

The results show that our technique is very interesting because it is able to obtain a temporal landmarks graph whose structure is very similar to the final plan (in average, there is a difference of less than 30%).

At this moment, we are working in improving this technique to handle cycles in the temporal landmarks graph, which will provide further information about the structure of the plan.

### References


