TLP-GP: New Results on Temporally-Expressive Planning Benchmarks

F. Maris P. Régnier

IRIT, Université Paul Sabatier,
118 route de Narbonne
31062 Toulouse, cedex 9, France.
{maris, regnier}@irit.fr

Abstract

One of the major challenges for planning is to take into account the time dimension. In this paper, we present a simple approach to deal with temporally expressive problems, that is problems for which all possible solutions require concurrency of actions. Our planner TLP-GP mixes some of the advantages of GRAPHPLAN search with a constraint-based and flexible temporal formalism. Its language is consistent with PDDL 2.1 and extends its expressivity. Experimental trials on new temporally expressive benchmarks show the efficiency of our approach and demonstrate the practical possibility of solving temporally expressive problems which up until now were unsolvable by existing techniques.

1. Introduction

To solve real-world planning problems, one of the major challenges to be met is to take into account the time dimension. Indeed, numerous real-world problems require concurrent actions to be solved or to be executed more efficiently. Airport or station management, baking ceramics, cooking… are just some examples. Although many temporal planners have been compared in the International Planning Competitions (IPC), recent theoretical studies have brought to light the limitations of the current approaches to temporal planning. [5] shows that the domains and problems which have been used up until now in the last competitions can always be solved with a sequential plan. They propose a method to prove that a domain can only be solved using concurrent actions. In fact, the winning planners in the IPC competitions, even if they are efficient in a restricted temporal framework, cannot solve problems for which all possible solutions require parallelism (temporally expressive problems) but only those for which there is at least a sequential solution (temporally simple problems). So, they are therefore far from being capable of solving real-world problems. The objective evaluation of these systems requires the setting up of new benchmarks corresponding to temporally expressive problems. [24] prove that solving such problems is EXP-SPACE complete.

In this article, we present an algorithm, called TLP-GP (Temporally Lifted Progression GraphPlan), based on the use of a simplified planning graph and a SAT Modulo Theory (SMT) solver. Our planner can solve problems expressed in a language whose temporal expressivity is greater than that of PDDL2.1 since preconditions can be required and effects can occur on any temporal interval relative to the start-time of an action. TLP-GP can also take into account, in a very natural way, exogenous events as well as temporally extended goals and we propose several different means of extending expressivity even further. It is complete for the temporally expressive sublanguages of PDDL2.1. We compare our planner on a set of new temporally-expressive benchmarks, with two state-of-the-art temporally-expressive planners such as LPGP and VHPOP. These experimental trials show that TLP-GP outperforms these planners and demonstrate the practical possibility of solving temporally expressive problems.

TLP-GP uses a similar method to that used by the family of BLACKBOX [19] planners. It first builds a planning graph without mutexes and constructed in an atemporal manner, without taking into account, to start with, either the duration or the start-time of actions. To extract a solution-plan it can then use two different methods which we compare between them:

- TLP-GP-1 looks for a solution-plan, searching backwards in the planning graph, using a Disjunctive Temporal Problem (DTP) solver (section 2.3.1). To achieve this, it places temporal constraints between actions and uses an agenda which is different than that used by TGP [21]. Instead of placing actions and propositions on a discrete time-line, we use real-valued temporal variables on which constraints are defined. At each back-chaining step, this set of constraints is fed into

a DTP solver which verifies its global consistency. This operation is repeated until a solution is found. In case of failure, another level is added to the graph and the back-chaining process is restarted.

- TLP-GP-2 (section 2.3.2) encodes the graph in a first-order (most commonly Quantifier-Free) Real Difference Logic **(**QF-RDL) and this set of formulas is given to a SMT solver which search for a solution. In case of failure, another level is added to the planning graph and the process is restarted until a solution is found.

In section 2, after giving the representation of actions in TLP-GP, we describe, by means of an example, the expansion of the planning graph and then, the two methods that TLP-GP can use to extract a floating solution-plan. In section 3, we present experimental results obtained on several new temporally expressive benchmarks. We conclude by an overview of the main related works (section 4) and by suggesting topics for future research (section 5).

2. The TLP-GP algorithm

For solving real-world planning and scheduling problems, quantitative temporal constraint networks in the form of the Simple Temporal Problem (STP) [6] are widely known. STP has been extended into Disjunctive Temporal Problem (DTP) [28] to express concepts such as « A occurs before or after B ». DTP admit more than one disjunct in a constraint and a variable may appear in more than one disjunct. To solve a DTP, a simple method is to consider the different STP obtained by selecting one disjunct from each constraint. The computational complexity of solving each STP is polynomial, but the worst-case complexity of solving or verifying the consistency of a DTP is NP-hard [28]. In practice, efficient solving techniques have been developed [30], and tractability results are known for some classes of DTP [25]. SMT solvers are regularly compared during the annual competitions SMT-COMP. The majority of the temporally expressive planners (VHPOP [33], LPGP [21], IxTeT [14, 15, 20]) uses STP to avoid the computational complexity of solving DTP. The processing of disjonctions is then carried out by adding branches in the search tree.

2.1. Representation Language

Even if many extensions that have been proposed for temporal planning (conditions and/or effects over arbitrary intervals of an action execution, predictable exogenous events, deadlines, temporally extended goals…) can be polynomially compiled into PDDL 2.1 [10], our approach directly authorizes an expressivity which is greater than the one of PDDL 2.1.

Actions are represented by 4-tuples (<action-name>, <preconditions>, <effects>, <duration>). The sets <effects> and <preconditions> contain propositions each with a corresponding temporal label. This label represents an interval, defined relative to the start-time of the action, during which a precondition must be verified or an effect is produced. A label [t,t] containing a single value t is written simply as [t]. <duration> represents the duration of the action. We use $\tau_S(A)$ to denote the temporal variable corresponding to the start-time of the action A. We introduce two imaginary and instantaneous actions A_I and A_G which correspond, respectively, to the initial state and the achievement of the goal. The start and end times of a plan are given by $\tau_I = \tau_S(A_I)$ and $\tau_G = \tau_S(A_G)$. **Example:** consider the action (A, $\{a_{[-1,\ 2]}$, $b_{[0]}\}$, $\{\neg a_{[3]}, c_{[5,7]}\}$, 5). If $\tau_S(A)$ is the start-time of A, its duration is 5, the proposition a must hold between $\tau_S(A)$-1 and $\tau_S(A)$+2, b must hold at instant $\tau_S(A)$, ¬a appears at $\tau_S(A)$+3 and c appears at $\tau_S(A)$+5 and remains true at least until $\tau_S(A)$+7.

In the simpliest version of TLP-GP, each precondition or effect p have an associated interval over which p must be true and not p false. This corresponds to "over all" preconditions in PDDL2.1, extended to effects by [5]. However, to represent real-word domains, we have implemented some extensions to PDDL2.1. These extensions allow to represent the fact that a precondition or effect p must be true (and not p false) during a minimal duration d anywhere within an interval [a,b]. Our language also allows the user to disassociate p and not p by stipulating, for example, that p must be true at the end of an interval [a,b] over all of which ¬p cannot be established. It is also possible, in a natural way, to represent external events or goals which have a duration. The former can be easily coded as effects of the dummy action A_I and the latter as preconditions of the dummy action A_G. Here is an example of use of this language.

```
(:durative-action fire-kiln2
 :parameters (?k - kiln20)
 :duration (= ?duration 20)
 :condition (over all (energy ))
 :effect (and
    (somewhere [start (+start 2)] (ready ?k))
    (over [(+ start 2) end[ (ready ?k))
    (at end (not (ready ?k)))))
(:durative-action bake-ceramic1
 :parameters (?p - piecetype1 ?k - kiln)
 :duration (= ?duration 15)
 :condition (over all (ready ?k))
 :effect (and
    (over [start end[ (not (baked ?p)))
    (over [start end[ (baking ?p))
    (at end (not (baking ?p)))
    (somewhere [(- end 5) end] (baked ?p))))
```

```
(:durative-action treat-ceramic1
 :parameters (?p - piece)
 :duration (= ?duration 4)
 :condition (and (over all (baking ?p)))
 :effect (and (minimal-duration 3 anywhere
    [start end] (treated ?p))))
```

In this "temporal-machine-shop-2-3" domain the action fire-kiln is described using (somewhere [start (+ start 2)] (ready ?k)) to express the fact that the kiln will be ready at an unknown instant[1] between start and start + 2. The expression (over [(+ start 2) end[(ready ?k)) is used to enforce the kiln to be ready from start + 2 up to the end. Moreover, the time which is necessary to bake a ceramic is not completely known and can be represented using (somewhere [(- end 5) end] (baked ? p)). Finally, a ceramic can be treated anywhere between the start and the end of the action treat-ceramic with a minimal duration of 3.

In the following, we will use a simple example to explain our algorithm. In this planning problem Π = <O, I, G>, the initial state is I = {}, the goal is G = {b, d, e} and the set of actions is O = {(A, {}, {$a_{[0;5[}$, $\neg a_{[5]}$, $b_{[5]}$, $\neg d_{[5]}$}, 5) ; (B, {$a_{[0]}$}, {$c_{[0;4[}$, $d_{[4]}$, $\neg c_{[4]}$}, 4) ; (C, {$c_{[0]}$}, {$\neg b_{[1]}$, $e_{[1]}$}, 1)}. This problem, given in [4], is interesting because it cannot be solved by any planner based on either of the following assumptions (in particular decision epoch planners): (1) every action will start immediately after some other action has started or ended; (2) the only conflicts preventing an earlier dispatch of an action, however indirect, involve actions which start earlier.

2.2. Expansion of the temporal planning graph

Unlike the temporal planners TGP [27] or LPGP [21], TLP-GP uses a planning graph without mutexes and constructed in an atemporal manner, without taking into account, to start with, either the duration or the start-time of actions. Conflicts between actions, including mutual exclusions, are managed entirely during the solution extraction phase by a constraint satisfaction system. This minimal usage of the planning graph means that TLP-GP does not have the same restrictions as other planners: decision epochs or time-line which limits the completeness to temporally simple problems or a number of levels which is too large to allow the practical resolution of temporally expressive problems (LPGP). The method we have chosen consists in entrusting a large part of the work to a DTP solver. This method has the double advantage of producing floating temporal plans and considerably increasing the expressivity of the representation language that can be used. Moreover, any progress made in the field of SMT solvers is immediately translated by an improvement of the efficiency of TLP-GP. Since an action can produce and destroy the same proposition at different times, we also store in the graph the negations of propositions. Finally, unlike other temporal planners based on GRAPHPLAN, the levels are not linked to a fixed time scale. Level 0 contains the dummy action A_I, which has no preconditions and produces all the propositions of the initial state, together with the corresponding effect arcs. In our running example, we omit level 0 since I is empty. For each level $n \geq 1$, we apply all those actions whose preconditions are all present at level n-1 and we add the corresponding precondition arcs to the graph. We then add the effects of these actions at level n together with the corresponding effect arcs. The graph is built in this way, level by level, until all the goals appear. Finally, we build an extra level containing the dummy action A_G, which has no effects and has all the goals as its preconditions, and we add the corresponding precondition arcs. Fig. 1 shows the graph corresponding to our running example. Each arc in the planning graph has a corresponding temporal label according to its type:

- **Precondition arc** (proposition → action): the label represents the interval, relative to the start of the action, over which the precondition must be verified.
- **Effect arc** (action → proposition): the label represents an interval, relative to the start of the action, at the start of which the effect appears and during which it remains true. After the end of this interval, there is no guarantee that it still holds.

2.3. Extraction of a floating solution-plan

Once the planning graph has been extended to a level at which all goals are present, the extraction algorithm is called and TLP-GP can use two methods to search for a solution-plan.

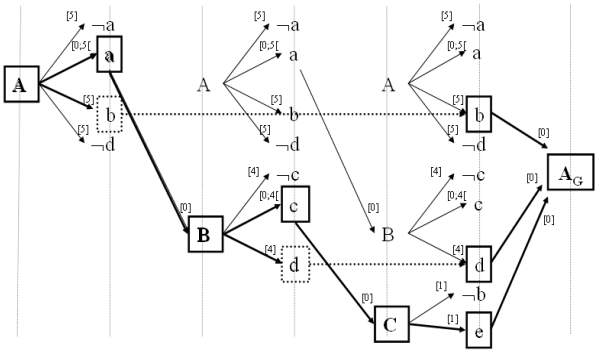
2.3.1. Backward search: TLP-GP-1

Using this method, TLP-GP places temporal constraints between actions and, instead of assigning actions and propositions to fixed time points, it uses mutually constraining real-valued temporal variables. At each step, the set of constraints is fed to a disjunctive temporal constraint satisfaction system which checks its satisfiability. This operation is repeated until a solution is found. In case of failure, an extra level is added to the planning graph and the back-chaining procedure is restarted.

[1] Extensions to « unknown instant » and « not completely known time » only refer to imprecise temporal constraints, not to contingent/uncontrollable ones, which would need the use of specific temporal uncertainty management techniques.

Extraction algorithm

```
Goals ← Pre(A_G);
For every effect e ∈ Eff(A_I):
   Add an interval I to Agenda(e) for the
   apparition of the proposition e;
End For;
While Goals ≠ ∅
   For every proposition p ∈ Goals:
      Goals ← Goals – p;
      Select (* Backtrack point *), using the
      heuristic, an action A to produce p;
      Goals ← Goals ∪ Pre(A);
      Add a precedence constraint between A and
      B, the action whose p is a precondition;
      Add an interval I to Agenda(p) to maintain
      the precondition p;
      For every interval I in Agenda(¬p):
         Add a constraint to forbid the
         overlapping of I and I';
      End For;
      For every effect e of A (except for p when
      the label of p is a singleton):
         Add an interval I to Agenda(e) for the
         apparition of e;
         For every interval I' in Agenda(¬e):
                Add a constraint to forbid the
                overlapping of I and I';
         End For;
      End For;
      Check the consistency of constraints (call
      to the DTP solver);
      In case of failure, return to the back-
      track point to select another action A;
   End For;
End While;
If constraints is satisfiable
   Then return the floating solution-plan
   (selected actions and constraints)
   Else there is no solution in this level;
End If;
End.
```



Agenda	
a	$[\tau_s(A_1); \tau_s(B_2)]$ $[\tau_s(A_1); \tau_s(A_1)+5[$
¬a	$[\tau_s(A_1)+5]$
b	$[\tau_s(A_1)+5]$ $[\tau_s(A_1)+5; \tau_G]$
¬b	$[\tau_s(C_3)+1]$

c	$[\tau_s(B_2); \tau_s(C_3)]$ $[\tau_s(B_2); \tau_s(B_2)+4[$
¬c	$[\tau_s(B_2)+4]$
d	$[\tau_s(B_2)+4]$ $[\tau_s(B_2)+4; \tau_G]$
¬d	$[\tau_s(A_1)+5]$
e	$[\tau_s(C_3)+1; \tau_G]$

Figure 1. Backward search (TLP-GP-1)

The search for a solution requires two data structures: *Agenda* and *Constraints*:

- *Agenda* is a set of lists, one for each proposition. For a proposition p, the corresponding list Agenda(p) is composed of intervals of the form $[\tau_s(A)+\delta_1, \tau_s(B)+\delta_2]$, over which p must be true. When the interval consists of a single value, we denote it by $[\tau_s(A)+\delta]$. Two types of temporal intervals can be added to Agenda(p): (1) Intervals which correspond to a causality relationship between actions (if the proposition p is produced by an action A in order to fulfill a precondition of an action B, p must remain true until used by B). (2) Intervals which correspond to the appearance of the effects of a selected action.
- *Constraints* is a list of disjunctions of (at most two) binary temporal constraints between time points: (1) Constraints corresponding to causality relationships representing the fact that a precondition must be produced (by an action A at time $\tau_s(A)+\delta_1$) before it is required (by an action B at time $\tau_s(B)+\delta_2$). Such constraints are not, in fact, disjunctions since they are of the form $\tau_s(A)+\delta_1 \leq \tau_s(B)+\delta_2$. (2) Constraints imposing the non-intersection of the two intervals $[\tau_s(A)+\delta_1, \tau_s(B)+\delta_2]$ and $[\tau_s(C)+\delta_3, \tau_s(D)+\delta_4]$ over which a proposition and its negation hold. In the most general case, these constraints are disjunctive since they are of the form $(\tau_s(B)+\delta_2 \leq \tau_s(C)+\delta_3) \vee (\tau_s(D)+\delta_4 \leq \tau_s(A)+\delta_1)$.

  The inequalities may or may not be strict depending on the type of intervals under consideration (open, closed or mixed). In most cases, this constraint simplifies to a non-disjunctive constraint.

The set of constraints created by TLP-GP therefore constitutes a disjunctive temporal problem (DTP). The ordering heuristic used in the choice of subgoals to be established and the choice of actions to establish them is the classical « level-based » heuristic [2], [18] which is very efficient in GRAPHPLAN-based planners: priority to subgoals which appear (for the first time) in the highest levels of the graph and, among actions which establish them, priority to those actions which appear (for the first time) in the lowest levels of the graph.

2.3.2. Encoding of the planning graph: TLP-GP-2

The other method than can use TLP-GP is to encode the graph and the set of temporal constraints in a Quantifier-Free Real Difference Logic. The set of formulas is then given to a SMT solver which searches a solution. In case of failure, another level is added to the planning graph and the process is restarted until a solution is found. The advantage of this approach is that it is not necessary to restart the solver at every

stage of the search. On the other hand, the implementation of domain or problem-dependant heuristics to direct the solver search is difficult. The encoding rules are given below.

1. $n_{Init} \wedge n_{Goal}$
2. $\bigwedge_{(n_p,n_b)\in Arcs\,Prec} n_b \Rightarrow \bigvee_{(n_a,n_p)\in Arcs\,Add} Link(n_a, p, n_b)$
3. $\bigwedge_{(n_a,n_p)\in Arcs\,Add} \bigwedge_{(n_p,n_b)\in Arcs\,Prec} Link(n_a, p, n_b) \Rightarrow \begin{matrix} n_a \wedge n_b \\ \wedge\ \tau(n_a \underset{begin}{\longrightarrow} p) \le \tau(p \underset{begin}{\longrightarrow} n_b) \end{matrix}$
4. $\bigwedge_{(n_a,n_p)\in Arcs\,Add} \bigwedge_{(n_p,n_b)\in Arcs\,Prec} \bigwedge_{(n_c,n_{\neg p})\in Arcs\,Del} \left(Link(n_a, p, n_b) \wedge n_c\right) \Rightarrow \begin{matrix} \tau(n_c \underset{end}{\longrightarrow} \neg p) < \tau(n_a \underset{begin}{\longrightarrow} p) \\ \vee\ \tau(p \underset{end}{\longrightarrow} n_b) < \tau(n_c \underset{begin}{\longrightarrow} \neg p) \end{matrix}$

$\bigwedge_{(n_a,n_p)\in Arcs\,Add} \bigwedge_{(n_b,n_{\neg p})\in Arcs\,Del} (n_a \wedge n_b) \Rightarrow \begin{matrix} \tau(n_b \underset{end}{\longrightarrow} \neg p) < \tau(n_a \underset{begin}{\longrightarrow} p) \\ \vee\ \tau(n_a \underset{end}{\longrightarrow} p) < \tau(n_b \underset{begin}{\longrightarrow} \neg p) \end{matrix}$

5. $\left(\tau(n_{Init}) \le \tau(n_{Goal})\right)$

$\bigwedge_{n_a \in NoeudsAction} \begin{matrix} \tau(n_{Init}) \le \underset{p\in Prec(a)}{Min} \tau(p \underset{begin}{\longrightarrow} n_a) \\ \wedge\ \underset{q\in Eff(a)}{Max} \left(\tau(n_a \underset{end}{\longrightarrow} q)\right) \le \tau(n_{Goal}) \end{matrix}$

Figure 2. SMT encoding rules (TLP-GP-2)

- **Rule 1 (Initial state and Goal):** the dummy action nodes Init (producing the initial state) and Goal (needing the goal) are both true.
- **Rule 2 (Conditions production by causal links):** if an action B is active in the plan, then for each of its preconditions, it exists at least one causal link (noted Link(n_A, p, n_B)) from the action A (which produces this precondition) to the action B.
- **Rule 3** (**Action activation and partial order):** if a causal link exists between an action A which produces a precondition p for an action B, then A and B are actives in the plan. Moreover, the instant when A certainly produces p precedes or is the same than the instant when B begins to need p.
- **Rule 4 (Temporally extended mutual exclusions):** if a causal link protects a proposition p and if an action that produces ¬p is active in the plan, then the temporal interval which corresponds to the causal link and the temporal interval which corresponds to the activation of ¬p by the action are disjunctive. If two actions producing respectively a proposition p and its negation ¬p are active in the plan, then the temporal intervals which correspond to the activation of p and ¬p are disjunctive.
- **Rule 5 (Lower and upper bounds):** the initial instant (when propositions of the initial state are true) precedes every instant of the beginning of the preconditions of the other actions. The final instant (when the goal propositions are true) follows all instants of the end of the effects of the other actions.

For the previous example, a solution is found by the solver for the encoding of the 3-level planning-graph (18 prop.l variables, 8 real variables, 88 clauses).

2.3.3. Floating solution-plan

These two methods construct a solution-plan which is a set of selected actions and constraints (Fig. 3). Time-point $\tau_s(x)+\delta$, defined relative to the start time $\tau_s(x)$ of an action x, which occur in the set of constraints are indicated by vertical dotted lines. It's a "floating" solution-plan because the constraints between fluents indicate the maximum possible slide of each action.

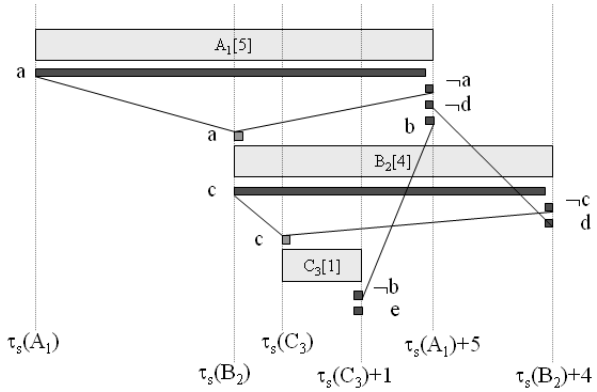


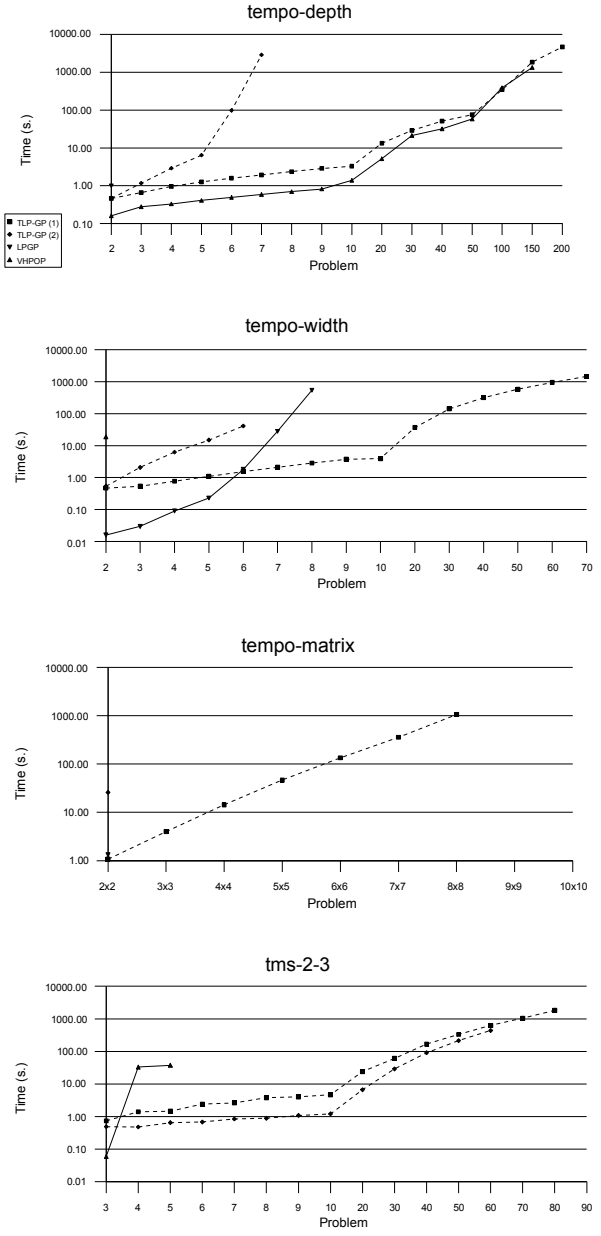
Figure 3. Floating solution-plan

3. Experimental trials

The examples were tested on a 1.6 GHz Pentium M machine with a 512 Mbyte DDRAM. TLP-GP uses the SMT solver MathSat[2] 3.4. We compared TLP-GP with two state-of-the-art planners which are complete for temporally expressive PDDL2.1 problems: the LPGP planner [21], and the partial-plan space planner VHPOP [33]. Even if TLP-GP can solve the first IPC temporal benchmarks in few seconds (Storage-3, DriverLog-1, OpenStacks-1, Depots-1), TLP-GP, as the others temporally-expressive planners (IxTeT, VHPOP, LPGP), is not very efficient on temporally-simple problems. None of these planners, including TLP-GP, really outperforms the others on these temporally-simple problems. Given that the IPC competition benchmarks were inappropriate for our study, since temporally simple, we drew up several new temporally expressive benchmarks which can be found at this adress[3]. The first test domain extend the

[2] http://mathsat.itc.it/
[3] http://tlpgp.free.fr/

problem of [4] in three different ways: (1) the tempo-depth-n problems extend the original problem over a greater number of levels. The solution-plans require n stages of triples of concurrent actions; (2) the tempo-width-m problems require m triples of concurrent actions over only three levels of the graph; (3) the tempo-matrix-n×m problems combine the difficulties of the above two cases.



The last domain, tms-k-t-p, is a simplified version of the previous temporal machine shop domain. It concerns the use of k kilns, each with different baking times, to bake p ceramic pieces (bake-ceramic) of t different types. Each of these types requires a different baking time. They can then be assembled to produce different structures (make-structure). The resulting structures can then be baked again to obtain a bigger structure (bake-structure). On the tested domains, TLP-GP solves more problems than VHPOP and LPGP. Solution-plans are also found faster. The comparison between the two methods of extraction of TLP-GP, shows that TLP-GP-1, which uses the DTP solver, is generally the most efficient. On the domain "tempo-depth", TLP-GP-1 is a little less efficient than VHPOP but it still solves the problem tempo-depth-150 in less than one hour, with 2,095 simple (STC) and 1,347 disjunctive constraints (DTC). On the domain "tempo-width", TLP-GP-1 solves all the problems. The problem tempo-width-70 is the last problem solved in less than one hour whereas the DTP contains 5,598 simple and 29,469 disjunctive constraints. On the domain "tempo-matrix", TLP-GP-1 is the only planner that solves more than one problem. It gives a floating solution-plan for the problem tempo-matrix-8×8 in less than 20 mn. whereas the DTP contains 1,416 simple constraints and 4,864 disjunctive constraints. On the domain "tms-2-3", TLP-GP-1 solves 29 problems in less than 1 minute. The problem tms-2-3-80 is the last problem solved; it needs 30 mn. to be solved whereas the DTP contains 5,406 simple constraints and 1,569 disjunctive constraints. For this last problem, the floating solution-plan contains 277 actions. On all these temporally-expressive problems, the makespan of the compacted floating solution-plan given by TLP-GP, is sometimes smaller than for VHPOP or LPGP, but never greater.

We give summarily here some results on two more complicated benchmarks. The complete results and enlarged figures can be found here[4]. LPGP and VHPOP cannot handle such benchmarks because the expressivity of their langage is not sufficient. The "cooking" domain allows to plan the preparation of a meal, as well as its consumption by respecting constraints of warmth. Problems cooking-carbonara-n which we used for this test allow to plan the preparation of n dishes of pasta. The concurrency of actions is required to obtain the goal because it is necessary that the electrical plates works so that water and oil are hot to cook pasta and bacon cubes. It is also necessary to perform this baking in parallel to serve a hot dish during its consumption. On this domain, TLP-GP can solve problems up to 23 dishes of pasta in less than one hour. In the domain temporal-machine-shop-2-3 (extended version of the domain tms-2-3), the last problem solved by TLP-GP in less than one hour is temporal-machine-shop-2-3-80 with a solution-plan of 324 actions. After simplification of constraints, the "cooking" domain only needs the resolution of a STP to find a solution-plan, where the "temporal-machine-shop-2-3" domain requires the resolution of a DTP.

[4] http://tlpgp.free.fr/

4. Related works

Temporal planners are primarily based on three types of algorithms:

Planning in extended state-space is currently the technique which produces the best results, but the expressivity of the languages remains weak. Planners of this type use the state of the world as the central attribute in a temporal state, which allows them to use the well-tried techniques of heuristic search employed in atemporal planning. They are very efficient, which is demonstrated by the fact that SGPLAN [3] won the IPC 2004 and 2006 competitions in the temporal planner category. Most of these systems restrict the possible starting times of an action to specific instants called decision-epoch, which makes them incomplete for many problems requiring simultaneous actions [4]. To try to minimise overall plan makespan, some of these planners can (for temporally simple problems) produce plans with concurrent actions [3] but they cannot handle temporally expressive problems. They are only complete for certain sublanguages of PDDL2.1 [9] which can eventually be reduced to STRIPS [4]. [4] propose a theoretical state-space temporally expressive algorithm, exploiting the idea of lifting over time: the decisions about « when to execute » are delayed until all of the decisions about « what to execute » have been made. The planner CRIKEY [16] attempts to do classical planning, and, in case of failure, switches to a TEMPO like search to handle actions that could lead to required concurrency. It can be considered as an implementation of TEMPO but [4] note that CRIKEY is not complete.

The plan-space planning approach has also been extended to the temporal framework. The first HTN (Hierarchical Tasks Network) planners to introduce a temporal aspect to problems were DEVISER [31] and FORBIN [22]. More recently, the planner IxTeT [14], [15], [20] used specific procedures to manage almost linearly the addition of temporal constraints in a lattice of time-points. Its language is very expressive and IxTeT can manage explicit time, resources, uncertainty and plan repair at execution but results[5] of the IPC'02 demonstrate that IxTeT is less efficient than VHPOP (see below). The planning and scheduling system HSTS [23], also provides general devices for representing complex states and causal justifications. It has been used for several unconventional domains (scheduling for the Hubble space telescope and troops transportation planning). Classical partial-order planners (POP) have also been extended to the temporal framework with the planners VHPOP [33] and DT-POP [29]. These POP-type techniques are particularly interesting when searching for optimal plans; the planner CPT2 [32] is among the temporal planners which obtained the best results, notably at the IPC'06 planning competition in the optimal temporal planning category but the representation of actions restricts it use to temporally simple languages.

Extensions of GRAPHPLAN: the use of planning graphs [1] has also been extended to temporal problems by several planners. The system TGP [27] which uses a temporally simple language, was the first such planner. In the algorithms based on the TGP representation, the construction of the graph is entirely guided by a discrete time-line : at each step, the planner considers the next instant when a change can occur. It follows that each action starts immediately after the start or the end of another action. TPSYS [11] and CPPLANNER [7] are other planners which provide a temporally expressive language. Unfortunately, the start-times of actions remain linked to fixed decision-epoch, which renders these algorithms incomplete for temporally expressive domains [4]. TGP, TPSYS and CPPLANNER are nevertheless complete for the temporally simple sublanguages of PDDL2.1. Other systems, such as LPG-TD [13] (or the planners SGPLAN [3] and MIPS [8]) only consider sequential plans which they optimize using temporal information in order to render them concurrent. LPGP [21] is one of the most expressive among recent planners. Unlike previous systems, it is complete for temporally expressive languages but splits actions into three components: a start, invariant, and end action. This is why a large number of levels need to be built, which makes the problems more difficult to solve. [17] presents a planner that uses a CSP encoding of an LPGP-style planning graph to perform planning in temporally expressive domains. As TLP-GP it uses constraints over temporal timestamp variables.

From the point of view of the constraints taken into account in the description language, [12] presents a formulation model which is close to our. Nevertheless, the main purpose of this work is to encode a plan (not a problem) with complex constraints as a CSP in order to incorporate such a modelling as a part of an integrated planning and scheduling module. [26] presents TM-LPSAT, a domain-independent planning SAT-based system which uses a similar approach to the one of TLP-GP-2. It can reason about actions over continuous time. The planning problem is first compiled into a system of boolean combinations of propositional atoms and linear constraints over numeric variables. A SMT solver is then used to find a solution from which a correct plan is extracted.

5. Conclusion and future work

In the majority of the temporally expressive planners, the use of STP implies that the processing of

[5] http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume20/long03a-html/node5.html

disjonctions must be carried out by adding branches in the search tree. Even if the computational complexity of solving DTP is NP-hard, the planner TLP-GP demonstrates that the use of this technique can, in practice, lead to good performances in solving temporally expressive benchmarks. On temporally expressive benchmarks our planner clearly outperforms the two state-of-the-art temporally expressive planners VHPOP and LPGP. The expressivity of the language of TLP-GP is greater than PDDL2.1 and no compromise needs to be made in terms of completeness in order to achieve this expressivity. These results indicate the possibility of representing and solving problems which are closer to real-world applications. The production of a floating solution-plan rather than a single fixed solution also allows for a greater flexibility during the execution of the plan.

At present, we are working to improve TLP-GP by investigating two main ideas. First at all, we study the possibility of adding new constraints incrementally in a DTP solver instead of restarting the solver from scratch at each successive call. Secondly, in the current state of TLP-GP, the usual mutex reasoning is not relevant. All the conflicts between actions are entirely managed during the solution extraction phase by a solver. As the mutex management is a very strong and efficient mechanism which contributes to the performance of GRAPHPLAN, we study the temporal constraints to determine new kinds of mutex-like constraints which could be calculated during the graph construction.

6. References

[1] A.Blum, M.Furst, "Fast Planning Through Planning Graph Analysis", IJCAI, 1995.

[2] M.Cayrol, P.Régnier, V.Vidal, "Least commitment in GRAPHPLAN", Artificial Intelligence, 130, pp. 85-118, juin 2001.

[3] Y.X.Chen, B.W.Wah, C.W.Hsu, "Temporal planning using subgoal partitioning and resolution in SGPlan", Journal of AI Research, 2006.

[4] W.Cushing, S.Kambhampati, Mausam, D.S.Weld, "When is temporal planning really temporal ?", IJCAI, 2007.

[5] W.Cushing, S.Kambhampati, K.Talamadupula, D.S.Weld, Mausam, "Evaluating temporal planning domains", ICAPS, 2007.

[6] R.Dechter, I.Mieri, J.Pearl, "Temporal constraint networks", AI, 49: 61-95, 1991.

[7] T.B.Dinh, B.M.Smith, "CPPlanner: A Temporal Planning System using Critical Paths", APES, 2003.

[8] S.Edelkamp, M.Helmert, "The Model Checking Integrated Planning System", AI Magazine, 22(3), 2001.

[9] M.Fox, D.Long, "PDDL2.1 : An Extension to PDDL for Expressing Temporal Planning Domains", Journal of AI Research, 20, 2003.

[10] M. Fox, D. Long, K. Halsey, “An Investigation into the Expressive Power of PDDL2.1”, ECAI'04, 328-342, 2004.

[11] A.Garrido, M.Fox; D.Long, "TPSYS: Temporal Planning with PDDL2.1", ECAI, 2002.

[12] A. Garrido, E. Onaindía, M. Arangu, "Using Constraint Programming to Model Complex Plans in an Integrated Approach for Planning and Scheduling", Proc. of 25th UK Planning and Scheduling SIG Workshop, pp. 31-38, 2006.

[13] A.Gerevini, A.Saetti, I.Serina, "An Approach to Temporal Planning and Scheduling in Domains with Predictable Exogenous Events", JAIR, 25, 2006.

[14] M.Ghallab, A.M.Alaoui, "Managing Efficiently Temporal Relations Through Indexed Spanning Trees", IJCAI, 1989.

[15] M.Ghallab , H.Laruelle, "Representation and Control in Ixtet, a Temporal Planner", AIPS, 1994.

[16] K.Halsey, D.Long, and M..Fox,"CRIKEY - A Planner Looking at the Integration of Scheduling and Planning", Proc. of the “Integration Scheduling Into Planning” Workshop, ICAPS'03, pp. 46-52, 2004.

[17] Y.Hu, “Temporally-Expressive Planning as Constraint Satisfaction Problems”, ICAPS'07, 2007.

[18] S.Kambhampati, R.S.Nigenda, "Planning as constraint satisfaction: Solving the planning graph by compiling it into CSP", Artificial Intelligence, 132 (2001).

[19] H.Kautz, B.Selman, "Unifying SAT-based and Graph-based Planning", IJCAI, 1999.

[20] S.Lemai, F.Ingrand, "Interleaving Temporal Planning and Execution in Robotics Domains", AAAI'04, 2004.

[21] D.Long, M.Fox, "Exploiting a graphplan framework in temporal planning", ICAPS, 2003.

[22] D.Miller, J.Firby, T.Dean, "Deadlines, travel time and robot problem solving", IJCAI, 1985.

[23] N.Muscettola, "HSTS: integrating planning and scheduling". In Zweben, M., and Fox, M., eds., *Intelligent Scheduling*, 169-212. Morgan Kaufmann, 1994.

[24] J.Rintanen, "Complexity of Concurrent Temporal Planning", ICAPS, 2007.

[25] T.K. Satish Kumar. On the tractability of restricted disjunctive temporal problems. In *Proc. of ICAPS’05*, pages 110–119, Monterey, CA, 2005.

[26] J.A.Shin, E.Davis: "Processes and continuous change in a SAT-based planner". Artificial Intelligence 166(1-2): 194-253, 2005.

[27] D.E.Smith, D.Weld, "Temporal planning with mutual exclusion reasoning", IJCAI, 1999.

[28] K. Stergiou and M. Koubarakis. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, 120(1):81–117, 2000.

[29] P. Schwartz, M. E. Pollack, "Planning with Disjunctive Temporal Constraints," ICAPS'04 Workshop on Integrating Planning into Scheduling, 2004.

[30] I. Tsamardinos and M. E. Pollack, "Efficient solution techniques for disjunctive temporal reasoning problems". *Artificial Intelligence*, 151(1–2):43–89, 2003.

[31] S.A.Vere, "Planning in time: windows and durations for activities and goals", IEEE Transactions on Pattern Analysis and Machine Intelligence, 5(3), 1983.

[32] V.Vidal, H.Geffner, "Branching and Pruning: An Optimal Temporal POCL Planner based on Constraint Programming", AI, 170 (3), 2006.

[33] H.L.S.Younes, R.G.Simmons, "VHPOP: Versatile Heuristic Partial Order Planner", Journal of AI Research, 20, 2003.