

Efficiently handling temporal knowledge in an HTN planner*

Luis Castillo and Juan Fdez-Olivares and Óscar García-Pérez and Francisco Palao

Dpto. Ciencias de la Computación e I.A.
University of Granada, SPAIN
{L.Castillo,Faro,Oscar,Palao}@decsai.ugr.es
<http://siadex.ugr.es>

Abstract

This paper presents some enhancements in the temporal reasoning of a Hierarchical Task Network (HTN) planner, named SIADEX, that, up to authors knowledge, no other HTN planner has. These new features include a sound partial order metric structure, deadlines, temporal landmarking or synchronization capabilities built on top of a Simple Temporal Network and an efficient constraint propagation engine boosted by exploiting the causal structure of plans.

Introduction

The achievement of an efficient and expressive handling of time is still a pending task for most HTN planners. This issue becomes even harder if the planner follows a state-based forward paradigm like SHOP (Nau *et al.* 2003) or SIADEX (de la Asunción *et al.* 2005) since, despite being a very fast HTN planning paradigm, it does not easily allow to obtain plans with timed concurrent branches. From a practical point of view, real world applications need the plans to have the possibility of executing several activities at the same time. Furthermore, real applications usually require complex synchronization mechanisms between the activities of the plan for a successful execution. And last, but not least, temporal knowledge has to be efficiently handled so as to allow a fast response of the planning system. This paper explains how the HTN planner SIADEX has been extended to cope with all these requirements thanks to the use of Simple Temporal Networks (STN) (Dechter, Meiri, & Pearl 1991). STNs have been widely used as the underlying representation of temporal constraints in planning and scheduling frameworks like in Mapgen (Ai-Chang *et al.* 2004), Mexar (Oddi *et al.* 2002), PASSAT (Myers *et al.* 2002), Ixtet (Laborie & Ghallab 1995) or OPlan (Tate, Drabble, & Kirby 1994); since they provide a very expressive power to represent a variety of temporal constraints and a flexible execution of plans with flexible timelines. All these approaches use temporal constraints of different nature to represent their

own temporal knowledge. The main contribution of this paper is based on how temporal constraints are extracted and propagated in a HTN framework:

- Any temporal constraint, either precedence constraints or deadlines, defined on an abstract task, or between several of them, are implicitly inherited by its constituent sub-tasks.
- Abstract tasks and primitive actions may generate temporal landmarks on their start or end points to achieve complex synchronization schemas between them, either between tasks or between actions and tasks.
- PDDL 2.2 timed initial (Edelkamp & Hoffmann 2004) literals are easily represented in a STN framework and they are used as backtracking points that serve as anchorage points for tasks and actions.
- Despite being an HTN planner, the causal rationale of primitive actions is recorded and used to propagate accurate temporal constraints between them.
- Since HTN planners are used in many real applications with tight response times, the propagation algorithms defined for STN, although polynomial, still produce a considerable overload when handling large temporal plans in the order of hundreds or thousands of actions. Therefore, this paper also proposes a modification of a well known propagation algorithm named PC-2 (Dechter 2003) to boost its performance thanks to the information extracted from the causal structure of the plan, obtaining excellent results.

These contributions have raised in the framework of application of SIADEX devoted to forest fire fighting planning (Fdez-Olivares *et al.* 2006) due to the need to obtain plans with complex temporal requirements like the synchronization of several teams of workers, the existence of temporal windows of activity, precedence constraints due to the use of shared resources or dealing with timed exogenous events. However, they are very common needs for other real problems and therefore of general interest for other application areas.

The paper is structured as follows. Next section outlines SIADEX, an HTN state-based forward planner. Then, we will present the main extensions to SIADEX to cope with the most important representational issues of time in

*This work has been partially supported under the research contract NET033957/1 with the Andalusian Regional Ministry of Environment and the CICYT Project TIC2002-04146-C05-02
Copyright © 2006, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

```

(:task travel-to
:parameters (?destination)
(:method Fly
:precondition (flight ?destination)
:tasks ((go-to-an-airport)
        (take-a-flight-to ?destination)))
(:method Drive
:precondition (not (flight ?destination))
:tasks ((take-my-car)
        (drive-to ?destination))))

```

(a)

```

(:durative-action drive-to
:parameters(?destination)
:duration (= ?duration
           (/ (distance ?current ?destination)
              (average-speed my-car)))
:condition(and (current-position ?current)
               (available my-car))
:effect(and (current-position ?destination)
            (not (current-position ?current))))

```

(b)

Figure 1: The basics of HTN planning domains in SIADEX' domain language: (a) A compound *task* with two different *methods* of decomposition. (b) A primitive *action*.

a HTN framework. Later, we introduce the use of deadlines and their inheritance from tasks to actions and present a very simple improvement in temporal constraints propagation that achieves great experimental results in different domains. Last section relates this approach with the existing literature.

Description of SIADEX

SIADEX is a state-based forward HTN planner with the same foundations than SHOP (Nau *et al.* 2003). Before going into the details, some introductory notions are explained first.

HTN planning foundations

HTN planning domains are designed in terms of a hierarchy of compositional activities. Lowest level activities, named actions or primitive operators, are non-decomposable activities which basically encode changes in the environment of the problem. In SIADEX, these primitive operators are represented as PDDL 2.2 level 3 durative actions (Edelkamp & Hoffmann 2004) (Figure 1.b). PDDL is the standard planning domain description language and it is the basis of most well known planners. On the other hand, high level activities, named tasks, are compound actions that may be decomposed into lower level activities. Depending on the problem at hand, every task may be decomposed following different schemas, or methods, into different sets of sub-activities. These sub-activities may be either tasks, which could be further decomposed, or just actions (Figure 1.a).

Tasks and their, possibly multiple, decompositions encode domain dependent rules for obtaining a plan, that can only be composed of primitive actions. Other HTN features are

- Set \mathcal{A} , the agenda of remaining tasks to be done, to the set of high level tasks specified in the goal.
- Set $\Pi = \emptyset$, the plan.
- Set \mathcal{S} , the current state of the problem, to be the set of literals in the initial state.
 1. Repeat while $\mathcal{A} \neq \emptyset$
 - (a) **Extract** a task t from \mathcal{A}
 - (b) if t is a primitive action, then
 - i. If \mathcal{S} satisfies t preconditions then
 - A. Apply t to the state, $\mathcal{S} = \mathcal{S} + additions(t) - deletions(t)$
 - B. Insert t in the plan, $\Pi = \Pi + \{t\}$
 - C. **Propagate-Temporal-Constraints**(Π)
 - ii. Else **FAIL**
 - (c) if t is a compound action, then
 - i. If there is no more decomposition methods for t then **FAIL**
 - ii. **Choose** one of its decomposition methods of t whose preconditions are true in \mathcal{S} and map t into its set of subtasks $\{t_1, t_2, \dots\}$
 - iii. Insert $\{t_1, t_2, \dots\}$ in \mathcal{A} .
 2. **SUCCESS**: the plan is stored in Π .

Figure 2: A rough outline of an HTN planning algorithm showing the point at which temporal constraints are propagated (step 1(b)iC).

the following ones:

- The initial state is a set of literals that describe the facts that are true at the beginning of the problem.
- Unlike non HTN planners, goals are not specified as a well formed formula that must be made true by the planner from the initial state. Instead, goals are described as a partially ordered set of tasks that need to be carried out.
- The main planning algorithm (Figure 2) takes the set of tasks to be achieved, explores the space of possible decompositions replacing a given task by its component activities, until the set of tasks is transformed into a set of primitive actions that make up the plan.

Including inference capabilities in SIADEX

HTN planning approaches have a very rich knowledge representation that may arise in a variety of forms, from methods preconditioning and control of the search (Nau *et al.* 2003), ontology based representation of planning objects (Gil & Blythe 2000; Fdez-Olivares *et al.* 2006) or knowledge-intensive planning procedures (Wilkins & desJardins 2001). In our case we use two augmented inference capabilities that allows the planner to infer new knowledge, either by abduction or deduction, over the current state of the problem at every planning step.

Regarding the use of deductive inference, SIADEX provides inference tasks that may be fired when needed in a task decomposition scheme. The effect of these inference tasks is that they may produce binding of variables and assert/retract new literals into the current state. The form of these deductive inference rules is

```
(:inline <precondition> <consequents>)
```

where both precondition and consequent are logical expressions, what means that when $\langle\text{precondition}\rangle$ is true in

```

(:task travel-to
:parameters (?destination)
(:method Fly
:precondition (flight ?destination)
:tasks ((go-to-an-airport)
(:inline (and (current-position ?airport)
(has-wifi-hotspot ?airport))
(enabled-wifi))
(take-a-flight-to ?destination))) ... )

```

Figure 3: Including deductive rules in the expansion of a high level task

the current state, the effects of the `<consequent>` are applied to the current state, allowing to assert or retract new literals. Let us consider the task description shown in Figure 3. We may see that the deductive inference rule allows the planner to include a new literal in the state (`enabled-wifi`) whenever the departure airport has a wi-fi hot-spot.

This might be seen as a conditional effect of some previous action but there are some differences that have to be clarified. Firstly, the activities in the decomposition which precede the inference rule may not be primitive actions, therefore, it is not always possible to encode them as conditional effects since high level tasks are not allowed to have effects. And secondly, the problem of the context of firing has to be considered, that is, if we encode every possible deducible consequence of a primitive action as sequence of exhaustive conditional effects, then we might be leading to a very well known problem in planning, the ramification problem (McIlraith 2000), and to an overload of the deductive process and unifications with the current state. Therefore, the inclusion of an inference rule into the decomposition of a high level task provides the context in which this inference is necessary and therefore, it will only fire at that moment.

Although the use of deductive inference has also appeared in the literature (Wilkins 1988), the use of abductive inference is becoming more widely used in the form of axioms (Nau *et al.* 2003) or derived literals, in terms of PDDL 2.2 (Edelkamp & Hoffmann 2004). These are abductive rules which appear in the form of a Horn clause and that allow to satisfy a given condition when this condition is not present in the current state, but it might be inferred by a set of inference rules of the form

```

(:derived <literal> <logical-expression>)

```

meaning that `<literal>` is true in the current state when the literals in `<logical-expression>` are also true in the current state. Both deductive and abductive inference rules are extensively used to incorporate additional knowledge to the planning domain while maintaining actions and tasks representations as simple as possible. This is particularly true in real-world planning problems like forest fire fighting plans design, the main application of SIADEX so far. In these cases, planning knowledge usually comes from very different sources, not always related to causality and therefore not very appropriate to encode in the causal structure of actions preconditions and effects. Another of the main purposes of these rules is to allow a process of temporal landmarking over the temporal representation of

SIADEX that enrich the set of temporal constraints that may be encoded in a planning problem.

Temporal enhancements of SIADEX

One of the main drawbacks of state-based forward planners (HTN and non-HTN) is that they usually return plans as a total order sequence of activities, that is, a chain of actions.

$$\Pi = \{a_1, a_2, \dots, a_n\}$$

If the planner is based on states, then, this sequence of actions also induces a sequence of states.

$$INITIAL + s_1 + s_2 + s_3 + \dots + s_n$$

where s_i is the state that results from the execution of action a_i over the state s_{i-1} . However, in many real world applications, several activities may be carried out in parallel and a total order of activities is not very appropriate for practical reasons. In order to obtain plans with parallel branches, SIADEX uses several techniques: the definition of qualitative partial order relationships in the domain, the inference of new metric temporal constraints from the causal structure of the plan, the definition of deadline goals and complex synchronization schemas. These techniques are based on the inference capabilities explained before and the handling of metric time over a STN. A STN (Dechter, Meiri, & Pearl 1991) is a structure (X, D, C) such that X is the set of temporal points, D is the domain of every variable and C is the set of all the temporal constraints posted. In our case, a plan is deployed over a STN following a simple schema, every action $a_i \in \Pi$ owns two time points $start(a_i)$ and $end(a_i)$. Besides actions, every task t_i that had been expanded in the plan Π generates two time points $start(t_i)$ and $end(t_i)$ which bound the time points of its subactivities. There is an additional time point TR which represents the absolute temporal reference at the beginning of a plan. All the time points share the same domain $[0, \infty)$ and the constraints in C are posted and propagated at planning time in a two-steps refinement process. First, once an abstract task is decomposed into subtasks, low detail qualitative temporal constraints are introduced in the plan to arrange subtasks according to their order relation expressed in the domain. This could be seen as the plan's temporal skeleton. Later, once their sub-actions are being included definitely in the plan, more precise temporal constraints are added to encode the causal relationships between every final action in the plan.

First step: introducing qualitative orderings

One of the main sources of temporal constraints between the actions of the plan comes from the order in which the subtasks of a given task appear in its decomposition. In this way, SIADEX' domains may impose three different types of qualitative ordering constraints in every decomposition.

Sequences They appear between parentheses $(T1, T2)$, and they are a set of subtasks that must execute in the same order than the decomposition, that is, first T1 and then T2. Please note that any possible subtask of T1 and T2 inherit these relations too, so that the imposed ordering is maintained even through their decompositions.

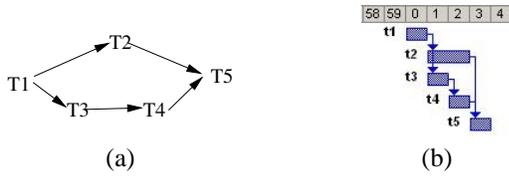


Figure 4: A graphical representation of the partial order decomposition given by $(T1 [T2 (T3 T4)] T5)$ (left hand side) and the plan representation found by SIADEX (right hand side)

Unordered They appear between braces $[T1, T2]$, and they are a set of subtasks that are not ordered in their decomposition, that is, either T1 or T2 could execute first.

Permutations They appear between angles $\langle T1, T2 \rangle$, and they are set of subtasks that must execute in any of the total orders given by any of their permutations, that is, first T1 and then T2 or vice versa. In these cases, the choice of the best permutation takes part in the search process of the planner. Please note that any possible subtask of T1 and T2 also inherit these relations.

For example, a method that decompose the task t into $(T1 [T2 (T3 T4)] T5)$ represents a partially ordered decomposition depicted in Figure 4.a) and the plan obtained by SIADEX is shown in Figure 4.b).

These qualitative temporal constraints are posted as $[0, \infty)$ between the start and end points of the respective actions in the STN.

Second step: introducing quantitative orderings from causal links

The qualitative orderings seen before allow to encode relatively simple order relations that are the main sources of partial ordering in the final plan. But there is an additional source of ordering information that is very useful for encoding more accurate metric temporal constraints: the causal structure of actions. This section explains how to enhance the representation of states with temporally annotated literals, and how to exploit the existence of causal links between primitive activities to encode sharply defined metric temporal constraints between them, providing a more precise, but complementary, source of temporal constraints than these qualitative orderings.

All the literals l_j^k in the effects of every action a_j have a delay Δt_j^k by which they are achieved after the execution of their corresponding action. This delay Δt_j^k may range from 0 (the effect is achieved at the beginning of a_j) or the duration of a_j (it is achieved at the end). This is a generalization of PDDL 2.2 *at-start* and *at-end* effects

```
(:durative action a
...
:effects (at  $\Delta t_j^k$  (literal))
```

We may have “at start” effects, leaving $\Delta t_j^k = 0$, or “at

end” effects, in this case $\Delta t_j^k = duration(a_j)$ or any other intermediate value.

SIADEX uses the information about delayed effects to represent states as a temporally annotated extension of classical states, where every literal is timestamped with the time by which it is achieved with respect to the action that produced it. Therefore, given a total order sequence of actions and its induced sequence of states

$$\Pi = a_1, a_2, \dots, a_n \rightarrow INITIAL + s_1 + s_2 + s_3 + \dots + s_n$$

every state s_i is given by $s_i = \{ \langle l_j^k, a_{j,j \leq i}, \Delta t_j^k \rangle \}$, that is, a set of literals l_j^k coming either from the effects of some previous action a_j ($l_j^k \in effects(a_j)$) or from the initial state (in this case the literal would have the form $\langle l_0^k, 0, 0 \rangle$). This means that literal l_j^k was introduced in the state s_i Δt_j^k time units after the execution of a_j given by the time point $start(a_j)$. Then, a temporally annotated state allows to know which action, if any, produced that literal amongst the preceding actions and at which time they were achieved.

This information is particularly useful during the planning phase for two different reasons.

Firstly, they allow to propagate accurate temporal constraints between actions (Figure 2, step 1(b)iC). For every literal in the preconditions of an action a_i that matches a temporally annotated literal $\langle l_j^k, a_j, \Delta t_j^k \rangle \in s_{i-1}$ we record a temporal causal link and use this information to post a temporal constraint, given by $[\Delta t_j^k, \infty)$, between the producing action, i.e. $start(a_j)$, and the consumer action, i.e. $start(a_i)$. This means that the consumer action $start(a_i)$ must wait at least Δt_j^k time units after $start(a_j)$, that is, the time needed for a_j to produce the desired effect l_j^k . Therefore, a temporal causal link allows to propagate temporal constraints due to the causal structure of the plan and taking into account the delays of the effects of supporting actions, and adds specific temporal constraints to the qualitative temporal constraints included formerly.

And secondly they allow to unfold the total order sequence of actions into a partially ordered plan where actions only depend of those other actions that produce any of the literals needed to satisfy their preconditions and are independent of the others. To ensure a correct causal structure of the plan within this unfolding operation, a simple protection mechanism, similar to threat removal operations in partial order planning (Weld 1994), adds some additional constraints. When an action a_i is included in the plan, and a_i deletes a literal l_j^k , an empty causal link is added such that orders a_i after any other action a_k , $k < i$ already present in the plan that depends on the same literal (has a non-empty causal link). Resulting temporal plans are not required to have a total order structure, in fact, they only have the minimum required temporal constraints to ensure a correct causal structure, which may be a total order or not depending on the causal structure of the plan.

Temporally extended goals

The process explained so far was mainly devoted to unfold the total order relation in which actions are obtained (see Figure 2) into a partial order plan that only records the causal structure of the plan, as a least commitment unfolding strategy (Figure 4.b). However, some real problems require the posting of more complex temporal constraints between the actions of a plan that not always have a cause-effect relationship. In this section, two additional enhancements are presented devoted, on one hand side, to deadline goals, that is, goals that must be achieved at a certain time, and in the other hand, complex synchronization schemas to allow actions to interact along the time.

Deadline goals

A deadline activity (either a task or an action) is an activity that may have defined one or more metric temporal constraints over its start or its end or both. Furthermore, in the case of tasks, SIADEX also allows to post deadline constraints on the start or the end of an activity (or both). Any sub-activity (either task or action) has two special variables associated to it: `?start` and `?end` that represent its start and end time points, and some constraints (basically `<=`, `=`, `>=`) may be posted to them. In order to do that, when any activity appears in the decomposition of a higher level task, it may be preceded by a logical expression that defines the desired deadline, either simple or compound, as it is shown in Figure 5. It shows that task A has no deadlines in its decomposition but task A' has a deadline that states that its subaction A2 must start after 3 time units and end before 5 time units from the beginning of the plan. Deadline goals are easily encoded in the STN of a temporal plan as absolute constraints with respect to the reference time point *TR*, the absolute start point of a STN. They may also appear in the top level goal and they are very useful for defining time windows of activity, that is, sets of activities that must be executed within a given temporal interval, or timed trajectories of goals, i.e., sequences of goals that may be achieved one after the other like a recipe.

Time points of subtasks of any task with deadlines are embraced by the time points of the task, this means that, in practice, subtasks inherit the deadlines. For example, Figure 6 shows how the deadlines defined over task A2 in Figure 5 constraint the timepoints of its subactions a21 and a22. In this example, this inheritance process also has an useful collateral effect. Since the makespan of task A2 is restricted to be within $[3, 5]$ time units, this enforces the planner to look only for valid decompositions of A2 that meet these constraints, or in other words, any possible decomposition of A2 that takes longer than 2 time units would produce a inconsistency in the STN and, thus, would make the planner to backtrack. Therefore, posting deadlines either on top level tasks or on intermediate tasks, would lead to a simple optimization of the makespan either of the whole plan or of isolated branches of it.

```
(:task A
:parameters ()
(:method A
:precondition ()
:tasks ((A1)
(A2))))

(:task A'
:parameters ()
(:method A'
:precondition ()
:tasks ((A1)
((and (>= ?start 3)(<= ?end 5)) (A2))))))

(:task A2
:parameters ()
(:method A2
:precondition ()
:tasks ((a21)
(a22))))
```

Figure 5: Task A has no deadlines in its decomposition, just that A2 must appear after A1. Task A' has deadlines in its decomposition stating that subtask A2 must appear after A1 but A2 must start after 3 time units and end before 5 time units since the beginning of the plan.

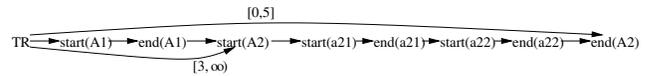


Figure 6: A task with deadlines imposes its constraints to its subtasks.

Temporal landmarking and complex synchronizations

SIADEX is also able to record the start and end of any activity and to recover these records in order to define complex synchronizations schemas between either tasks or actions as relative deadlines with respect to other activities. The first step is the definition, by assertion, of the temporal landmarks that signal the start and the end of either a task (Figure 7.a) or an action (Figure 7.b). These landmarks are treated as PDDL fluents but they are associated to the time points of the temporal constraints network and, therefore, fully operational for posting constraints between them in the underlying STN.

These landmarks are asserted in the current state, and later on, they may be recovered and posted as deadlines to other tasks in order to synchronize two or more activities. For example, Figure 8 shows the constraints needed to specify that action b must start exactly at the same time point than task A2.

In particular, thanks to the expressive power of temporal constraints networks and to the mechanism explained so far, a planning domain designer may explicitly encode in a problem's domain all of the different orderings included in Allen's algebra (Allen 1983) between two or more tasks, between two or more actions or between tasks and actions. Table 1 shows how these relations may be encoded between task A2, composed of actions a21 and action a22 with a

```

(:task A2
:parameters ()
(:method A2
:precondition ()
:tasks ((:inline ()
      (and (assign (start A2) ?start)
            (assign (end A2) ?end)))
      (a21)
      (a22))))
(a)

(:durative-action b
:parameters()
:duration (= ?duration 1)
:condition()
:effect(and (assign (start b) ?start)
            (assign (end b) ?end)))
(b)

```

Figure 7: Generating temporal landmarks both for a task (a) and for an action (b). These landmarks are treated as fluents but they really represent time points of the underlying Simple Temporal Network underlying the plan, enabling the posting of additional constraints over them.

```

(:task A3
:parameters ()
(:method A3
:precondition (...)
:tasks ((= ?start (start A2)) (b))))

```

Figure 8: Recovering a temporal landmark in order to define a synchronization scheme in which action b starts exactly at the same time that task A2

duration of 1 time unit each, and action b with a duration of 5 time units. These constraints posted on start and end points of tasks are also inherited by its corresponding subtasks.

Timed initial literals

Timed initial literals, as defined in PDDL 2.2 (Edelkamp & Hoffmann 2004) are also easily supported by SIADEX to represent timed exogenous events, that is, events that are produced (and possibly repeated) along the timeline outside of the control of the planner. In fact, SIADEX uses a generalization of timed initial literals to allow them to appear regularly along the timeline. Timed initial literals in SIADEX appear either like `(between <time1> and <time2> <literal>)` to represent that `<literal>` is true from time point `<time1>` to time point `<time2>`, or `(between <time1> and <time2> and every <shift> <literal>)` to represent that `<literal>` appear regularly at intervals given by `<shift>` along an infinite timeline. For example

```

(between "8:00:00" and "20:00:00"
 and every "24:00:00" (daytime))
(between "22:00:00" and "8:00:00"
 and every "24:00:00" (nighttime))

```

represents that the literal `(daytime)` is true between a time point fixed at “8:00” and the time point fixed at

“20:00” and that this is repeated every 24 hours. The literal `(nighttime)` behaves similarly.

These intervals in which a literal is true are represented as a temporal skeleton underlying the temporal plan as fixed time points with an absolute temporal reference to `TR`, the time point that fixes the absolute beginning of the STN. Since they may appear several times along the timeline, they also represent a choice point and, therefore, a backtracking point during the satisfaction of the preconditions of an action (either “at-start” or “over-all”) (Figure 9).

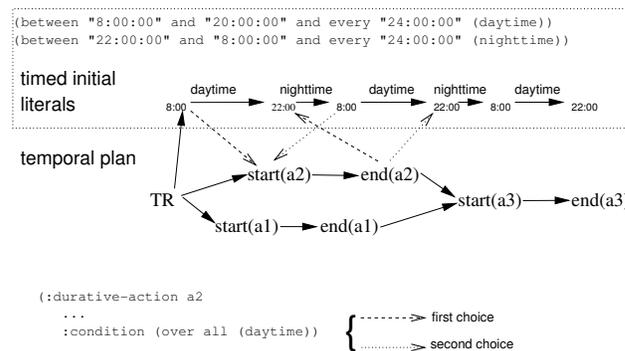


Figure 9: Timed initial literals deployed over a STN and its relation with a temporal plan. Precondition `(daytime)` of action a2 may have multiple satisfiers amongst the timed initial literals.

Boosting constraint propagation

The constraint propagation engine used in SIADEX is the algorithm PC-2 (Dechter 2003) that is sketched in Figure 10. This is an incremental propagation method, very useful in planning problems, where constraints are posted increasingly as the problem is being solved. Propagation is needed in SIADEX for two reasons. The main one is to check the consistency of the underlying STN and the other one is to schedule actions along the timeline. Although it is a very efficient algorithm (it is $O(|X|^3)$ where X is the set of time points of the underlying STN) it still requires a high computational effort in large plans in the order of hundreds or thousands of actions. This overload comes from step 4c in Figure 10, where, once a constraint has been modified, all the time points and their connections with the just modified constraint are revised. The fact is that many of these revisions could not be necessary if the involved time points are causally independent and, therefore, their relation would not affect the final solution.

It is well known that the use of additional knowledge may speed up constraint propagation with structural information able to prune unnecessary propagation effort (Yorke-Smith 2005). Furthermore, cause-effect relationships between its actions or the plan rationale (Wilkins 1988), is a source of structural information that might be used to improve the efficiency of the underlying search processes in planning problems (Helmert 2004). Therefore, we propose a simple, but effective, improvement of PC-2 to reduce unnecessary con-

Allen's relation	SIADEx encoding	SIADEx output
<p>A2 BEFORE b</p>	<code>((> ?start (end A2)) b)</code>	
<p>A2 MEETS b</p>	<code>((= ?start (end A2)) b)</code>	
<p>A2 OVERLAPS b</p>	<code>(and (> ?start (start A2)) (< ?start (end A2)) (> ?end (end A2)) b)</code>	
<p>A2 DURING b</p>	<code>(and (< ?start (start A2)) (> ?end (end A2)) b)</code>	
<p>A2 STARTS b</p>	<code>((= ?start (start A2)) b)</code>	
<p>A2 FINISHES b</p>	<code>((= ?end (end A2)) b)</code>	
<p>A2 EQUAL b</p>	<code>(and (= ?start (start A2)) (= ?end (end A2)) b)</code>	

Table 1: Encoding all Allen's relations between task A2 and action b. In all the cases action b has a duration of 5 time units except in the equal relation, whose duration is 2 time units.

• **Propagate-Temporal-Constraints-PC2(II)**

1. Let $\Pi = \{a_1, a_2, \dots, a_n\}$ be a temporal plan and $R = (X, D, C)$ a Simple Temporal Network on top of which Π is built, such that X is the set of temporal points (given by start and end points of all of the actions in Π), D is the domain of every variable, that is, $[0, +\infty)$, and C is the set of all the temporal constraints posted in Π
2. Let (i, j) the time points affected by the last posted constraint in C .
3. Let $Q \leftarrow \{(i, k, j), 1 \leq i < j \leq |X|, 1 \leq k \leq |X|, k \neq i, k \neq j\}$
4. while $Q \neq \emptyset$
 - (a) Select and delete a tuple (i, j, k) from Q
 - (b) $C_{ij} = \text{Revise}(i, j, k)$
 - (c) if C_{ij} has changed after the call to $\text{Revise}(\cdot)$ then
 - foreach $l, 1 \leq l \leq |X|, l \neq i, l \neq j$
 - i. $Q \leftarrow Q \cup \{(l, i, j)(l, j, i), \}$

• **Revise(i,j,k)**

1. $C_{ij} = C_{ij} \cap (C_{ik} \circ C_{kj})$

Figure 10: The incremental algorithm PC2 for constraint propagation in STNs

	P1	P2	P3	P4	P5	P6	P7	P8
Sequential	8	11	12	14	19	22	27	32
Parallel	40	55	60	70	95	110	135	160

Table 2: Experiments "Sequential" and "Parallel" and the sizes of plans, that is, the number of actions obtained for every problem instance out of 8 different problems.

straints propagations by exploiting the causal structure of the plan and to boost its performance.

Algorithm PC-2 propagates constraints between all of the actions in a plan, causally dependent or not. This produces a large amount of information that is mostly unnecessary regarding the final solution of a problem. In a general constraint satisfaction framework, this process is enough to ensure a correct propagation of constraints and to detect inconsistencies. However, taking into account that it is being used in a planning framework, the completeness of the algorithm only depends on a correct constraint propagation through actions that have an explicit temporal relation, that is, they have a cause-effect relationship or that have a deadline or a landmark between them. Thus, a new propagation schema is proposed, named PC2-CL (Figure 11), that extends PC2 with an analysis of the causal structure of the plan to eliminate unnecessary propagations. In order to do that, step 4(c)i only includes in the propagation queue Q those time points belonging to an action that have an explicit temporal constraint either because there is a causal link with the action involved in the new constraint C_{ij} or because there is a deadline or a landmark between them.

• **Propagate-Temporal-Constraints-PC2-CL(Π)**

1. Let Π be a temporal plan and $R = (X, D, C)$ its Simple Temporal Network
2. Let (i, j) the time points affected by the last posted constraint in C .
3. Let $Q \leftarrow \{(i, k, j), 1 \leq i < j \leq |X|, 1 \leq k \leq |X|, k \neq i, k \neq j\}$
4. while $Q \neq \emptyset$
 - (a) Select and delete a tuple (i, j, k) from Q
 - (b) $C_{ij} = Revise(i, j, k)$
 - (c) if C_{ij} has changed after the call to $Revise(.)$ then
 - foreach $l, 1 \leq l \leq |X|, l \neq i, l \neq j$
 - i. if l belongs to an action with a causal link towards the action that owns either i or j or l is a time point with a deadline or landmark associated to i or j then
 $Q \leftarrow Q \cup \{(l, i, j), (l, j, i), \}$

Figure 11: The modified version of PC2 that accounts for the existence of causal links

This extension obviously implies less propagations and a lower number of calls to the procedure $Revise(.)$ and, given that the representation of a list of temporal causal links is very simple, it is not expected to produce a computational overload. What is clear is that the whole improvement depends largely on the density of causal links, i.e., the greater the number of causal links the lower the gain. In addition to this, PC2-CL is provably correct since making use of the temporal causal links, it is guaranteed that constraints are propagated only to those actions that are affected by a cause-effect relationship or have an explicit constraint and not to unnecessary independent actions.

Some experiments

In order to empirically check the efficiency of SIADEX with PC2-CL four different experiments are performed. Two of them, named “Sequential” and “Parallel” are extracted from a domain of electronic tourism in which a planner is used to find plans of visit adapted to the preferences of a certain tourist (Fernández, Sebastiá, & Fdez-Olivares 2004). The third one, named “Infoca”, is extracted from (Fdez-Olivares *et al.* 2006) and it consists of the application of a planner to the design of real forest fire fighting plans in Andalusia (Spain). The last experiment, named “Zeno”, is framed in the Zeno domain of the 2002 international Planning Competition (Fox & Long 2002) and is performed over the hard instances of temporal and numeric problems. The size of the plans obtained for every experiments are shown in Tables 2, 3 and 4.

The experimental results of Figure 12 show much less propagation effort in PC2-CL measured as the number of calls to the $Revise(.)$ procedure. And, additionally, in terms of CPU time (Figure 13), PC2-CL is much faster in all the experiments except some instances of the “Sequential” domain in which, since the time is very low and plans are very small, the time devoted to check causal links might exceed

P1	P2	P3	P4	P5	P6	P7	P8
55	92	98	106	114	122	130	167
P9	P10	P11	P12	P13	P14	P15	P16
200	233	241	248	268	306	339	372

Table 3: Experiment “Infoca” and the sizes of plans, that is, the number of actions obtained for every problem instance out of 16 different problems.

P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
73	107	156	171	272	296	297	318	373	245
P11	P12	P13	P14	P15	P16	P17	P18	P19	P20
237	240	262	255	298	290	287	304	277	284

Table 4: Experiment “Zeno” and the sizes of plans, that is, the number of actions obtained for every problem instance out of 20 different problems extracted from the hard instances of temporal and numeric problems in IPC 2002 (Fox & Long 2002).

the benefit of using PC2-CL, achieving thus a worse performance. However, in the remaining cases, the real example of “Infoca” and the hard temporal problems of “Zeno” PC2-CL outperforms PC2.

And just to give a relative idea of the overall performance of SIADEX, Figure 14 shows a comparison of the CPU time needed by SIADEX and SHOP2 (Nau *et al.* 2003), the best known HTN planner to date¹ to solve all the Zeno problems.

Related work

The use of STNs in planning frameworks is not new and has also been widely addressed in the literature. In a hierarchical planning setting, one of the earlier works is OPlan (Tate, Drabble, & Kirby 1994) in which many of the temporal constraints have to be explicitly encoded in the STN. SIADEX also encodes these constraints but only those that depend on high level goals (like deadlines or landmarks) are explicit and the remaining ones are implicitly encoded in the causal structure of the plan without an additional effort of the domain modeler to make them explicit. SHOP (Nau *et al.* 2003) does not use STNs but temporal constraints have also to be explicitly encoded amongst the effects and preconditions of operators in the Multi-Timeline Preprocessing scheme (MTP), requiring an important effort to write temporal domains. However, the use of causal links, either empty or not, to propagate constraints between actions and protect the achievements of literals in the current state seems to be fully equivalent to MTP. The main difference is that SHOP with MTP produces a unique schedule that coincides with the schedule of SIADEX’ STN with the earliest execution time for every action, but SIADEX is able to handle

¹Both planners running on the same machine, a Pentium IV 3 GHz, 1GB Ram. SIADEX is compiled from C++ and SHOP2 is written in Lisp running on Allegro CL. SIADEX was running a translation of the same domain used by SHOP2 in the IPC 2002 (Fox & Long 2002) with an equivalent HTN expressive power and the same constraints.

deadlines and landmarks and to obtain different schedules according to other criteria (i.e., latest execution time). Ixtet (Laborie & Ghallab 1995) also uses STNs with an equivalent expressive power and temporal cause effect relationships are encoded by means of two simple predicates named *events* (that represent an instantaneous change of the world) and *assertions* (that represent the persistence of some attributes along the timeline).

On the other hand, the use of different sources of knowledge to restrict temporal constraint propagation to a subnetwork of the former STN in order to obtain a better performance has also been studied like in Ixtet (Ghallab & Vidal 1995) or in PASSAT (Yorke-Smith 2005). These works and the one presented in this paper may have different performance results on the same tests and a comparative study might be useful, but it is worth noting that the approach in (Yorke-Smith 2005), in which constraints are propagated taking into account the HTN structure of the plan, is particularly relevant since it is an orthogonal approach to this one and it seems that they might be combined to obtain even more efficient results.

Conclusions

In summary, this paper has presented several valuable temporal extensions of an HTN planner that allow to cope with a very rich temporal knowledge representation like temporal causal dependencies, deadlines, temporal landmarks or synchronization schemas and timed initial literals. These capabilities have been found to be of extreme necessity during the application of SIADEX to the research contract NET033957 with the Andalusian Regional Ministry of Environment for the assisted design of forest fighting plans (de la Asunción *et al.* 2005; Fdez-Olivares *et al.* 2006) and, up to author's knowledge, no other HTN planner has these capabilities for handling temporal constraints. In addition to this temporal expressive power, SIADEX also shows an excellent performance compared to other well known HTN planner like SHOP.

References

Ai-Chang, M.; Bresina, J.; Charest, L.; Chase, A.; jung Hsu, J. C.; Jonsson, A.; Kanefsky, B.; Morris, P.; Rajan, K.; Yglesias, J.; Chafin, B. G.; Dias, W. C.; and Maldague, P. F. 2004. Mapgen: Mixed-initiative planning and scheduling for the mars exploration rover mission. *IEEE Intelligent Systems* 8–12.

Allen, J. 1983. Maintaining knowledge about temporal intervals. *Comm. ACM* 26(1):832–843.

de la Asunción, M.; Castillo, L.; Fdez-Olivares, J.; García-Pérez, O.; González, A.; and Palao, F. 2005. Siadex: an interactive artificial intelligence planner for decision support and training in forest fire fighting. *Artificial Intelligence Communications* 18(4).

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

Dechter, R. 2003. *Constraint processing*. Morgan Kaufmann.

Edelkamp, S., and Hoffmann, J. 2004. The language for the 2004 international planning competition. <http://ls5-www.cs.uni-dortmund.de/edelkamp/ipc-4/pddl.html>.

Fdez-Olivares, J.; Castillo, L.; García-Pérez, O.; and Palao, F. 2006. Bringing users and planning technology together: experiences in siadex. In *Sixteenth International Conference on Automated Planning and Scheduling, ICAPS*.

Fernández, S.; Sebastiá, L.; and Fdez-Olivares, J. 2004. Planning tourist visits adapted to user preferences. In *Workshop on Planning and Scheduling: Bridging Theory to Practice, European Conference on Artificial Intelligence*.

Fox, M., and Long, D. 2002. Domains of the 3rd. international planning competition. In *Artificial Intelligence Planning Systems (AIPS 02)*.

Ghallab, M., and Vidal, T. 1995. Focusing on the subgraph for managing efficiently numerical temporal constraints. In *FLAIRS-95*.

Gil, Y., and Blythe, J. 2000. PLANET: A shareable and reusable ontology for representing plans. In *AAAI 2000 workshop on representational issues for real-world planning systems*.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In *International Conference on Automated Planning and Scheduling, ICAPS*.

Laborie, P., and Ghallab, M. 1995. Planning with sharable resource constraints. In *IJCAI'95*, 1643–1649.

McIlraith, S. A. 2000. Integrating actions and state constraints: A closed-form solution to the ramification problem (sometimes). *Artificial Intelligence Journal* 87–121.

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

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

Oddi, A.; Cesta, A.; Policella, N.; and Cortellessa, G. 2002. Scheduling downlink operations in mars express. In *Proceedings of the 3rd International NASA Workshop on Planning and Scheduling for Space*.

Tate, A.; Drabble, B.; and Kirby, R. 1994. O-PLAN2: An open architecture for command, planning and control. In Zweben, M., and Fox, M., eds., *Intelligent scheduling*. Morgan Kaufmann.

Weld, D. 1994. An introduction to least commitment planning. *AI Magazine* 15(4):27–61.

Wilkins, D. E., and desJardins, M. 2001. A call for knowledge-based planning. *AI Magazine* 22(1):99–115.

Wilkins, D. E. 1988. *Practical planning: Extending the classical AI planning paradigm*. Morgan Kaufmann.

Yorke-Smith, N. 2005. Exploiting the structure of hierarchical plans in temporal constraint propagation. In *Proceedings of AAAI'05*.

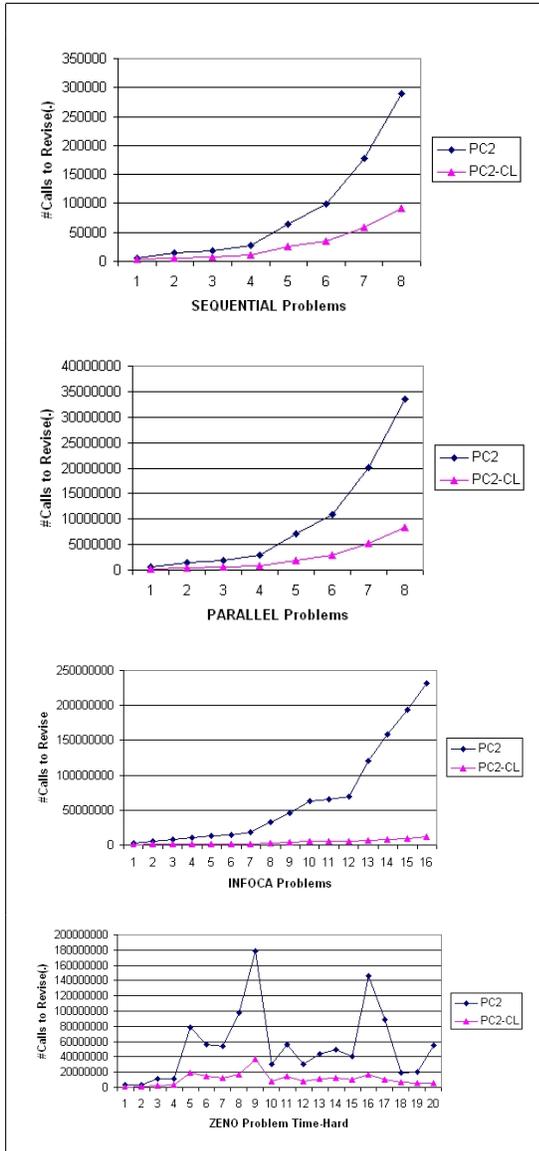


Figure 12: Number of calls to the procedure *Revise(.)* in PC2 versus PC2-CL for all the instances of every problem.

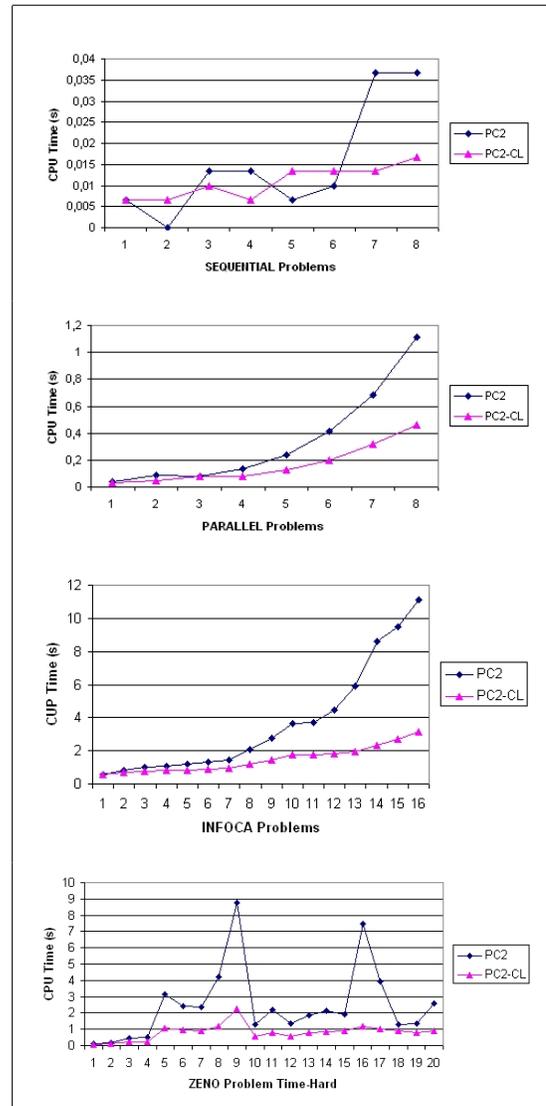


Figure 13: Compared CPU time in PC2 and PC2-CL in all the instances of the problems.

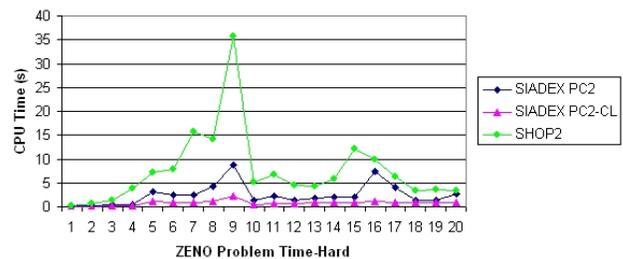


Figure 14: Compared CPU time between SIADEX and SHOP2 in the hard instances of ZENO (temporal+numeric).