TLP-GP: a Planner to Solve Temporally-Expressive Problems

F. Maris **P. Régnier**

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

Introduction

TLP-GP [Maris, Régnier, 2008], is a planner based on the use of a simplified planning graph and a Disjunctive Temporal Problem (DTP) solver. It 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 natural way, exogenous events and temporally extended goals which must be true over a certain period of time. It is complete for the temporally expressive sublanguages of PDDL2.1.

The TLP-GP Algorithm

TLP-GP [Maris, Régnier, 2008] uses a similar method to that used by the family of BLACKBOX [Kautz, Selman, 1999] and GP-CSP [Do, Kambhampati, 2001] planners. The planning graph is built until the goal are obtained, following the classic algorithm for the atemporal case, without the calculation of mutexes. TLP-GP then looks for a solution-plan searching backwards in the planning graph, using a Disjunctive Temporal Problem (DTP) solver. To achieve this, it places temporal constraints between actions and uses an agenda which is different to that used by TGP.
Instead of a placing actions and propositions on a discrete time-line, TLP-GP uses real-valued temporal variables on which constraints are defined. At each back-chaining step, this set of constraints is fed into a solver which verifies global consistency. This operation is repeated until a solution is obtained. In case of failure, another level is added to the planning graph and the back-chaining process is restarted.

Representation Language

Actions are represented by 4-tuples (<action-name>, <preconditions>, <effects>, <duration>). <effects> and <preconditions> are sets of 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 $\tau_I = \tau_s(A_I)$ et $\tau_G = \tau_s(A_G)$.

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 $\tau_s(A)$, $\neg$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 [Cushing & al., 2007.b]. However, to represent real-word domains, we have implemented a more expressive language in which we can 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 not 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.

```
(: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 the "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 between start and start + 2. The expression (over [(+ start 2) end[(ready ?k)) is used to enforce the kiln to be ready up to the end. Moreover, the time wich is necessary to bake a ceramic is not completely known and can be represented using (somewhere [(- end 5) end] (baked ?p)). A ceramic can be treated anywhere between the start and the end of the action *treat-ceramic*.

Expansion of the Temporal Planning Graph

Unlike the planners TGP [Smith, Weld, 1999] or LPGP [Long, Fox, 2003], 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. This strategy turned out to be very effective in the context of classical planning (cf. the SATPLAN'04 system, successor to BLACKBOX [Kautz, Selman, 1999], which is still the winner in the optimal-planner category of the IPC'06 competition).

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 [Blum, Furst, 1995], 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 [Maris, Régnier, 2008], 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.

The extraction algorithm is then called. 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.

Extraction of a Floating Solution-Plan

Once the planning graph has been extended to a level at which all goals are present, TLP-GP searches backward for a solution-plan. To this end, it 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 restarted. The simplified extraction algorithm is given below. 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):

- 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).
- Intervals which correspond to the appearance of the effects of a selected action (appearance of p).

Constraints is a list of disjunctions of (at most two) binary temporal constraints between time points:

- 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 $\tau_s(A)+\delta_1 \leq \tau_s(B)+\delta_2$.

- 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 problem of solving or verifying the consistency of a DTP is NP-hard [Dechter, Mieri, Pearl, 1991] but the performance of the algorithms to solve these problems are regularly improved. The default ordering heuristics used in the choice of subgoals to be established and the choice of actions to establish them is the following: priority is given to subgoals which appear (for the first time) in the highest levels of the graph and, among actions which establish them, priority is given to those actions which appear (for the first time) in the lowest levels of the graph.

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, e ≠ p:
         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.
```

Use of TLP-GP

Generalities

TLP-GP is implemented in OCaml 3.09.2. TLP-GP uses the SMT (Sat Modulo Theory) solver MathSat[1] 3.4 which is known to perform well on DTP (cf. results of the SMT-COMP'06[2] competition). The archive contains the source code of TLP-GP and statically linked binaries of MathSat for Linux. Use "make" to build the binaries of TLP-GP. If you want to perform the program on an other operating system, you have to ask the authors for the good binaries of MathSat.

The command line to run TLP-GP:

```
./tlp-gp domain.pddl problem.pddl
```

Experimental results

We compared TLP-GP with two state-of-the-art planners capable of solving temporally expressive problems: the LPGP [Long, Fox, 2003] planner which is an extension of GRAPHPLAN, and the partial-plan space planner VHPOP [Younes, Simmons, 2003]. Given that the previous temporal benchmarks were inappropriate, since temporally simple, we drew up several new temporally expressive benchmarks. All these benchmarks can be found at the adress which is given below[3].

The first test domains extend the problem of [Cushing et al., 2007.a, figure 3] in three different ways. The domain, tms-k-t-p (temporal machine shop, [Cushing et al., 2007.a]) is inspired by a real-world application. 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. These ceramics 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*). 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.

As the expressivity of LPGP and VHPOP is less than that of TLP-GP, we tested simplified, but nevertheless temporally expressive, versions of these benchmarks. On these benches TLP-GP clearly outperforms LPGP as well

[1] http://mathsat.itc.it/
[2] http://www.smtcomp.org/2006/
[3] http://tlpgp.free.fr/

as VHPOP. The results can be found at the address given below[4].

For the "cooking-carbonara-01" problem, TLP-GP finds a floating solution-plan and returns an example of valid static plan providing possible start times.

```
-----------------------------------------
ACTION (TLP-GP level) [possible start time]
-----------------------------------------
Init (0) [0.000000]
FIRE_COOKING_PLATE_p1 (1) [0.000000]
BREAKING_EGGS_e1 (1) [0.000000]
BOILING_WATER_p1_w1 (2) [1.000000]
BOILING_OIL_p1_o1 (2) [1.000000]
COOKING_NOODLES_w1_n1 (3) [3.000000]
COOKING_LARDON_o1_l1 (3) [3.000000]
COOKING_CARBONARA_n1_l1_e1 (4) [9.000000]
EATING_CARBONARA_n1_l1_e1 (5) [10.000000]
Goal (6) [17.000000]
-----------------------------------------
```

The floating plan can be recovered using the files default.smt and plan.txt which contain the constraints and variables assignments. In a future version of TLP-GP we shall provide a graphic interface which will allow exploiting floating solutions-plans.

Conclusion

We have presented TLP-GP, a planner which can solve temporally expressive problems in a language whose expressivity is greater than PDDL2.1. No compromise needs to be made in terms of completeness in order to achieve this expressivity. On temporally expressive benchmarks, TLP-GP performed much better then two state-of-the-art planners (LPGP and VHPOP) capable of solving the same types of problem. 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.

References

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

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

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

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

[Do, Kambhampati, 2001] M.B.Do, S.K Kambhampati, "Planning as Constraint Satisfaction: Solving the Planing Graph by compiling it into CSP", AI, 132, 2001.

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

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

[Maris, Régnier, 2008] F.Maris, P.Régnier, "TLP-GP: Solving Temporally-Expressive Planning Problems", TIME, 2008.

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

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

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