**Abstract**

This paper presents an approach to handle and satisfy time constraints when a temporal plan is to be instantiated over a real time line. This approach considers two types of constraints:

- **Domain/Environment constraints.** We introduce the concept of delay in durational contexts. In real domains, the actual duration of a temporal action is highly dependent on the timing at which actions are actually executed. A lot of external elements may cause an increase in the action duration, that is, the end time of the action execution is delayed with respect to its standard expected duration.

- **User time requirements like imposing a specific start/end time for the plan execution, a given maximum duration or deadlines for some actions in the plan.**

The main contribution of this paper is how to handle and satisfy time constraints when temporal plans are to be executed in a temporal setting.

**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 resolution of temporal planning problems, as the introduction of durative actions in PDDL2.1 (Fox & Long 2003) or timed initial literals in the most recent PDDL2.2 (Edelkamp & Hoffmann 2004).

However, handling time in real planning problems is much more than dealing with durative actions or incorporating temporal constraints. Time plays an important role because it is a source of imprecision and uncertainty. In this direction we can find in the literature several approaches to handle an extended model of durative actions, as the introduction of uncertainty in the time consumption of actions ((Biundo, Holzer, & Schattenberg 2004), (Bresina et al. 2002)).

Time also plays an important role when plans are to be executed in a specific temporal setting. The user might want the plan not to finish later than time $t$, or start at time $t'$ as earliest or to have a maximum duration $d$. These constraints may be considered during planning if the planner is able to handle them and the language used for domain modelling accounts for it.

More relevant are the restrictions derived from the own domain or environment. In this category we can find the typical delays that affect the execution of actions. Actions do not always take the same time to execute but real durations depend on the timing at which actions are actually executed, which may be affected by a longer or shorter delay. A lot of elements condition the real duration of actions: moving from one place to another is more costly at peak hours, loading objects in a container depends on the number of resources (hoists, humans) available when the load is carried out and a space operation might last differently whether it is carried out in daytime or at night.

In this paper we present an approach to modify a given temporal plan that is to be instantiated over a time line in a particular temporal setting. Firstly, the plan is “extended” to take into account the delays that may affect actions during the plan execution. Secondly, the plan is adapted to fulfill the restrictions the user imposes on the plan execution. This latter task is achieved through a CSP resolution.

The paper is organized as follows. The following section motivates the use of delays in temporal environments and introduces some concepts to define action delays. Section *Approach overview* sketches the overall working schema of our approach. Section *Applying delays* presents the algorithm used to apply delays to a given temporal plan. Section *Plan scheduling as a CSP* outlines the steps to follow to encode and fulfill the temporal requirements specified by the user. Section *Experimental Results* presents some results that show different schedulings of a temporal plan over a time line and how this information can be exploited to satisfy the time restrictions. Finally, the last section concludes and outlines some future work.

**Delays in a temporal setting**

In this section we give an exposition to motivate the introduction of action delays in temporal planning. The key issue in the proposed approach is that actions or activities seldom last as initially planned (expected) in daily life.

Usually, when we refer to a "delayed action" we mean the action execution does not finish at the scheduled time. There
are several reasons that can be a cause of delay:

- the action execution does not start at the expected time and therefore it does not finish at the scheduled time, i.e. the action is delayed due to a later happening of the starting time. In this case there are no variations in the duration and the action is simply shifted forward in time. These delays are usually produced by the causal relationships the action holds with other actions that, in turn, are also delayed (a chain-delay effect).
- the action execution starts at its scheduled time but the duration is longer than expected. These delays are normally caused by external factors that affect the action execution. Although these external conditions can be predicted in advance they are not static causes that happen forever but only rather at specific times along the action execution.
- both reasons stated in the above items. Delays in the action duration give rise to a later ending time and, consequently, dependent actions start later and so on.

In PDDL 2.1, the duration of the actions is dependent on the own action parameters: the origin, the destination, the truck, the driver etc. in, for example, a classical logistics or transportation domain. Variable durations can be modelled through the use of functions (level 3 durative actions).

Let’s now add one more dose of reality. We all know that driving at night is harder than driving during daytime; driving at peak-hours is slower than driving at regular hours; driving under the rain is also slower; a load/unload operation that is carried out at lunch time will take longer as the number of available resources is half the usual. Under all these situations the standard action duration (under normal conditions) is increased by external factors to the action itself.

In principle, delays are produced by generic causes that affect action durations equally, no matter the timing at which actions are executed. For instance, driving with a heavy traffic flow implies to move slower, regardless if it happens at 9 a.m. or at 9 p.m.

However, many common causes of delay typically occur at particular times or time intervals. Peak-hours usually occur early in the morning or late in the afternoon after work. A lack of human resources may likely occur at lunch time. Therefore, an action may be affected by different delays along its execution depending on the time intervals over which the action is executed. If action drive-truck T1 A B starts at 8 a.m. and it takes two hours under normal conditions, the first driving hour may be affected by a heavier traffic flow than the second hour. Unloading a truck that usually takes one hour may be longer because the action starts at 11.30 a.m. and there are fewer workers from 12.00 a.m. on.

A delay is the extra-time to put in the action duration over a time interval due to an external factor. Let’s define \( \alpha_{<\text{cause}>} \) the delay caused by any reason. Figure 1 shows two driving actions affected by different delays at different times. Road connecting B to C is a highway so heavy traffic never occurs. And road connecting A to B is located in the west part so darkness comes later.

![Figure 1: Causes of delays in two drive actions](image)

The value of \( \alpha_{<\text{cause}>} \) can be specified as:

1) a fixed value (i.e. 5 minutes)
2) a value proportional to the standard action duration (i.e. 5% of the standard duration or equivalently a fraction of 5/100 in the interval [0,1])
3) a value given by a formula (i.e. speed * weight – distance)

Additionally, we can distinguish between causes of delays that are known in advance and dynamic causes that appear during the plan execution and cannot be predicted at planning time. In the first group we can find: the traffic flow in the roads, the existence of works in the roads, the happening a mass event in the city, a shortage of workers, the night fall, etc. Among the second group we can find: the appearance of rain, traffic-lights stop working and any kind of unexpected event. In the rest of the paper we will focus just on the first group of delays.

### Delays over time

As we mentioned in previous section, the value of \( \alpha_{<\text{cause}>} \) can be specific for each action if \( \alpha_{<\text{cause}>} \) is defined in terms of some of the action parameters. We will denote by \( \alpha_{<\text{cause},a_i>} \) the value that results from computing \( \alpha_{<\text{cause}>} \) for \( a_i \). \( \alpha_{<\text{cause},a_i>} \) represents the increase fraction that has to be applied on \( a_i \) when the \( <\text{cause}> \) of delay is found during the execution of \( a_i \).

The total delay of an action will depend on its start time of execution so the same action will have different delays at different starting execution times. In order to compute the delay of an action execution we have first to define the delays that affect the action over the time line. Thus, it is necessary to represent the delay patterns of actions over a time window. Typically, the range of the time window would be the overall plan duration but it is also possible to specify a repetitive delay pattern over sequential time windows along the plan duration. In this paper we study a general case where the time window covers the range of a whole day from 0 to 24 hours but the analysis can also be applied to a different temporal scope.

For each action \( a_i \) and time window \([t_b, t_e]\) (in our case \( t_b = 0, t_e = 24 \)) we define a set of subintervals over \([t_b, t_e]\) and specify a delay associated to each of them. This set of subintervals along with their corresponding delays are called delay patterns.
Timing intervals of an action $a_i$. When $a_i$ is executed over a timing interval, the actual duration of $a_i$ is augmented with the delays indicated in the timing interval.

There can be more than one cause of delay over a timing interval. We denote by $\alpha_{c[1][a_i,t_1,t_2]}$, $\alpha_{c[2][a_i,t_1,t_2]}$, $\ldots$, $\alpha_{c[n][a_i,t_1,t_2]}$ the values of the delays produced by causes $c_1, c_2, \ldots, c_n$ on action $a_i$ over timing interval $[t_1,t_2]$. That is, the result of $\alpha_{\langle c_k,a_i \rangle}$ is totally delayed over $[t_1,t_2]$ due to all the delay causes produced over such a timing interval.

The definition of delays over timing intervals for an action $a_i$ can be interpreted as a continuous function over the time line. This scheme would also allow to specify delays as a probability distribution. Figure 2 shows the timing intervals of the action drive-truck $T1 \ A \ B$ for $\alpha_{\text{heavy-traffic}}$ and $\alpha_{\text{night}}$. Thus, heavy traffic only affects timing interval [8,10] and the action is delayed $0.4 \times dur(a_i)$; and driving at night produces a retard over the timing interval [20,7] of $0.2 \times dur(a_i)$.

**Approach overview**

Nowadays, some planners have been extended to handle timed initial literals and derived predicates (LP-G-TD (Gerevini, Saetti, & Serina 2004), SATPLAN (Kautz 2004)). In particular, timed initial literals may be used to simulate the allocation of a plan execution at a certain time in the day. For example, we can assume that time point 0 corresponds to 8:00 and that each time point represents one hour. Obviously, a planner able to handle this functionality must be modified to capture these exogenous events so that actions that need timed initial literals can only be executed after the time instant the literal becomes true.

A similar situation arises when a planner must consider delays in actions. The duration of an action may be different depending on the timing interval over which it is executed. The most important implication of using delays in a domain is that the standard duration of a plan may no longer be its actual duration in practice. Therefore, the objective of a temporal planner that works with delays is not only to obtain a temporal plan but also to indicate at which time the plan should start its execution in order to keep its actual duration as close as possible to the standard one. In other words, we want the plan to be affected the least possible by the delays of the environment.

Dealing with delays when the plan is being computed adds a great complexity to the planning process, mainly due to the fact that the actual duration of each action is unknown until it is allocated in the time line. One alternative would be to compute in advance the duration of each single action at each time point. However, although this is a polynomial process, the result would be a prohibitive number of different instantiated actions (one instance for each different possible duration) which would cause a blow up in the planning process. Moreover, computing accurate estimates of the plan duration to define heuristics would be much harder as the exact duration of actions is unknown.

Therefore, we can conclude it is more efficient not to consider the plan scheduling (allocation of a plan in time) into the planning process but in a separate post-process. Our proposal consists in obtaining a plan from a temporal planner and then applying a post-planning process that schedules the plan over a time line. This process computes the exact extended duration of the plan due to delays and then attempts to re-order and re-allocate actions to meet the temporal requirements of the user. Next, we outline in more detail the working schema that our approach follows:

1. **To obtain a temporal plan by using an existing temporal planner.** In this stage, we use an existing temporal planner to solve the planning problem without delays. The plan returned by the planner will be a partially-ordered set of actions, where each action is assigned a start time point and a duration. The plan is then allocated in time according to the user restrictions on the start or finish time of the plan, or it is initially allocated at any time if no specifications are given.

2. **To compute the actual duration of the plan when actions are allocated in their corresponding timing interval.** Once we have allocated the plan in time, we know the timing intervals over which actions are executed. We can compute now the actual duration of this plan by using the algorithm of applying delays shown in next section. This process calculates the plan duration based on the actual duration of each single action. The result will be an instantiated plan with extended duration that most likely will not satisfy the user requirements.

3. **Checking the user temporal restrictions.** In this stage, we check whether the user requirements hold in the current plan or not. In this paper, we will consider the following temporal requirements:
   - restrictions on the start/end time of the plan (Cresswell & Coddington 2003)
   - restriction on the plan duration
   - restrictions on the start/end time of actions in the plan

If the user restrictions do not hold in the plan, the next stage is executed.
4. To modify the plan to meet some temporal requirements specified by the user. When a plan does not meet the user requirements, the plan is converted into a CSP and actions are reordered, if possible. The goal of this stage is to re-allocate actions in time so as to minimize the overall delay while preserving the plan semantics.

The next two sections are devoted to stages 2 and 4 of the working schema.

Applying delays

In this section we show the calculation process to compute the extended duration of a temporal plan given the action delays at their timing intervals. The extended (real) duration of a parallel plan $P$ composed of several sequences of actions will be the extended duration of the longest sequence.

Firstly, the plan is allocated in the time line according to the user restrictions on the start or end time of the plan. If they cannot be satisfied we will only consider the restriction on the start time. We show here how to compute the extended duration of a single action; the same process is then repeated for every action and the final extended plan duration is calculated as indicated above.

Let $a_k$ be a durative action which start time and standard duration are denoted by $beg_k$ and $d_k$ respectively. $a_k$ starts at timing interval $[t_0, t_1]$ and finishes at timing interval $[t_{n-1}, t_n]$ (see case 0 in Figure 4). The basic procedure of the algorithm works as follows: for action $a_k$ and timing interval $[t_i, t_j]$, find out “how much” of the duration of $a_k$ over $[t_i, t_j]$ actually corresponds to the action execution and “how much” corresponds to causes of delays in the timing interval; then subtract the action duration from $d_k$ and repeat again for the next timing interval. Figure 3 shows the algorithm for applying delays to a single action $a_k$.

The expression $(t_{p+1} - \max(t_p, beg_k))$ in the if condition always returns the range of timing interval $[t_p, t_{p+1}]$ except the first time if $a_k$ does not start exactly at $t_p$. The expression $(d_k + d_k \times \alpha_{[a_n, t_{p}, t_{p+1}]}$) denotes the extended duration of $a_k$ over timing interval $[t_p, t_{p+1}]$. If this duration goes beyond $t_{p+1}$ then we compute “how much” of $(t_{p+1} - \max(t_p, beg_k))$ (duration of $a_k$ over $[t_p, t_{p+1}]$) actually corresponds to the action execution ($\Pi$), such that $\Pi + \Pi \times \alpha_{[a_n, t_{p}, t_{p+1}]} = t_{p+1} - \max(t_p, beg_k)$ (cases (1), (2), (3) and (4) in figure 4). $d_k$ is then updated and the process is repeated for the next timing interval while $d_k > 0$. If the extended duration completely falls within $[t_p, t_{p+1}]$ (case 5 in figure 4) then end$_k$ is updated by adding this extended duration to the range of the previous timing intervals (case 6 in figure 4). The final result will be an extended $a_k$ duration, as it can be observed in case 6 of figure 4.

Plan scheduling as a CSP

This step is executed when the scheduled plan $P$ does not fulfill the user requirements. We build a CSP that encodes:

- information in plan $P$
- all possible planning alternatives in $P$ (different ways of achieving a literal with actions in $P$)
- the different durations actions in $P$ can take on according to their timing intervals
- user constraints

It is important to remark that, unlike other approaches (Do & Kambhampati 2001) our CSP does not encode the planning problem but a scheduled plan in time plus all the necessary information to modify the plan and make it accomplish the user requirements. **MOVERLO A LA INTRODUCCION JUNTO CON REFERENCIA A SATPLAN**

CSP specification

Let $P$ be a partially-ordered set of actions $\{a_0, a_1, \ldots, a_n\}$: $a_0$ and $a_n$ are the initial and final fictitious actions that represent the start and end time of $P$, respectively. We will use $s_i$ and $e_i$ to denote the start and end time of an action $a_i$, and $t_i$ to refer either $s_i$ or $e_i$ indistinctly. Let $X$ be the set of variables $X = \bigcup_{i=0}^{n} s_i \cup \bigcup_{i=0}^{n} e_i$. For the fictitious initial and final actions, $s_0 = e_0$ and $s_n = e_n$. Time is modelled by $\mathbb{R}^+$ and their chronological order. Particularly, the list of possible values for an end time point...
is $D_{ai} = \{0...24\} \in \mathcal{R}^+$ and for a start time point is $D_{ai} = \{0, v_1, v_2, ..., v_n, 24\} \in \mathcal{R}^+$ where $v_i$ denote the left/right extreme points of the timing intervals. We distinguish three types of constraints $C$: planning constraints, timing interval constraints and user constraints.

This specification gives rise to a TCSP (Ghallab, Nau, & Traverso 2004) as the problem involves a set of variables $X$ having continuous domains ($\mathcal{R}^+$), each variable represent a time point and the three types of constraints are encoded as a set of unary and binary constraints (see below).

(A) Planning constraints. We show here how to encode planning relationships into temporal constraints.

- **Causal link.** We denote a causal link (Penberthy & Weld 1992) between actions $a_i$ and $a_j$ as $(a_i, a_j, p)$ where $p$ is the literal that $a_i$ produces for $a_j$. The translation of a causal link into a temporal constraint has four variants, depending on literal $p$ is produced at start or end of $a_i$ and required at the start or end of $a_j$. These four variants are encoded as binary constraints: $s_i \leq s_j$, $s_i \leq e_j$, $e_i \leq s_j$ or $e_i \leq e_j$. We will use the general temporal constraint $t_i \leq t_j$ to refer to any of the four variants.

The different planning alternatives (ways of producing literal $p$ with actions in $P$) are encoded as a disjunctive temporal constraint:

$$\bigvee_{v_{a_i}/p \in \text{add}(a_i)} t_k \leq t_j \tag{1}$$

- **Threat.** Let’s suppose $a_i$ threatens causal link $(a_i, a_j, p)$. The set of disjunctive constraints to model the demolition and promotion choices are:

$$\bigwedge_{v_{a_i}/p \in \text{del}(a_i)} t_k < t_i \lor t_k > t_j \tag{2}$$

In the case of an overall condition the disjunctive constraint would be $\bigwedge_{v_{a_i}/p \in \text{del}(a_i)} t_k < t_i \lor t_k > e_j$.

We classify planning constraints into three different categories:

1. **Safe causal link.** When $a_i$ is the only way to produce literal $p$ and there is no action, except may be $a_j$, that deletes $p$. This implies $a_i$ is the only producer so the causal link is unmodifiable and there are no threats over $(a_i, a_j, p)$. Therefore, a safe causal link is encoded as temporal constraint that represents a causal link.

2. **Hard causal links.** When $a_i$ is the only way to produce $p$ and it exists at least one action that deletes $p$. This implies $a_i$ is the only producer so the causal link is unmodifiable and there is no action, except may be $a_j$, that deletes $p$. Therefore, a hard causal link is encoded as a temporal constraint that represents the causal link plus a set of disjunctive constraints to avoid threats over such a causal link.

3. **Weak causal links.** When there are several producer actions of $p$ for $a_j$. A weak causal link is encoded by using the disjunctive temporal constraint of the planning alternatives plus a set of disjunctive constraints to avoid threats on each possible causal link. The combination of these constraints is encoded as:

$$\bigvee_{\forall a_i/p \in \text{add}(a_i)} t_i \leq t_j \land (\forall a_k/p \in \text{del}(a_k) (t_k < t_i \lor t_k > t_j)) \tag{3}$$

(B) Timing intervals constraints. These constraints represent the different durations an action $a_i$ can take on depending on the timing interval where $a_i$ starts its execution. For example, assuming we have $\alpha[a_3, 3, 5]$ and $\alpha[a_4, 5, 9]$, the final duration of $a_i$ will be different if $s_i \in [3, 5]$ or $s_i \in [5, 9]$. Actually, the final duration also depends on the specific time point within the corresponding timing interval but we will simplify the modelling by just assuming that the extended duration of $a_i$ is the same when $s_i$ falls at any point within the same timing interval\(^1\). The timing interval constraints are encoded as:

$$\forall a_i \in P \bigvee_{\forall [t_p, t_{p+1}] \in [s_i, s_i]} t_p \leq s_i \leq t_{p+1} \land e_i = s_i + d \tag{4}$$

where $d$ is calculated by applying the algorithm of figure 3. We restrict the plan to be executed before $t_e = 24$.

(C) User constraints. Constraints on the start/end time of the plan are expressed as $s_0 \geq s_0$, $e_n \leq e_n$. The same specification is used to represent restrictions on the start/end time of the actions in the plan ($t_i \leq u_i \leq v_i$). A limit in the plan duration is encoded as $e_n = s_0 + d_k$.

**Example.** Figure 5 represents a scheduled plan in the interval $[10, 14:30]$. It shows a plan where each action is represented by a rectangle with its preconditions (Pr) and effects (Ef). The causal links between the actions are denoted by arrows with labels on the form (type of causal link, literal). All conditions are at start except condition $v$ of $a_4$ which is overall and all add and del effects are at end.

\[\text{Figure 5: A scheduled plan over a time line}\]

The set of temporal constraints representing the planning relationships are:

- **Safe causal links:** $e_0 \leq s_2$, $e_1 \leq s_4$, $e_3 \leq s_5$, $e_4 \leq s_7$
- **Hard causal links:** $e_0 \leq s_4$, $e_2 > e_4$ (this latter constraint represents the only way to avoid the threat of $a_2$ over the hard causal link)

\(^1\)In this first approximation we only consider the left extreme of timing intervals. In a future work, we will extend this approach to work with the exact duration of each action according to the time point within the timing interval.
• Weak causal links: Let’s take, for instance, the weak causal link \((a_0, a_1, r)\). The backup producer is \(a_2\) so the planning alternatives are \(e_0 \leq s_1 \lor e_2 \leq s_1\). The possible threatener is \(a_3\). Consequently, the final encoding for this weak causal link implies one way of solving a threat over the first planning choice and the two ways of avoiding a potential threat on the second planning choice:
\[
[e_0 \leq s_1 \land (e_3 > s_1)] \lor [e_2 \leq s_1 \land (e_3 < s_2 \lor e_3 > s_1)]
\]
Similarly for the rest of weak causal links encodings.

The set of temporal constraints to represent the timing intervals and user restrictions are:

- \(s_0 \geq 10, e_n \leq 14 : 30\) (user constraints)
- \((0 \leq s_1 \leq 12 \land e_1 = s_1 + 1) \lor (12 \leq s_1 \leq 24 \land e_1 = s_1 + 2)\)
- \((0 \leq s_2 \leq 6 \land e_2 = s_2 + 1.5) \lor (6 \leq s_2 \leq 14 \land e_2 = s_2 + 2) \lor (14 \leq s_2 \leq 24 \land e_2 = s_2 + 3)\)
- \((0 \leq s_3 \leq 10 \land e_3 = s_3 + 2) \lor (10 \leq s_3 \leq 18 \land e_3 = s_3 + 1) \lor (18 \leq s_3 \leq 24 \land e_3 = s_3 + 4)\)
- \(0 \leq s_4 \leq 24\)

### Solving the CSP

After encoding the current plan into a TCSP, we have a representation of this plan joint with all the possible alternative plans we could build. The next step is to solve this TCSP in order to obtain a new plan, which satisfies the user requirements. As shown above, this is a disjunctive TCSP, with unary and binary constraints. A straightforward way of solving it is to decompose it into several non-disjunctive TCSPs (Detcher 2003). This is the approach we have adopted, but we need to distinguish between two levels of decomposition: the first level considers the disjunctive planning constraints (weak and hard) while the second level considers the disjunctive timing interval constraints. Once a non-disjunctive TCSP has been built, we apply an arc-consistency procedure in order to obtain a consistent assignment of intervals for each variable. The remainder of this section formalizes the algorithm to solve the general TCSP.

First, we need to define the intersection of intervals (Detcher 2003). Let \(T = \{I_1, \ldots, I_t\}\) and \(S = \{J_1, \ldots, J_m\}\) be two constraints representing the domain of intervals of a temporal variable \(s_i\) or \(e_i\). The intersection of \(T\) and \(S\), denoted by \(T \cap S\), admits only values that are allowed by both \(T\) and \(S\), that is, \(T \cap S = \{K_1, \ldots, K_n\}\) where \(K_k = I_i \cap J_j\) for some \(i\) and \(j\).

Our algorithm works in four stages. At any time, the algorithm can return that there is no solution when any of the variable domains becomes empty. These stages are:

1. **Dealing with the user requirements.** The first step of this algorithm consists in shrinking the given domains of the variables according to the user requirements. We distinguish three types of user requirements and each of them has a different process:
   - **Start and end of the plan:** Given a start and end of the plan constraints of the form \(s_0 \geq v_0\) and \(e_n \leq v_n\), we can shrink the domains of all the variables in the CSP as we implicitly know that \(\forall a_i \in P, s_0 \leq a_i \land e_i \leq e_n\).
   - **Duration of the plan:** This requirement cannot be used to shrink the variable domains, because it does not restrict at what time an action may or may not start. It only shrinks the makespan of the plan which is not represented in the variable domains.
   - **Start and end of one action:** Given two constraints indicating the start and end of an action of the form \(s_i \geq v_0\) and \(e_i \leq v_n\), we can shrink the domain of these variables in the same way as with the start and end of the plan.

2. **Selection of the planning disjunction.** We focus on those constraints which produce disjunctions: hard and weak causal link constraints. Let \(CL = (a_i, a_j, p)\) be a causal link.
   - Assuming that \(CL\) is a hard causal link, let \(c_j\) be a threat constraint related with \(CL\) on the form of (2). Therefore, the number of different TCSP we can build considering only \(CL\) is \(NH_j = 2^{|a_k|}\), where \(|a_k|\) is the number of actions that threaten \(CL\).
   - Assuming that \(CL\) is a weak causal link, let \(c_j\) be the constraint corresponding to \(CL\) on the form of (3). Therefore, the number of different TCSP we can build considering only \(CL\) is \(NW_j = |a_i| \times 2^{|a_k|}\), where \(|a_i|\) is the number of actions that can solve \(p\) for \(a_j\) and \(|a_k|\) is the number of actions that threaten \(CL\).

The number of different TCSP grows exponentially as the number of hard and weak causal links increases. Namely, this number is:
\[
\prod_{c_i \in HCL} c_i \times \prod_{c_i \in WCL} c_i
\]
where \(HCL\) and \(WCL\) are the set of hard and weak causal link constraints, respectively. We select one of the obtained TCSPs randomly\(^2\).

3. **Selection of the timing interval disjunction.** At this moment, we only consider the TCSP selected in the previous step. Let \(c_{ij} = \sqrt{\prod_{t_p, t_{p+1}} (t_p \leq s_i \leq t_{p+1} \land e_i = s_i + d)}\) be a timing interval constraint. Again, we can build a number of different TCSP, each of them considering a different execution interval for each action. This number is \(\prod_{N_{tai} \in P} |N_{tai}|\), where \(|N_{tai}|\) is the number of timing intervals for each action. Fortunately, this number is an upper bound of the number of the TCSP we must actually consider, as some of these combinations rule out due to the domains restriction performed in previous steps. We select a TCSP which will be solved in the next step.

\(^2\)At this moment, there is no reasoning about which of the obtained TCSP may lead to a solution with a higher probability. This will be discussed in the experiments section.
4. Arc-consistency. An arc-consistency process is applied in order to obtain the minimal domain for each variable, that is, the set of all their feasible values. The process we have implemented takes $O(n^2)$, where $n$ is the number of variables.

Example We continue with the example of the previous section. The domains of the variables are those represented in the timing interval constraints. Taking into account the user requirements, the new domains of the variables after applying the first step of the TCSP resolution process will be:

- $D_{s_0} = D_{s_0} = \{[10, 14 : 30]\}$
- $D_{s_1} = \{[10, 12], [12, 14 : 30]\}, D_{e_1} = \{[10, 14 : 30]\}$
- $D_{s_2} = \{[10, 12], [12, 14 : 30]\}, D_{e_2} = \{[10, 14 : 30]\}$
- $D_{s_3} = \{[10, 14 : 30]\}, D_{e_3} = \{[10, 14 : 30]\}$
- $D_{s_4} = D_{e_4} = \{[10, 14 : 30]\}$

As shown in the previous section, there is a number of hard and weak causal links, which in turn define different TCSP. We select the following planning disjunctions: $e_3 \leq s_2$, $e_2 \leq s_1 \land e_3 < e_2$ and $e_0 \leq s_3 \land e_1 > s_3$.

Now, we have to select a set of disjunctions from the timing interval constraints. Let’s assume we select the following ones: $(0 \leq s_1 \leq 12 \land e_1 = s_1 + 1)$, $(0 \leq s_2 \leq 6 \land e_2 = e_2 + 1.5)$, $(0 \leq s_3 \leq 10 \land e_3 = s_3 + 2)$ and $0 \leq s_4 \leq 24$.

In this case, it is obvious that the obtained TCSP is inconsistent, as constraint 2 cannot be satisfied ($D_{s_2} = \{[10, 12], [12, 14 : 30]\}$). If instead of disjunction $(0 \leq s_2 \leq 6 \land e_2 = e_2 + 1.5)$, we select disjunction $(6 \leq s_2 \leq 12 \land e_2 = e_2 + 2)$, the TCSP is consistent and after applying the arc-consistency, we obtain the following start time points for each action: $s_0 = 10, s_1 = 13, s_2 = 11, s_3 = 10, s_4 = 10, e_n = 14$. Therefore, this new plan fulfills the user requirements.

Experimental Results
In order to check the behavior of our TCSP, we run different experiments with a hand-made problem from the driverlog domain. The temporal plan returned by LPG (Gerevini, Saetti, & Serina 2004) for this problem instance contains 18 actions and a makespan of 200 time units (before applying delays).

The above tables represent different temporal restrictions specified by the user for each timing interval setting (2, 4 and 6 timing intervals). The first row shows the plan makespan, the second row the start and end time of the plan and the third row the time in seconds. All experiments were run on Pentium IV 3GHz with 512 Mb of memory and censored after 15 minutes.

Table 1 shows the results when the user requirement is to set the plan duration below three different values. Table 2 shows the results when the user imposes restrictions on the start and end time of the plan within a time interval. Table 3 represents a restriction on the plan duration plus a restriction on the start time of the plan. Values in brackets represent the obtained values from a second CSP.

An obvious conclusion from the experiments is that the more number of timing intervals the longer time for the CSP resolution. When dealing with 2 timing intervals, the number of possible action instantiations is about 4000; for 4 timing intervals is about 16 million choices and for 6 timing intervals is about 2000 million of combinations.

In table 2 we can see that the most time-consuming experiment is for the longest plan execution interval. This clearly shows that when the plan execution interval is more restricted, the number of consistent timing intervals decreases and the search is faster. However, in this case, if there is not a valid combination of timing intervals, the necessary time to return 'No plan' increases as long as the number of timing intervals.

In the rest of experiments the time depends on when the intervals that provide the shorter duration to actions are selected in the CSP resolution, the set of valid intervals according to the start/end or duration restriction, etc. As a whole, the sooner the plan starts and the less duration for the plan the more time is necessary to find a solution.

---

3Domain from the International Planning Competition 2002: http://planning.cis.strath.ac.uk/competition
Timing intervals | Duration / Start time | Start-time | Start-time | Start-time | Start-time |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Makespan</td>
<td>245</td>
<td>No plan (No plan)</td>
<td>264</td>
<td>No plan (No plan)</td>
</tr>
<tr>
<td></td>
<td>Start-End time</td>
<td>10:00-14:05</td>
<td>12:00-16:24</td>
<td>12:00-16:24</td>
<td>12:00-16:24</td>
</tr>
<tr>
<td></td>
<td>Time (secs.)</td>
<td>0.17</td>
<td>2.25 (4.46)</td>
<td>0.0156</td>
<td>0.0156 (0.078)</td>
</tr>
</tbody>
</table>

Timing intervals | Duration / Start time | Start-time | Start-time | Start-time | Start-time |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>Makespan</td>
<td>291</td>
<td>286</td>
<td>268</td>
<td>258</td>
</tr>
<tr>
<td></td>
<td>Start-End time</td>
<td>10:00-14:51</td>
<td>12:00-16:46</td>
<td>14:00-18:28</td>
<td>17:18-21:36</td>
</tr>
<tr>
<td></td>
<td>Time (secs.)</td>
<td>2.2</td>
<td>0.0426</td>
<td>0.0316</td>
<td>0.0624</td>
</tr>
</tbody>
</table>

Timing intervals | Duration / Start time | Start-time | Start-time | Start-time | Start-time |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>Makespan</td>
<td>284</td>
<td>286</td>
<td>273</td>
<td>249</td>
</tr>
<tr>
<td></td>
<td>Start-End time</td>
<td>10:00-14:44</td>
<td>12:00-16:46</td>
<td>14:00-18:33</td>
<td>17:13-21:22</td>
</tr>
<tr>
<td></td>
<td>Time (secs.)</td>
<td>0.87</td>
<td>6.62</td>
<td>3.49</td>
<td>3.73</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Makespan</th>
<th>Start-End time</th>
<th>Time (secs.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>291</td>
<td>10:00-14:51</td>
<td>2.2</td>
</tr>
<tr>
<td>286</td>
<td>12:00-16:46</td>
<td>0.0426</td>
</tr>
<tr>
<td>268</td>
<td>14:00-18:28</td>
<td>0.0316</td>
</tr>
<tr>
<td>258</td>
<td>17:18-21:36</td>
<td>0.0624</td>
</tr>
<tr>
<td>240</td>
<td>18:00-22:00</td>
<td>0.0171</td>
</tr>
<tr>
<td>219</td>
<td>18:59-22:38</td>
<td>0.3124</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Makespan</th>
<th>Start-End time</th>
<th>Time (secs.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>284</td>
<td>10:00-14:44</td>
<td>0.87</td>
</tr>
<tr>
<td>286</td>
<td>12:00-16:46</td>
<td>6.62</td>
</tr>
<tr>
<td>273</td>
<td>14:00-18:33</td>
<td>3.49</td>
</tr>
<tr>
<td>249</td>
<td>17:13-21:22</td>
<td>3.73</td>
</tr>
<tr>
<td>228</td>
<td>18:00-21:49</td>
<td>0.156</td>
</tr>
<tr>
<td>219</td>
<td>18:59-22:38</td>
<td>0.3124</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Makespan</th>
<th>Start-End time</th>
<th>Time (secs.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>245</td>
<td>10:00-14:05</td>
<td>0.17</td>
</tr>
<tr>
<td>264</td>
<td>12:00-16:24</td>
<td>2.25 (4.46)</td>
</tr>
<tr>
<td>240</td>
<td>18:00-22:00</td>
<td>0.0156</td>
</tr>
<tr>
<td>219</td>
<td>18:59-22:38</td>
<td>0.3124</td>
</tr>
<tr>
<td>212.5</td>
<td>2.125 (4.2)</td>
<td></td>
</tr>
</tbody>
</table>

Table 3: Constraints on the duration and the start time of the plan

What we want to highlight from the experiments is that the CPU time for the CSP resolution is perfectly affordable even though no heuristic information at all has been used to select the most appropriate CSP or timing intervals for a particular solution. Moreover, modeling time restrictions at execution time is much simpler in a CSP framework than in a planning system.

Conclusions and further work
In this paper we have presented an extended model of durative actions which takes into account the fact that actions are delayed at the time of being executed in a real domain. This way, we introduce the concept of delay as the increase in the duration of an action due to very common causes in daily life but rarely considered in planning modelling. The introduction of delays create a complete different scenario in temporal planning. Now plans may have a different makespan according to its timing of execution and this aspect has to be taken into account in order to meet the user restrictions on the start/end time or duration of the temporal plan.

Using a CSP approach to tackle time restrictions at execution time is a very promising working line. Experiments show that even using a completely uninformed CSP, this approach brings significant benefits at a very low cost, specially if we consider to obtain the same gains from a planning perspective. This makes us keep on considering a separate process for allocating a plan in time. This process could be used not only to handle delays or temporal user requirements but also all kind of restrictions that come out when a plan is to be instantiated in a particular temporal setting.

References
Gerevini, A.; Saetti, A.; and Serina, I. 2004. Planning in pddl2.2 domains with lpg-td. In International Planning Competition, 14th Int. Conference on Automated Planning and Scheduling (ICAPS-04), abstract booklet of the competing planners.