Managing Temporal Cycles in Planning Problems Requiring Concurrency

Martin C. Cooper, Frédéric Maris, Pierre Régnier

IRIT, University of Toulouse 3,
{cooper, maris, regnier}@irit.fr

Abstract: In order to correctly model certain real-world planning problems, it is essential to take into account time. This is the case for problems requiring the concurrent execution of actions (known as temporally-expressive problems). In this paper we define and study the notion of temporally-cyclic problems, that is problems involving sets of cyclically-dependent actions. We characterize those temporal planning languages which can express temporally-cyclic problems. We also present a polynomial-time algorithm which transforms a temporally-cyclic problem into an equivalent acyclic problem. Applying our transformation allows any temporal planner to solve temporally-cyclic problems without explicitly managing cyclicity. We first present our results for temporal PDDL 2.1 and then extend them to a language which allows conditions over arbitrary intervals and effects at arbitrary instants.

Keywords: planning, time, temporally-expressive problems, temporally-cyclic problems.

1 Introduction

An important challenge for today's planners is the management of the time dimension, to allow the simultaneous execution of non-instantaneous actions. A large majority of real-world problems, in order to be solved or to be solved as efficiently as possible, require the execution of concurrent actions. In this article we specifically study temporally-expressive problems, in other words, problems that cannot be solved without using the concurrency of actions. A typical example of temporally-expressive problem is cooking: several ingredients or dishes must be cooked simultaneously in order to be ready at the same moment. As another example, driving a car requires concurrent actions on the steering wheel and the pedals. Large-scale applications include the management of an airport or a railway station. In industrial environments, concurrency of actions is often used to keep storage space and turn-around times within given limits. In this paper we do not study the case in which concurrency is necessary simply in order to achieve all the goals within a given time span.

When (Cushing et al., 2007) published their important work on temporally-expressive planning, the majority of temporal planners were incapable of solving temporally-expressive problems, although certain planners have since been improved

to solve such problems. Apart from HTN-type planners such as IxTeT (Ghallab & Alaoui, 1989), (Laborie & Ghallab, 1995) or HSTS (Muscettola, 1994), only CRIKEY3 (Coles et al., 2008), VHPOP (Younes & Simmons, 2003), LPGP (Long & Fox, 2003), TLP-GP (Maris & Régnier, 2008a/b), TM-LPSAT (Shin & Davis, 2004), (Hu, 2007), STEP (Huang et al., 2009) and the most recent version of LPG (Gerevini et al., 2010) can solve this type of problem.

Certain temporally-expressive problems involve a cyclically-dependent set of actions (temporally-cyclic problems). A simple example of this type of problem is the construction of two pieces of software, written by two different subcontractors, each needing to know the specification of the other program in order to correctly build the interface between the two programs. As another example, a bank will lend me money provided I pay it back after some mutually-agreed number years, and I will pay it back provided the bank lends me the money in the first place. Many contracts (such as between an employer and an employee or between two companies) can be seen as a mutually-agreed plan to solve temporally-cyclic problems. For example, a condition for an employee to start work is that they will be paid at the end of the month, and a condition for the employer to pay their salary is that the employee did indeed work during the given month.

Within this framework, an interesting question is the characterisation of planning languages which allow the user to express temporal cycles. We first solve this theoretical question. We then propose a general method for simplifying the solution of problems involving temporal cycles: they are transformed into equivalent problems in which cycles cannot occur. From the point of view of the designer of a temporal planner, this transformation avoids having to explicitly manage temporal cycles. Finally, from the user's point of view, we propose an extension of these results to a higher-level language allowing temporal intervals. This potentially facilitates the task of expressing temporal planning problems.

This article is structured as follows. After defining the framework of temporal planning (Section 2), we give a formal description of temporally-cyclic problems studied in this paper together with a simple example (Section 3). Then, we characterize the temporal sublanguages of PDDL 2.1 (Planning Domain Description Language) (McDermott, 1998), (Fox & Long, 2003) which can express these temporally-cyclic problems (Section 4). We then propose a polynomial-time transformation to convert temporally-cyclic problems into equivalent acyclic problems (Section 5) before extending these results to a richer temporal planning language (Section 6). We conclude with a discussion of related work (Section 7) together with topics for future research (Section 8).

2 Temporal planning

A temporal planning problem consists of a set A of durative actions a = <Cond(a), Eff(a), Duration(a)>, an initial state I and a goal G. I and G are sets of fluents

(propositions) i.e. positive or negative atomic propositions. Initially only the propositions in I are true and, after the execution of all the actions in the plan, all propositions in the goal G must be true. For each action *a*: Cond(*a*) is a set of propositions p together with an instant (a positive rational number representing the time relative to the start of *a*) at which p is required to be true or an interval over which p is required to be true; Eff(*a*) is a set of propositions p together with an instant (relative to the start of *a*) at which p is produced; Duration(*a*) is a positive rational number. A condition which is a disjunction of propositions can be simulated by introducing a copy of the action for each proposition in the disjunction. Note that our definition of an action includes its duration purely in order to be compatible with PDDL2.1. The duration is not actually necessary: it could simply be calculated, by default, to be the time between the first and last condition/effect of the action. A temporal plan is a set of instances of actions $<a, t_{start}>$, where *a* is an action and t_{start} a positive rational number representing its start time. For each action a in the temporal plan, each of its conditions $p \in Cond(a)$ must be true at the instant when it is required by *a* (or during the whole of the interval over which it is required by *a*), and each of its effects $p \in Eff(a)$ is assigned true at the instant when it is produced by *a*.

We consider that a solution-plan should be robust to infinitesimal shifts in the starting times of actions. This means that we disallow plans which require perfect synchronisation between different actions. (Fox et al., 2004) show how this condition can be imposed within PDDL2.1. For example, suppose that a plan P contains an action a_1 with a condition p on an interval I and another action a_2 which produces $\neg p$ at a time t. Clearly t cannot belong to the interval I, but our assumption that plans are robust to infinitesimal shifts implies that t cannot be an end-point of the interval I. However, within a same action, conditions and effects can be perfectly synchronised. Indeed, we allow an action *a* to have p as a condition on an interval I and to produce $\neg p$ at the very end of I. If I is of zero length, then *a* tests p and immediately destroys p. On the other hand, we can disallow actions *a* which produce $\neg p$ at the very beginning of an interval I on which *a* has p as a condition. This is because such an action *a* could only be executed if a second action *a*' was perfectly synchronised so as to produce p at the beginning of I. Similarly, an action *a* cannot have condition p on an interval I which touches an interval I' on which the same action *a* has the condition $\neg p$, since such an action could only be executed if a second action *a*' was perfectly synchronised so as to change the truth-value of p at the exact instant at which the intervals I and I' touch.

To begin with, we restrict ourselves to the PDDL 2.1 language and its sublanguages (described in detail in Section 5). If we allow several instances of the same action to overlap, then testing the existence of a valid plan becomes an EXPSPACE-complete problem (Rintanen, 2007). This is why in this article, as in almost all previous work in temporal planning, we do not authorize two instances of the same action to execute in parallel. This means that we conserve the PSPACE-complete complexity of classical planning (Bylander, 1994). Indeed, we also impose the slightly stronger condition that no two instances of the same action touch (i.e. there is always some non-empty interval between the end of one instance and the start of the following instance). This is a consequence of our assumption of robustness to

infinitesimal shifts in the starting times of actions together with the fact that we forbid overlapping instances of the same action. One way to impose the non-overlapping/non-touching constraint on instances of the same action is by introducing a fluent f_a for each action a: f_a is initially true (i.e. $f_a \in I$), f_a is added as a condition and $\neg f_a$ as an effect at the start of a, and f_a is added as an effect at the end of a.

3 Temporally-cyclic problems

We restrict ourselves to planners which are capable of solving temporally-expressive problems (Cushing et al., 2007). We start by giving an exemple of a certain type of temporally-expressive problem. Consider the simple example of the problem of interfacing two programs, described in Section 1. A temporal plan for this simple temporally-expressive problem is given in Figure 1. Each rectangle represents a durative action a = <Cond(a), Eff(a), Duration(a)>. The conditions Cond(a) are shown above the action a and the effects Eff(a) below. The duration of the action is given in square brackets. In this problem, the initial state is empty, the goal consists in building the two programs to be interfaced (effects r and s) and:

- action A (of duration 4) corresponds to the construction of the first program for which we need to know, in order to complete it, the specifications of the second program (condition p).
- action B (of duration 2) corresponds to the construction in parallel of the second program for which we need to know, in order to complete it, the specifications of the first program (condition q).

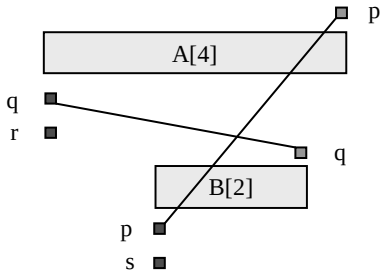

Figure 1 – temporal plan for the program-interfacing problem.

This temporal plan is characteristic of a class of plans that we call *temporally-cyclic* and that can be formalised as follows:

Definition 1 (cyclic set of actions): A set of actions $Q = \{a_1, \ldots, a_n\}$ is a cyclic set if there is a set of propositions $\{p_1, \ldots, p_n\}$ such that $\forall i \in \{1,\ldots, n-1\}$ $p_i \in \text{Eff}(a_i) \cap \text{Cond}(a_{i+1})$ and $p_n \in \text{Eff}(a_n) \cap \text{Cond}(a_1)$. We denote this cyclic set together with the propositions p_i by: $a_1 \rightarrow p_1 \rightarrow a_2 \ldots \rightarrow p_n \rightarrow a_1$.

Definition 2 (temporally-cyclic plan): A temporal plan is temporally-cyclic if it contains a set of action-instances $\{<a_1,t_1>,...,<a_n,t_n>\}$, where $a_1 \rightarrow p_1 \rightarrow a_2 \ldots \rightarrow p_n \rightarrow a_1$ is a cyclic set of actions, and is such that deleting p_i from $Eff(a_i)$ in the action-instance $<a_i,t_i>$ (for any $i \in \{1,\ldots, n\}$) would invalidate the plan since $<a_i,t_i>$ must produce p_i in order to satisfy the conditions of $<a_{i+1},t_{i+1}>$ (for each $i \in \{1,...,n-1\}$) and $<a_n,t_n>$ must produce p_n for a_1.

It is easily verified that the temporal plan of Figure 1 is temporally-cyclic.

Although the problem of interfacing two programs could be modelled using actions with conditional effects, the language allows the statement of this problem using only deterministic actions. Hence, to be complete, a planner must be able to find temporally-cyclic plans.

For this particular problem, in a forward search (in which actions can only be triggered when all their conditions are satisfied) or during the construction of the planning graph starting from the initial state, it is impossible to produce $r \in G$ or $s \in G$ because neither of the conditions {p, q} required by the actions {A, B} can be produced from I. Indeed, neither of the two actions A or B can be triggered.

In a backward search, this difficulty does not apply since, as the choice of actions is based on the goals, actions A and B will necessarily be selected. However, this is not sufficient to guarantee the completeness of the search algorithm. Indeed, if the heuristic which guides the choice of actions does not prefer those actions which are already present in the plan, then the algorithm can enter an infinite loop.

In Section 7, we discuss how existing planners such as CRIKEY (Coles et al., 2008) and LPGP (Long & Fox, 2003) transform the problem so they can find temporally-cyclic solution-plans to the original problem.

4 Temporally-expressive and temporally-cyclic languages

(Cushing et al., 2007) use the notation $L^{\text{conditions}}_{\text{effects}}$ to represent the expressivity of a temporal-planning language, where <conditions> and <effects> represent the instants at which the conditions must be true and those at which the effects are produced. The values taken by <conditions> and <effects> can be:

- s: the start of the action;
- e: the end of the action;
- o: (only concerns conditions) over all the duration of the action.

For example, the notation $L^{o}_{s,e}$ denotes a language for which the conditions of each action must be true over all its duration, and each of its effects can occur either at the start instant or the end instant of the action. PDDL2.1 is a temporally-expressive language of type $L^{s,o,e}_{s,e}$. (Cushing et al., 2007) also give a necessary and sufficient

condition for a planning domain to be temporally expressive. This allows them to partition the space of temporal languages into:

- Temporally-expressive languages in which it is possible to represent problems whose solution requires concurrency of certain actions.
- Temporally-simple languages in which it is not possible to represent such problems.

They also give the following definitions and prove the temporal-expressivity theorem which we give below.

Definition 2 (before-condition/after-condition):

- A *before-condition* is a condition which is required at the same time as, or before, the production of at least one of the effects of the action.
- An *after-condition* is a condition which is required at the same time as, or after, the production of at least one of the effects of the action.

Definition 3 (temporal gap):

- A gap between two disjoint temporal intervals which do not meet (in the sense of the primitive "meets" of (Allen, 1984)) is the interval between them (and such that the union of these three intervals forms a single interval).
- An action has a *temporal gap* if there is a gap between a before-condition and an effect, or between an effect and an after-condition, or between two effects.

Theorem 1 (Cushing et al., 2007): A sub-language of PDDL2.1 is temporally simple if and only if it disallows temporal gaps.

We will now show that a certain type of temporal gap allows a language to express temporally-cyclic problems.

Notation: Let a be an action, with $p \in \text{Cond}(a)$ and $e \in \text{Eff}(a)$. In a temporal plan containing an action-instance $\langle a,t \rangle$, we use $\tau(p \rightarrow \langle a,t \rangle)$, or more simply $\tau(p \rightarrow a)$ when no confusion is possible, to represent the instant at which this instance of a begins to require the condition p and $\tau(\langle a,t \rangle \rightarrow e)$, or more simply $\tau(a \rightarrow e)$, to represent the instant at which this instance of a produces e.

Definition 4 (temporally cyclic/acyclic language): A language is temporally cyclic if it allows temporally-cyclic plans. Otherwise, it is temporally acyclic.

Theorem 2: A sub-language of PDDL2.1 is temporally cyclic if and only if it authorises temporal gaps between an effect and an after-condition.

This theorem, whose proof is given below, allows us to refine the taxonomy of temporal languages proposed by (Cushing et al., 2007) by distinguishing a subclass of temporally-expressive languages corresponding to temporally-cyclic languages. This new taxonomy is given in Figure 2.

Proof of Theorem 2: (1) Firstly, we show that if a sub-language of PDDL2.1 authorises temporal gaps between an effect and an after-condition then it authorises

temporally-cyclic plans. The problem whose solution is given in Figure 1 (consisting of the two actions A and B) demonstrates that the language L_s^e authorises the existence of temporally-cyclic plans. As all sub-languages of PDDL2.1 which authorise temporal gaps between an effect and an after-condition contain L_s^e as a sub-language (Cushing et al., 2007), all these languages authorise cyclic sets of actions.

(2) Secondly, we show that if a temporal planning language authorises the existence of temporally-cyclic plans, then it authorises temporal gaps between an effect and an after-condition. Consider a temporal planning problem with a temporally-cyclic plan. This means that the temporal plan contains a set of action-instances $\{<a_1,t_1>,...,<a_n,t_n>\}$, where $a_1 \rightarrow p_1 \rightarrow a_2 \ldots \rightarrow p_n \rightarrow a_1$ is a cyclic set of actions (i.e. $\forall i \in \{1,\ldots, n-1\}\ p_i \in (\text{Eff}(a_i) \cap \text{Cond}(a_{i+1})) \wedge (p_n \in \text{Eff}(a_n) \cap \text{Cond}(a_1))$). Furthermore, $<a_i,t_i>$ must produce p_i to satisfy the conditions of $<a_{i+1},t_{i+1}>$ (for each $i \in \{1,...,n-1\}$) and $<a_n,t_n>$ must produce p_n for a_1. It follows that $(\forall i \in \{1,\ldots, n-1\}\ \tau(a_i \rightarrow p_i) < \tau(p_i \rightarrow a_{i+1})) \wedge (\tau(a_n \rightarrow p_n) < \tau(p_n \rightarrow a_1))$. Suppose that there cannot be a temporal gap between an effect and an after-condition of an action. Then $(\forall\ i \in \{1,\ldots, n-1\}\ \tau(p_i \rightarrow a_{i+1}) \leq \tau(a_{i+1} \rightarrow p_{i+1})) \wedge (\tau(p_n \rightarrow a_1) \leq \tau(a_1 \rightarrow p_1))$. Therefore $\tau(a_1 \rightarrow p_1) < \tau(p_1 \rightarrow a_2) \leq \tau(a_2 \rightarrow p_2) < \tau(p_2 \rightarrow a_3) \leq \ldots \leq \tau(a_n \rightarrow p_n) < \tau(p_n \rightarrow a_1) \leq \tau(a_1 \rightarrow p_1)$. From which we can deduce that $\tau(a_1 \rightarrow p_1) < \tau(a_1 \rightarrow p_1)$, which is impossible. Therefore there is a temporal gap between an effect and an after-condition. □

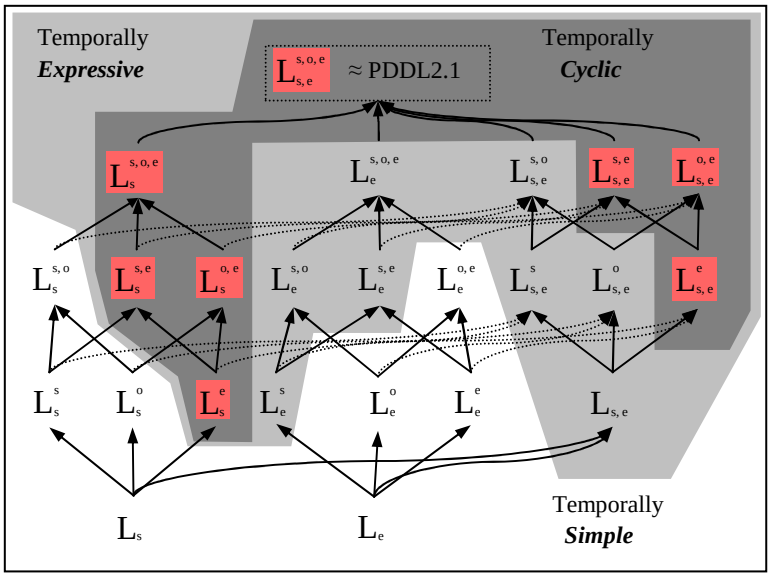

Figure 2 – Taxonomy of temporal languages and their expressivity.

Corollary 1: A sub-language of PDDL2.1 is temporally-cyclic if and only if it is a super-language of L_s^e.

An important question is whether it is possible to model temporally-cyclic problems differently so as to avoid temporal cycles. The next section shows that it is indeed possible to reduce the temporally-cyclic language $L_{s,e}^{s,o,e}$ (i.e. the whole of PDDL2.1) to the temporally-acyclic language $L_{s,e}^{s,o}$. This is achieved by modelling differently actions which have a temporal gap between an effect and an after-condition.

Note that neither Theorem 2 nor Corollary 1 require the two technical assumptions of Section 2 that we forbid perfect synchronisation of actions and overlapping instances of the same action. They are, on the other hand, necessary for the reduction described in the next section.

5 Transformation of a temporally-cyclic problem into a temporally-acyclic problem

In this section we show how to transform a problem whose solutions potentially contain cycles, expressed in the PDDL 2.1 language $L_{s,e}^{s,o,e}$, into an equivalent problem expressed in the language $L_{s,e}^{s,o}$ which, by Theorem 2, does not authorise cycles. In practice, according to the problem, we could transform all actions, which amounts to expressing the problem in the language $L_{s,e}^{s,o}$, or only transform those actions which can possibly lead to the creation of cycles, while still remaining in the language $L_{s,e}^{s,o,e}$. The proof of Theorem 2 shows that a necessary condition for a problem to have a temporally-cyclic solution-plan is the existence of a cyclic set of actions $a_1 \rightarrow p_1 \rightarrow a_2 \ldots \rightarrow p_n \rightarrow a_1$ involving a temporal gap between an effect p_i and an after-condition p_{i-1} of some action a_i.

In order to detect such actions, we construct the causality graph <V,E> where the set of vertices V is the union of the set of actions and the set of propositions, and the set of directed edges E = {<p,*a*>: p ∈ Cond(*a*)} ∪ {<*a*,p>: p ∈ Eff(*a*)}. For each candidate action A (i.e. an action which has a gap between an effect and some after-condition p), we check a necessary condition for the existence of a cycle: does the link p → A belong to a cycle in the causality graph. If not, then there is no need to transform A.

An action A which has after-conditions which could cause a cycle must be transformed into an action A' for which these conditions have been deleted (to eliminate the temporal gap). Each such deleted after-condition p of A is reformulated as a link between A and the presence in the plan of some action B which produces this condition p for A. (In fact B is the last, in the temporal order of the plan, to produce p

before it is needed by A). This link must be active whenever A is present in the plan. The presence of this link is checked by an action $LC_{p,A}$ (Link Check between A and some action B which produced p for A).

To transform an action A in order to delete an after-condition p, we:

- Add a proposition Linked(p,A) to the initial state and to the goal of the problem. This link-proposition will mean that, when A is executed and deletes Linked(p,A) (see below) this forces the execution of the check action $LC_{p,A}$ to re-establish Linked(p,A). $LC_{p,A}$ can only execute if some action B actually produced p for A.
- Transform A into A':
  - by deleting the after-condition p,
  - by adding the proposition LC-Active(p,A) at the instant when p was required, to denote the fact that p must be produced by some action before it is required by A,
  - by adding the effect ¬Linked(p,A) which forces the execution of the action $LC_{p,A}$ to re-establish Linked(p,A) (this proposition being present in the initial state and the goal),
  - by adding the effect ¬LC-Active(p,A) at the start of A' to ensure that $LC_{p,A}$ is executed after the end of A'.
  - by adding the condition Linked(p,A) at the start of A' to ensure that every execution of A' is followed by an execution of $LC_{p,A}$.
- For each action B which produces p, transform B into B' by adding the effect ¬LC-Active(p,A) at the instant when B produces p to denote the fact that p must be produced (by at least one of these actions B) *before* being required by A.
- Add an instantaneous action $LC_{p,A}$ (Link Check) with the two conditions p and LC-Active(p,A) and with the effect Linked(p,A).
- In the special case that A also destroys p at the end of A, we also need to perform the following two modifications:
  - the effect ¬p is deleted from the end of A' and added to $LC_{p,A}$,
  - for each action C (apart from A) such that $p \in Cond(C)$, transform C into C' by adding the effect ¬LC-Active(p,A) at the instant when p is required by C, to ensure that such actions cannot be executed between the end of A' and $LC_{p,A}$.

These new propositions and actions cannot introduce new temporal gaps between effects and after-conditions since all the conditions which are added during the transformation are added at the start of actions. The resulting problem is equivalent to the original problem and the inverse transformation of a temporal plan is achieved by simply deleting the actions $LC_{p,A}$ and replacing transformed actions A',B',C' by the original versions A,B,C of these same actions. We prove formally the equivalence of the original and transformed problems in Theorem 4 in the Appendix.

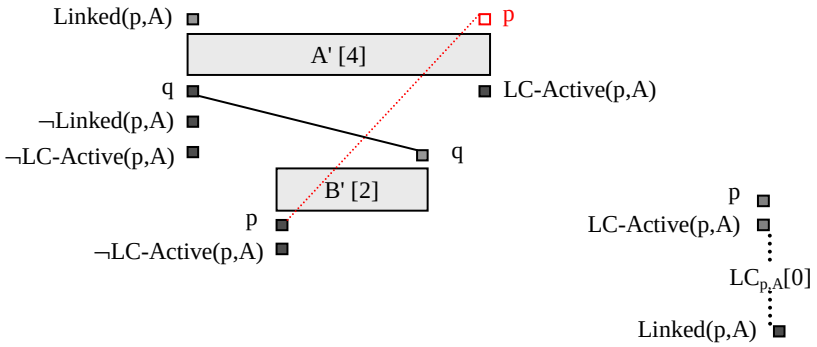

Figure 3 – Example of the transformation of a problem from $L^{s,o,e}_{s,e}$ to $L^{s,o}_{s,e}$.

In our running example, for action A with after-condition p, the transformation is illustrated in Figure 3.

```
Transformation Algorithm
   Candidates ← {actions with an effect/after-condition gap}
   While Candidates ≠ ∅
      Extract A from Candidates ;
      For each after-condition p of A which causes a gap
         Build the causality graph G from p → A ;
         If p → A belongs to a cycle in G then
            Transform-Cond-and-Eff(A,p,A') ;
            For each action B which produces p
               Transform-Eff-1(B,p,B') ;
            End For ;
            If A simultaneously requires and destroys p then
               For each action C ≠ A which requires p
                  Transform-Eff-2(C,p,C') ;
               End For ;
            End If ;
            Add the action LC_p,A to the problem ;
         End If ;
      End For ;
   End While ;
```

Figure 4 – Transformation algorithm using the causality graph.

The complete transformation of the problem into the temporally acyclic language $L^{s,o}_{s,e}$ can be achieved via the systematic deletion of all after-conditions (without testing for cycles in the causality graph) via the transformation described above. Let n denote the size of the problem, i.e. $n = \Sigma(|Eff(a)|+|Cond(a)|)$ where the sum is over all actions *a* (assuming, without loss of generality, that each action has at least one condition or effect). Then this simple algorithm has a complexity of $O(n^2)$. This is because, for each action A with an after-condition $p \in Eff(A)$, we may need to transform all other actions B. The transformation of the actions A and B, for a given after-condition $p \in Eff(A)$, requires constant time.

The worst-case time complexity of the partial transformation of a problem, in which we eliminate only those after-conditions that could potentially give rise to cycles (detected by causality graph analysis) is also $O(n^2)$, but this algorithm has the advantage that the number of new actions introduced will generally be less. The algorithm is given in Figure 4. For each action A and each after-condition $p \in Eff(A)$, we firstly test whether there is a cyclic set of actions including $p \rightarrow A$. This can be achieved by finding the strongly-connected components of the causality graph G using the algorithm of (Tarjan, 1972). On a directed graph <V,E>, this algorithm has a time complexity of O(|V| + |E|). But this is O(n), since n is exactly equal to the number of directed edges. A complete transformation can therefore be achieved in $O(n^2)$ by the argument above.

In both cases, the number of new actions $LC_{p,A}$ (as well as the number of new propositions LC-Active(p,A) and Linked(p,A)) is bounded above by the number of after-conditions present in the actions of the problem.

6 A richer temporal planning language

In this section we introduce a temporal planning language which is richer than temporal PDDL 2.1 (which is a language of type $L^{s,o,e}_{s,e}$). Although this language can be reduced in polynomial time to PDDL2.1 (Fox et al., 2004), it allows the user to express real-world problems more easily. In this language, instantaneous conditions and effects may occur at *any* instant of an action A; they are no longer restricted to occur only at the start or the end of A. Furthermore, conditions of an action A may now also occur over *arbitrary* intervals; they are not restricted to "overall" (which corresponds to an interval equal to the duration of A). We denote this new language by $L^{[]}_{inst}$ since actions can have conditions on arbitrary intervals (of possibly zero length) and effects at arbitrary instants. Since $L^{[]}_{inst}$ includes $L^{s,o,e}_{s,e}$ as a sublanguage, it is clearly temporally-cyclic.

Theorem 3: A sub-language $\mathcal{L}$ of $L^{[]}_{inst}$ is temporally cyclic if and only if $\mathcal{L}$ authorises actions with a temporal gap between an effect and the start of an after-condition.

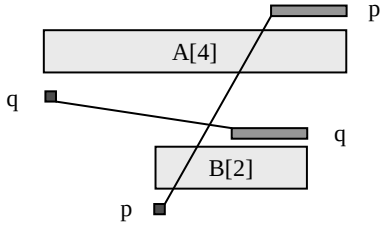

Figure 4 – Example of temporally-cyclic plan.

Proof of Theorem 3: (1) Firstly, we show that if the planning language $\mathcal{L}$ authorises actions with an effect before the start of a condition then $\mathcal{L}$ authorises temporally-cyclic plans. We use a slightly modified version of the example of Figure 1, shown in Figure 4. Let A be an action of length 4 with a condition p on the interval [3,4] and an effect q at the start of A. Let B be an action of length 2 with a condition q on the interval [1,2] and an effect p at the start of B. It is easy to see that a solution-plan for the temporal planning problem with empty initial state and goal {p,q} consists in executing A at time 0 and B at time 1 (since $\tau(A \rightarrow q)=0 < 2=\tau(q \rightarrow B)$ and $\tau(B \rightarrow p)=1 < 3=\tau(r \rightarrow A)$). In this temporal plan P, $p \in Eff(B) \cap Cond(A)$ and $q \in Eff(A) \cap Cond(B)$ and hence $A \rightarrow q \rightarrow B \rightarrow p \rightarrow A$ is a cyclic set of actions. Furthermore, deleting p from Eff(B) or q from Eff(A) would invalidate the plan. Thus, by Definition 2, P is a temporally-cyclic plan and hence $\mathcal{L}$ is temporally-cyclic.

(2) The proof that if a planning language $\mathcal{L}$ authorises the existence of temporally-cyclic plans, then it authorises actions with an effect before the start of a condition is identical to part (2) of the proof of Theorem 2. □

We now show how to generalise the transformation described in Section 6 to the richer language $L^{[]}_{inst}$. Once again it suffices to eliminate all temporal gaps between effects and after-conditions. In practice, according to the problem, we could transform all actions or only transform those actions which can possibly lead to the creation of cycles, as discussed in detail in Section 6: for each candidate action A (i.e. an action which has a gap between an effect and some after-condition p), we can check a necessary condition for the existence of a cycle: does the link $p \rightarrow A$ belong to a cycle in the causality graph. If not, then there is no need to transform A.

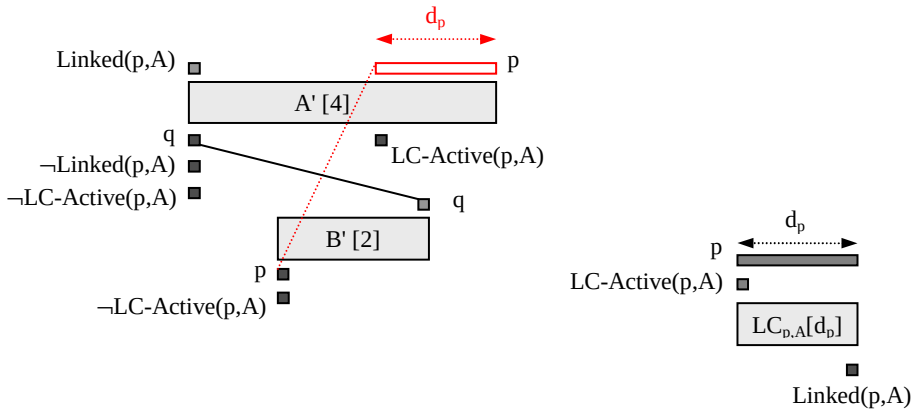

Figure 5 – Example of the transformation of a problem in $L^{[]}_{inst}$ to eliminate a temporal gap between an effect and the start of an after-condition.

An action A which has after-conditions which could cause a cycle must be transformed into an action A' for which these conditions have been deleted (to eliminate the temporal gap). Each such deleted after-condition p of A is transferred to a Link-Check action $LC_{p,A}$ which effectively checks the presence in the plan of some action B which produces this condition p for A. If the condition p of A is over an interval I, then the action $LC_{p,A}$ is not instantaneous (as in Section 6) but has a duration corresponding to the duration d_p of I. This is illustrated in Figure 5. If A has more than one after-condition, then we transform it separately for each of its after-conditions taken in reverse chronological order of the beginning of the interval over which it is required. In other words, we delete the last after-condition first. In particular, it is possible in our new language for an action A to require the same proposition p at several different instants and/or over several different intervals during the execution of A. In the following, we consider only the last interval I (of length 0 in the case of an instantaneous condition) over which p is required by A. Since p is an after-condition of A, the beginning of the interval I cannot coincide with the start of A.

To transform an action A in order to delete the after-condition p over interval I, we:

- Add a proposition Linked(p,A) to the initial state and to the goal of the problem. This means that, when A is executed and deletes Linked(p,A) this forces the execution of the check action $LC_{p,A}$ to re-establish Linked(p,A). $LC_{p,A}$ can only execute if some action B actually produced p for A.
- Transform A into A':

- by deleting the after-condition p over the interval I,
- by adding the effect ¬Linked(p,A) which forces the execution of the action $LC_{p,A}$ to re-establish Linked(p,A) (this proposition being present in the initial state and the goal),
- by adding the condition Linked(p,A) at the start of A', to ensure that there is an execution of $LC_{p,A}$ for *every* execution of A'.
- by adding the proposition LC-Active(p,A) at the beginning of the interval I, to prevent actions changing the value of p between this moment and the start of the execution of $LC_{p,A}$ (which has LC-Active(p,A) as a condition).
- by adding the effect ¬LC-Active(p,A) at the start of A' to ensure that $LC_{p,A}$ is executed *after* the beginning of the interval I.

• For each action B which produces p, transform B into B' by adding the effect ¬LC-Active(p,A) at the instant when B produces p to denote the fact that p must be produced (by at least one of these actions B) *before* being required by A.

• Add an action $LC_{p,A}$ (Link Check) of duration d_p (the length of interval I) with condition LC-Active(p,A) at the start of $LC_{p,A}$, effect Linked(p,A) at the end of $LC_{p,A}$ and p as an overall condition (i.e. required over the whole duration of $LC_{p,A}$).

• In the special case that A also destroys p precisely at the end of the interval I, we also need to perform the following two modifications:
  - the effect ¬p is deleted from the end of the interval I in A' and added to the end of $LC_{p,A}$,
  - for each action C (apart from A) such that p is required by C over an interval I' (of possibly zero length), transform C into C' by adding the effect ¬LC-Active(p,A) at the instant d_p time units before the end of I', to ensure that this instant does not fall between the beginning of the interval I in A' and the start of $LC_{p,A}$. This ensures that the interval over which C requires p does not overlap the end of the interval I in A' (corresponding to the moment when A destroys p).

These new propositions and actions cannot introduce new temporal gaps between effects and after-conditions since all the conditions which are added during the transformation are added at the start of actions. The resulting problem is equivalent to the original problem and the inverse transformation of a temporal plan is achieved by simply deleting the actions $LC_{p,A}$ and replacing transformed actions A',B',C' by the original versions A,B,C of these same actions. We prove formally the equivalence of the original and transformed problems in Theorem 5 in the Appendix. The time complexity of this transformation is again quadratic, by the same argument as in Section 5.

7 Related work

In this section we first review temporally-expressive planners, before examining the solutions that have been proposed to cope with temporal cycles. The planners which can solve temporally-expressive planning problems are essentially based on three different types of algorithm (Maris & Régnier, 2010):

- **State-space search:** (Cushing et al., 2007) describe TEMPO, a temporally-expressive paradigm based on search in an extended state-space in which decisions concerning when to execute an action are made after all decisions concerning which actions to execute have been made (lifting over time). The CRIKEY3 planner (Halsey et al., 2004), (Coles et al., 2008) performs a state-space search coupled with a Simple Temporal Network (STN) associated with each action. It also uses a novel version of the relaxed planning graph to optimise search.
- **Partial-order planners:** partial-order planners (POP) have been successfully extended to the temporally-expressive framework with planners such as VHPOP (Younes & Simmons, 2003) and DT-POP (Schwartz & Pollack, 2004). Many other planners have in the past used a hierarchical plan-space (HTN). They use a temporal logic based on instants and intervals, together with a Time Map Manager which manages the temporal constraints. This is the case for planners such as FORBIN (Dean et al., 1988), HSTS (Muscettola, 1994), IXTET (Ghallab & Alaoui, 1989), (Laborie & Ghallab, 1995), TEST (Reichgelt & Shadbolt, 1990), TIMELOGIC (Allen & Koomen, 1983), TLP (Tsang, 1987), and TRIPTIC (Rutten & Hertzberg, 1993). Although their representation languages are very expressive, there are many differences with PDDL 2.1, which makes the comparison with other systems particularly difficult. The limited experimental trials which we were able to perform highlighted their relatively poor performance.
- **Temporal extensions of GRAPHPLAN:** The use of the planning graph (Blum & Furst, 1995) has also been extended to temporally-expressive problems. In LPGP (Long & Fox, 2003) and TM-LPSAT (Shin & Davis, 2004) durative actions are decomposed into three instantaneous actions (start, invariant, end). Both of these planners use a solver to extract a solution. However, whereas LPGP uses backward search in the planning graph while maintaining the consistency of a set of temporal constraints, TM-LPSAT simultaneously codes the planning graph and the temporal constraints then calls the LPSAT solver (Wolfman & Weld, 1999). (Hu, 2007) also describes a similar method: all actions are decomposed into two simple actions, then the problem is coded as a CSP according to semantics based on modal operators and which include the coding of the planning graph. The TLP-GP planner (Maris & Régnier, 2008.a/b) uses similar methods to those of LPGP and TM-LPSAT. TLP-GP delegates a larger part of search to a solver which probably explains why it is able to outperform LPGP, VHPOP2.2 and CRIKEY3 (on problems without numeric effects). Any improvement in the performance of the solver automatically leads to an improvement of the performance of TLP-GP. The LPG planner (Gerevini et al., 2010) has been recently revised, decomposing each action into two instantaneous actions, in order to deal with temporally-expressive

problems. It uses temporal action graphs (which are similar to a planning graph) and local search.

In Section 3, we have shown that problems can occur when temporally-expressive problems involve temporally-cyclic sets of actions. Theorem 2 shows that one solution is to disallow after-conditions; this may permit a more efficient implementation (Maris & Régnier, 2008a/b) but is not totally satisfactory since it places what the user may perceive as a rather arbitrary restriction within the temporal planning language. Another approach is for the planner to transform the original problem, within a pre-processing step, in order to produce an equivalent problem without after-conditions. The transformation described in Section 5 introduces at most one new action for each after-condition. Other transformations have been proposed in the literature (Long & Fox, 2003) (Coles et al., 2008) which also eliminate after-conditions, although this was not an explicitly-stated aim in the descriptions of these transformations.

In CRIKEY3 (Coles et al., 2008), all durative actions are decomposed into instantaneous ("snap") actions denoting the start and end of the action. The correct duration of an action is then imposed as a simple linear constraint between the times of these two snap actions. A solution plan is a classical plan composed of snap-actions which also has a consistent solution for an STN associated with each envelope between corresponding start and end snap actions. In LPGP (Long & Fox, 2003) a transformation decomposes all durative actions into a start, an invariant and an end action. To guarantee equivalence with the original problem, "clip" actions have to be introduced to ensure that the start, invariant and end component actions are always contiguous in a solution plan. Intermediate conditions can be managed by splitting actions into component actions enclosed within an "envelope" action (Smith, 2003). All of these approaches have the effect of eliminating after-conditions from problems expressed in PDDL2.1, since, for example, conditions at the end of the original action are now associated with the instantaneous end-action. Compared with the transformation described in the present paper, these transformations each involve a constant factor increase in the number of actions even when the number of after-conditions is small. Of course, if the only aim is to eliminate temporal cycles, these transformations could also be applied selectively i.e. only to actions a with an after-condition p such that $p \rightarrow a$ is part of a cyclic set of actions.

(Rintanen, 2007) proposed an interesting and altogether different transformation, from discrete temporal PDDL2.1 to classical planning. For each durative action a, there is a counter in the transformed problem together with an action which initialises the counter to the duration of the corresponding action. The passage of time is itself simulated by a single action which decrements all counters (or a set of actions which simulate the passage of 1,2,4… basic time units). All conditions and effects of all actions are coded by a single action whose conditions and effects are conditioned by the values of the counters. This is a polynomial transformation which necessarily eliminates after-conditions since the time dimension itself has been eliminated. However, there are essential differences between discrete and continuous time. Even if all action durations are integers, there are simple temporal planning problems which have a solution over continuous time but no solution over discrete time. Figure 6

shows such an example, adapted from an example of (Cushing et al., 2007). The initial state is I = {} and the goal is G = {b, d, e}. The resolution of this problem requires the concurrent execution of the actions A, B, and C. Over discrete time, this problem has no solution. Indeed, in a solution we must have:

- $t_{start}(B) + 4 > t_{start}(A) + 5$ so that the goal fluent d is established by B after the effect ¬d of A;
- $t_{start}(A) + 5 > t_{start}(C) + 3$ so that the goal fluent b is established by A after the effect ¬b of C;
- $t_{start}(C) > t_{start}(B)$ since the fluent c must be established by B before it is required by C;

It is easily verified that this system of inequalities has a solution over the rationals but not over the integers. Although this particular problem can be rendered solvable over discrete time by dividing the basic time unit by three (as shown in Figure 6), it is an open problem whether there is even a polynomial bound on the number of times the basic time unit would need to be divided by a constant in order to guarantee finding a discrete solution to a problem which has a continuous solution. Such a bound is required to guarantee that plan-length over discrete time is a polynomial function of plan-length over continuous time.

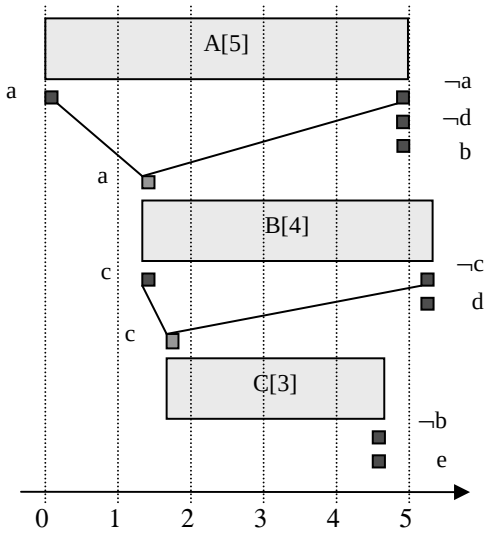

Figure 6 – Example of problem which has a solution over continuous time but no solution over discrete time.

8 Conclusion and future work

In this article we have studied a particular class of temporal plans involving concurrency of actions: those plans containing a set of cyclically-dependent actions. Care is required in developing a temporal planning algorithm so as to guarantee completeness with respect to the class of problems requiring temporally-cyclic plans. A language is temporally cyclic if it allows temporally-cyclic plans. We have given a simple characterisation of temporally-cyclic languages. We have also shown that it is possible to reduce the temporally cyclic language $L_{s,e}^{s,o,e}$ (i.e. the whole of PDDL2.1) to the temporally-acyclic language $L_{s,e}^{s,o}$ at the cost of a modest increase in problem size. This reduction allowed us to give a polynomial-time transformation to transform a temporally-cyclic problem into a temporally-acyclic problem. This transformation simplifies the task of writing complete temporal planners.

We have extended this work to a more expressive language which allows actions containing conditions over arbitrary intervals as well as conditions and effects at arbitrary instants. An interesting open question is whether a similar polynomial-time transformation exists for languages which allow perfect synchronisation between the actions in a plan (something which we have explicitly disallowed in this paper). Possible avenues of future research opened up by the work presented in this article include the completeness of algorithms which use even more expressive languages. For example, in the long term, one can imagine the extension to numerical planning involving incertitude and non-instantaneous transitions.

Our long-term aim is to build tools to allow a naive user to use a temporal planner to solve complex temporally-expressive problems. This involves constructing a rich high-level temporal planning language involving intervals but also modalities over intervals (Cooper et al., 2010). An essential part of this research programme is the building of compilers to translate to lower-level languages so that technical details are hidden from the user. The possible presence of temporal cycles is one such technical detail.

Acknowledgments

We would like to express our gratitude to the reviewers whose comments helped us significantly improve both the content and the presentation of this paper.

Appendix

Let TEMPORAL PLANNING($\mathcal{L}$) represent the set of instances of the temporal planning problem (as described in Section 3) belonging to the language $\mathcal{L}$.

Theorem 4: There is a polynomial-time reduction from TEMPORAL PLANNING($L_{s,e}^{s,o,e}$) to TEMPORAL PLANNING($L_{s,e}^{s,o}$).

Proof of Theorem 4: It suffices to show that the transformation given in Section 6 produces an equivalent problem, since we have already shown in Section 6 that it is a polynomial reduction. Let Π and Π' be the original and transformed problems, respectively.

Let P be a valid plan for Π, and let P' be identical to P except that all actions have been transformed (i.e. A becomes A', B becomes B', etc.). Let δ be strictly positive but smaller than the interval between any two distinct times at which a condition or effect occurs during the execution of the plan P. For each instance of A' in P', we add an instance of $LC_{p,A}$ at a time δ after the end of A'. Since nothing happens during this interval of length δ after the end of A', by definition of δ, the combined effect of this instance of A' and the following instance of $LC_{p,A}$ is equivalent to A in the original plan P.

Now consider a plan P' for Π'. Let P be obtained from P' by deleting all instances of $LC_{p,A}$ and by replacing all transformed actions by their original versions (i.e. A' by A, B' by B, etc.). We will show that P is a valid plan for Π. The condition LC-Active(p,A) of $LC_{p,A}$ is only established by action A' (at its end). Furthermore, LC-Active is destroyed at the start of A' and remains false during the execution of A', since, by assumption, no two instances of A' can overlap. Therefore, no instance of $LC_{p,A}$ can occur during the execution of A'. The condition Linked(p,A) of A' is only established by $LC_{p,A}$. Furthermore, A' destroys Linked(p,A) which is one of the goals of Π'. It follows that every instance of A' must be followed by an instance of $LC_{p,A}$ in P'. We must show that p is true at the end of the execution of each instance of A'. We know that p is true when the following instance of $LC_{p,A}$ is executed, so we only have to show that p was not established between the end of A' and the execution of $LC_{p,A}$. But this is impossible since all actions B' which establish p also destroy LC-Active(p,A). In the case that the effect $\neg$p is transferred from the end of A to $LC_{p,A}$, we have to show that no action C' in P' uses p as a condition during the interval between the end of A' and the execution of $LC_{p,A}$. Again, this is impossible since all such actions C' also destroy LC-Active(p,A). □

Recall that $L^{[]}_{inst}$ represents the temporal planning language in which actions can have conditions on arbitrary intervals (of possibly zero length) and effects at arbitrary instants. Let $nac\text{-}L^{[]}_{inst}$ represent the same language in which actions have no after-conditions.

Theorem 5: There is a polynomial-time reduction from TEMPORAL PLANNING($L^{[]}_{inst}$) to TEMPORAL PLANNING($nac\text{-}L^{[]}_{inst}$).

Proof of Theorem 5: By the same argument as in Section 6, the global transformation consisting in replacing every after-condition by a corresponding link-check action is a polynomial reduction. Thus, it suffices to show that the transformation described in Section 7 produces an equivalent problem. We consider a single transformation corresponding to the deletion of the after-condition p over the interval I in action A. Let Π and Π' be the original and transformed problems, respectively.

Let P be a valid plan for Π, and let P' be identical to P except that all actions have been transformed (i.e. A becomes A', B becomes B', etc.). By our assumption of robustness to small shifts in the execution times of actions, we can assume that no two actions are perfectly synchronised in terms of the start/end times of conditions or the times of effects. Let δ be strictly positive but smaller than the interval between any two distinct times at which a condition begins or ends or an effect occurs during the execution of the plan P. For each instance of A' in P', we add an instance of $LC_{p,A}$ starting at a time δ after the beginning of the interval I of A'. Since nothing happens during this interval of length δ after the beginning or the end of the interval I of A'. by definition of δ, the combined effect of this instance of A' and the following instance of $LC_{p,A}$ is equivalent to A in the original plan P.

Now consider a plan P' for Π'. Let P be obtained from P' by deleting all instances of $LC_{p,A}$ and by replacing all transformed actions by their original versions (i.e. A' by A, B' by B, etc.). We will show that P is a valid plan for Π. The condition LC-Active(p,A) of $LC_{p,A}$ is only established by action A' (at the start of the interval I over which p is a condition). Furthermore, LC-Active is destroyed at the start of A' and remains false during the execution of A' up to the start of I, since, by assumption, no two instances of A' can overlap. Therefore, no instance of $LC_{p,A}$ can begin during the execution of A' before the start of interval I. The condition Linked(p,A) of A' is only established by $LC_{p,A}$. Furthermore, A' destroys Linked(p,A) which is one of the goals of Π'. It follows that every instance of A' must be followed by an instance of $LC_{p,A}$ in P'. We must show that p is true during the interval I of each instance of A'. We know that p is true when the following instance of $LC_{p,A}$ is executed, so we only have to show that p was not established between the start of the interval I in A' and the start of the execution of $LC_{p,A}$. But this is impossible since all actions B' which establish p also destroy LC-Active(p,A). In the case that the effect ¬p is transferred from the end of the interval I of A to $LC_{p,A}$, we have to show that no action C' in P' requires p as a condition during the interval between the end of interval I of A' and the end of the execution of $LC_{p,A}$. Again, this is impossible since all actions C' with $p \in Cond(C')$ destroy LC-Active(p,A) at an instant d_p time units before the end of the interval I' over which C' requires p. This ensures that the end of I' does not fall between the end of I in A' and the end of $LC_{p,A}$, (since, otherwise, C' would destroy LC-Active(p,A) before it is required by $LC_{p,A}$). □

References

ALLEN J.F. (1984). Towards a General Theory of Action and Time, *Artificial Intelligence 23(2)*, pp. 123-154.

ALLEN J.F. & KOOMEN J.A.G.M. (1983). Planning Using a Temporal World Model, *Proc. of 8th International Joint Conference on Artificial Intelligence*, pp. 741-747.

BLUM A.L. & FURST M.L., (1995). Fast planning through planning-graphs analysis, *Proc. of 14th International Joint Conference on Artificial Intelligence*, pp. 1636-1642.

BYLANDER T. (1994). The Computational Complexity of Propositional STRIPS Planning, *Artificial Intelligence 69(1-2)*, pp. 165-204.

COLES A., FOX M., LONG D., SMITH A. (2008). Planning with Problems Requiring Temporal Coordination, *Proc. of AAAI 2008*, pp. 892-897.

COOPER M., MARIS F., REGNIER P. (2010). Compilation of a high-level temporal planning language into PDDL 2.1, *Proc. of 22nd International Conference on Tools with Artificial Intelligence*, vol. 2, pp 181-188.

CUSHING W., KAMBHAMPATI S., MAUSAM, WELD D.S. (2007). When is Temporal Planning Really Temporal? *Proc. of 20th International Joint Conference on Artificial Intelligence*, pp. 1852-1859.

DEAN T., FIRBY J., MILLER D., (1988). Hierarchical Planning involving deadlines, travel time and resources, *Computational Intelligence 6(1)*, pp. 381-398.

FOX M., LONG D., (2003). PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains, *Journal of Artificial Intelligence Research 20*, pp. 61-124.

FOX M., LONG D., HALSEY K., (2004). An Investigation into the Expressive Power of PDDL2.1, *Proc. of 16th European Conference on Artificial Intelligence*, pp. 328-342.

GEREVINI A., SAETTI A., SERINA I., (2010). Temporal Planning with Problems Requiring Concurrency through Action Graphs and Local Search, *Proc. of 20th International Conference on Automated Planning and Scheduling*, pp. 226-229.

GHALLAB M. & ALAOUI A.M. (1989). Managing Efficiently Temporal Relations Through Indexed Spanning Trees, *Proc. of 11th International Joint Conference on Artificial Intelligence*, pp. 1297-1303.

HALSEY K., LONG D., FOX M., (2004). CRIKEY - A Planner Looking at the Integration of Scheduling and Planning, *Proc. of the "Integration Scheduling Into Planning" Workshop, ICAPS'03*, pp. 46-52.

HU Y. (2007). Temporally-Expressive Planning as Constraint Satisfaction Problems, *Proc. of 20th International Joint Conference on Artificial Intelligence*, pp. 192-199.

HUANG R., CHEN Y., ZHANG W. (2009). An Optimal Temporally Expressive Planner: Initial Results and Application to P2P Network Optimization, *Proc. of 19th International Conference on Automated Planning and Scheduling*.

LABORIE P. & GHALLAB M. (1995) Planning with Sharable Resource Constraints, *Proc. of 14th International Joint Conference on Artificial Intelligence*, pp. 1643-1651.

LONG D. & FOX M. (2003). Exploiting a graphplan framework in temporal planning, *Proc. of 13th International Conference on Automated Planning and Scheduling*, pp. 52-61.

MARIS F. & REGNIER P. (2008a). TLP-GP: Solving Temporally-Expressive Planning Problems, *Proc. of 15th International Symposium on Temporal Representation and Reasoning*, pp. 137-144.

MARIS F. & REGNIER P. (2008b). TLP-GP: New Results on Temporally-Expressive Planning Benchmarks, *Proc. of 20th IEEE International Conference on Tools with Artificial Intelligence*, vol. 1, pp. 507-514.

MARIS F. & REGNIER P. (2010). Planification temporellement expressive, *Revue d'Intelligence Artificielle*, vol. 24/4 – 2010, pp 445-464.

MCDERMOTT D.V., (1998). PDDL, The Planning Domain Definition Language, *Technical Report*, http://cs-www.cs.yale.edu/homes/dvm/.

MUSCETTOLA N., (1994). HSTS: integrating planning and scheduling. In M. ZWEBEN & M. FOX Eds. *Intelligent Scheduling*, pp. 169-212.

REICHGELT H. & SHADBOLT N. (1990). A Specification Tool for Planning Systems, *Proc. of 9th European Conference on Artificial Intelligence*, pp. 541-546.

RINTANEN J. (2007). Complexity of Concurrent Temporal Planning, *Proc. of 17th International Conference on Automated Planning and Scheduling*, pp. 280-287.

RUTTEN E. & HERTZBERG J. (1993). Temporal Planner = Nonlinear Planner + Time Map Manager, *Artificial Intelligence Communications 6(1)*, pp. 18-26.

SCHWARTZ P. & POLLACK M.E. (2004). Planning with Disjunctive Temporal Constraints, *Proc. of ICAPS'04 Workshop on Integrating Planning into Scheduling*.

SHIN J. & DAVIS E. (2004). Continuous Time in a SAT-Based Planner, *Proc. of 19th National Conference on Artificial Intelligence (AAAI'04)*, pp. 531-536.

SMITH D.E. (2003). The Case for Durative Actions: A Commentary on PDDL2.1, *Journal of Artificial Intelligence Research*, 20, pp. 149-154.

TARJAN R. (1972). Depth-first search and linear graph algorithms, *SIAM Journal on Computing 1(2)*, pp. 146-160.

TSANG E. (1987). TLP – a temporal planner, *on Advances in artificial intelligence*, John Wiley & Sons, pp. 63-78.

WOLFMAN S. & WELD D., (1999). The LPSAT Engine and its Application to Resource Planning, *Proc. of 16th International Joint Conference on Artificial Intelligence*, pp. 310-317.

YOUNES H.L.S. & SIMMONS R.G. (2003). VHPOP: Versatile Heuristic Partial Order Planner, *Journal of Artificial Intelligence Research*, 20, pp. 405-430.