Temporal Planning with a Non-Temporal Planner

Keith Halsey

Department of Computer Science,
University of Durham, UK
keith.halsey@durham.ac.uk

Introduction

This extended abstract looks at the work that I am currently undertaking in the field of temporal planning. Temporal planning is the same problem as classical planning but, whereas in classical planning all actions are considered to be instantaneous, in temporal planning time is also modelled. One possible way to achieve this is to introduce actions with duration. These durations can rely on the parameters of the action (e.g. a plane takes longer to fly the further the distance between its departure and arrival cities). Relaxing this one assumption complicates the problem, since now a new metric is introduced to judge plans by; it is no longer the plan with the fewest actions which could be seen as best, but that with the shortest total duration. Concurrency can now be exploited within the plan, where two or more actions can overlap and be executed simultaneously. This replies on the capabilities of the executive and, as described later, the expressiveness of the description language. Concurrent actions must not interfere with each other; this is another challenge for the planner. Temporal planning can be seen as the merging of classical planning and scheduling.

Whilst classical planning, with its simplifying assumptions, is still a hard problem, good progress has been made over recent years (see Weld 1999). This partly led to the introduction of temporal planning (and planning with resources) at the AIPS2002 planning competition (Fox and Long 2002), and is seen as a current major challenge. The purpose of this work is to exploit existing classical planning technologies with some common solving strategies from scheduling to solve temporal planning problems, whilst exploiting any concurrency where it is available in the problem.

Overview of the Architecture

I have designed a temporal planner that works by pre-processing durative action descriptions into collections of instantaneous actions, building a plan using these actions with classical planning technology and then post-processing the resulting plan into a concurrent plan.

Figure 1 shows an abstracted overview of how the system works. Firstly, a temporal planning domain and problem is passed through a translator which takes out the temporal aspects, converting it to an equivalent STRIPS-like domain that preserves all the key temporal relationships. It stores the duration of the actions that it has removed from the original files in a separate file. These two STRIPS files are then passed through a classical planner. This is where the 'hard' work is done, but should be easier without the temporal information. The totally ordered sequential plan is passed through a program that produces a partially ordered plan, allowing actions that can be executed together to be happen concurrently, keeping only essential orderings in place. This partial ordering, along with the duration file created by the translator, are passed into a program that uses a simple temporal network at its core. This calculates the relative and actual timings of the actions, to produce a valid temporal plan. Each of the four main systems will be looked at in detail in the following section.

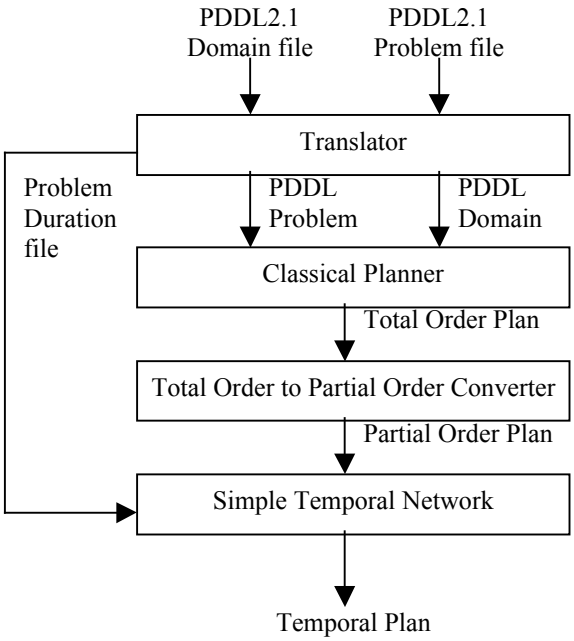

Figure 1 - Architecture Overview

The System

The Translator and PDDL2.1

As with any planning technology, the choice of how the problem is represented has a great impact on how the problem is solved. What can be described can change significantly how easy or hard the problem then becomes. PDDL2.1 (Fox and Long 2001) seemed a natural choice of description language for a number of reasons:

- It is an extension of PDDL (McDermott 1998), used in previous planning competitions, which is the de facto language for classical planning. Therefore, translation between PDDL2.1 and PDDL is simple (as described later).
- As with PDDL, it is domain independent, so no advice is given to the planner as to how to solve the problem.
- PDDL2.1 is split into levels, corresponding to the degree of expressiveness (and associated difficulty) of the problem.
- It is simple yet descriptive as to what it allows to happen concurrently. Conditions may hold at the start, at the end and over the duration of the action. Similarly, effects may happen at the start and end of the action, whilst some levels allow effects to take place over the duration of the action.
- PDDL2.1 was used in the AIPS2002 planning competition, so not only are there a number of domains readily available, but it is easy to compare this system against other temporal planners that entered the competition.

There is one major subtly of the validity of PDDL2.1 plans. Unlike classical planning, it is not just the order of the actions in the plan that ascertains its validity, but also the times when the actions are scheduled to take place. A temporal plan is a sequence of time stamped actions with associated duration. It becomes unclear as to the validity of a plan if the end of one action deletes or achieves the precondition of the start of another action at the precise moment that that action is scheduled to start. That is to say, is it possible to have P and $\neg P$ true at the same time? On a more practical note, any executive will only be able to execute the plan to a certain degree of accuracy with regard to the timings. Therefore, PDDL2.1 introduces '□', or *tolerance value*: the minimum time between one action achieving another. If it is less than this value then the plan is invalid according to PDDL2.1 semantics (Fox and Long 2001).

In the level of PDDL2.1 that this systems uses (level 3), continuous effects are not permitted in durative actions. As mentioned earlier, PDDL2.1 durative actions can have conditions that must hold at the start of the action or at the end of the action. Effects can also occur at the start or at the end. Invariants are propositions that must hold for the duration of the action. This conveniently allows a durative action to be split into three instantaneous actions; one for the start with its preconditions and effects, one for the end with its preconditions and effects, and one to represent the checking of the invariant. This can be shown in Figure 2. There are two extra propositions added during the conversion process. The first, load-truck-inv, is an effect of the start and invariant action and a condition of the invariant and end action. The second, iload-truck-inv, is an effect of the invariant action and a condition of the end action. Both of these ensure not only that all three actions are chosen during the planning process but also that they appear in the correct order within the plan.

The translator converts durative actions in this way. It was originally written for LPGP (Long and Fox 2002), a temporal graphplan based planner, and was only altered very slightly so that it would work with the classical planner described below. The durations file, for the example in figure 2, would contain an entry indicating that the LOAD-TRUCK action takes 2 time units. This corresponds to the Simple Time domains used in the competition. The translator can also translate actions whose duration depends on some of the parameters (the Time domain variants). In this case, it calculates the

```
(:durative-action LOAD-TRUCK
  :parameters (?obj - obj ?truck - truck ?loc - location)
  :duration   (= ?duration 2)
  :condition (and (over all (at ?truck ?loc))
                  (at start (at ?obj ?loc)))
  :effect     (and (at start (not (at ?obj ?loc)))
                  (at end (in ?obj ?truck))))
```

becomes

```
(:action LOAD-TRUCK-START
  :parameter (?obj - obj ?truck - truck ?loc - location)
  :precondition  (at ?obj ?loc)
  :effect        (and (not (at ?obj ?loc))
                  (load-truck-inv ?obj ?truck ?loc)))

(:action LOAD-TRUCK-INV
  :parameter (?obj - obj ?truck - truck ?loc - location)
  :precondition  (and (at ?truck ?loc)
                  (load-truck-inv ?obj ?truck ?loc))
  :effect        (and (load-truck-inv ?obj ?truck ?loc)
                  (iload-truck-inv ?obj ?truck ?loc)))

(:action LOAD-TRUCK-END
  :parameter (?obj - obj ?truck - truck ?loc - location)
  :precondition  (and (iload-truck-inv ?obj ?truck ?loc)
                  (load-truck-inv ?obj ?truck ?loc)))
  :effect        (and (in ?obj ?truck)
                  (not (load-truck-inv ?obj ?truck ?loc))
                  (not (load-truck-inv ?obj ?truck ?loc)))
```

Figure 2 - Translation from a Durative Action to 3 Instantaneous Actions

duration for all possible combinations of the depending parameters and writes these to the duration file.

The Classical Planner

The classical planner used is FF. This is a successful forward heuristic plan-space search planner that uses a heuristic created from a relaxed plan graph. It takes in the translated domain and problem description file and produces a totally ordered plan. Again, FF had to be altered slightly so that it not only prints out the actions that form the plan, but also the preconditions and the add and delete effects for each of those actions. This is necessary for the next stage to find out what actions achieve and threaten others. Also printed out are the goals of the problem. Again, this is used in finding the partial order.

Total Order to Partial Order Conversion

To convert to the partial order plan, only those actions that interact need be ordered. These arise in two cases; firstly, where one action achieves another, and secondly, where one action threatens another by deleting a precondition. A greedy algorithm that works backwards through the plan, looking for achieving and threatening actions is used and described in figure 3 (Moreno et al 2002). Although this will not find an optimal plan, that is to say one that exploits all concurrency possible, it is complete and sound (i.e. it will find a valid partial order plan).

for i = n down-to 1 do

1. for each *procond* ∈ Preconditions(op_i)
   Find an operator op_j in plan with effect *precond*
   Add an ordering from op_i to op_j
2. for each *del* ∈ Delete-Effects(op_i)
   Find all operators with precondition, a delete effect of op_i
   Add an ordering from these to op_j
3. for each *add* ∈ PrimaryAdd(op_i) (if it appears in the goal or sub-goal chain)
   Find all operators that delete any primary adds of op_i
   Add an ordering from these to op_i

Figure 3 - Total Order to Partial Order Algorithm

As can be seen from step 3 of the algorithm, the goal and sub-goal chain are needed. Unfortunately these are lost after the planning phase. However, by recursively storing the preconditions of any actions that achieve a goal or sub goal, these can soon be found out again.

The Simple Temporal Network

Simple Temporal Networks (STNs) (as described in Dechter, Meiri and Pearl 1989) take a set of constraints of the form:

$$b_1 \leq x - y \leq b_2$$

These describe the minimum (b_1) and maximum (b_2) time between actions (x,y) and are put into a graph to allow reasoning to occur, namely propagation of constraints, and checking their satiability. The orderings from the partial order plan are converted into this form. The maximum time between two ordered actions is infinity and the minimum 0.01 (or whatever ε is set to). So if LOAD-END must precede DRIVE-START:

$$0.01 \leq DRIVE\_START - LOAD\_END \leq \infty$$

However, if the ordering is an invariant action before another action, it is made between the corresponding end action and the other action. If it is an invariant action that occurs second, then the ordering is between that action and the corresponding start action of the invariant action. In both case, the minimum time allowed to elapse is set to zero (i.e. the two actions could abut). This happens because the invariant action is not actually an instant, but the duration of the action. Therefore this does not need scheduling, but rather anything preceding it, must happen before or at the same time as the action starts, and anything following it must happen after, or at the same time as the durative action finishes. This is all necessary for the protection of invariant conditions.

For this reason it is also important at this point to match up the corresponding start, end and invariant actions. It can be the case that there is more than one durative action with the same name and parameters in the plan. In this case there will be more than one of its instantaneous actions. To match them up correctly (i.e. the correct start with the correct end), a greedy approach is taken whereby the first start is matched to the first invariant found, which in turn is matched with the first end action found for that combination of parameters. And so forth through the plan. Identical actions in the must have the same duration (even in Time domain variants) so it can never be the case that the classical planner had interleaved identical with the intention of one action occurring and completing during the duration of the other. If this were the case, the greedy approach would not work.

Lastly, the durations are expressed as constraints. Once again it must be known which instantaneous actions pair up to form durative actions. The constraints expressed state that the minimum and maximum time between the start and end action equals its duration. For example, if a drive action has a duration of 8, then the following constraint would be added to the STN:

$$8 \leq DRIVE\_END - DRIVE\_START \leq 8$$

Each constraint can be seen as two edges on a graph with the weights representing the minimum and

maximum time differences, and the vertices being the instantaneous actions. In this system, the graph is represented as an 2n+1 x 2n+1 array, where n is the number of durative actions (and so 2n is the number of start and end actions, with the extra one as a special timepoint to represent the start of the plan). By running Floyd-Warshalls (Gallo and Pallottion 1988) algorithm on this, the transitive closure of the graph is be calculated.

Extracting the Durative Action Times

Once the transitive closure of the graph has been calculated, it is possible to look at the earliest and latest possible times that any instantaneous action occurs. Figure 4 shows the greedy algorithm to find out the exact times at which the durative actions occur (i.e. at what time the start instantaneous actions occur). It continually finds the potentially latest finishing action, and then sets that to the earliest it could possibly finish. However, it then must re-compute the transitive closure as changing this latest possible finishing time may in turn change (although only ever decrease) the latest possible finishing times of other actions. Computing this transitive closure with Floyd-Warshalls is $O(n^3)$ and this must be done at most once for each durative action.

The plan finally produced, along with the original problem and domain file, can be passed into the validator (Howey and Long 2003) to check on its validity.

Find the latest possible instantaneous action where its latest possible time does not equal its earliest possible time.

1. Set this action's latest possible time to equal its earliest possible time.
2. Re-compute transitive closure.

Loop until all actions latest and earliest possible times are equal.

Figure 4 - Algorithm for Setting the Durative Action Times

Initial Results

To gain some initial results, the quality of the plans produced and the time it takes to produce the plans were compared against LPG (Gerevini and Serina 2002) and MIPS (Edelkamp and Helmert 2000) on the Driverlog domain, both the Simple Time and Time variants, as used in the AIPS2002 planning competition (Fox and Long 2002). LPG is a planner based on local search and planning graphs and was awarded "Distinguished Performance of the First Order" at the competition. The version used here trades time spent planning to produce better quality plans. MIPS works similar to this planner as it splits up durative actions, and then combines symbolic and explicit heuristic search planning. It received the "Distinguished Performance" award. Table 1 compares the three planners in the Simple Time Driverlog domain, and table 2 compares them in the Time Driverlog domain.

	This Planner		LPG		MIPS	
	Time	Quality	Time	Quality	Time	Quality
1	360	91.05	30	91.082	110	302.1
2	580	100.03	90	92.073	260	246.22
3	450	40.02	10	40.021	110	173.1
4	670	89.03	1740	52.033	330	250.18
5	1080	109.04	40	51.042	160	163.2
6	780	64.04	340	52.052	239	238.14
7	700	51.03	20	40.021	260	287.15
8	2620	151.06	12940	52.052	11380	320.28
9	313580	284.15	47100	92.073	349	403.24
10	2730	91.04	15300	38.037	440	231.23
11	3590	74.01	123020	65.064	689	306.23

Table 1 - Driverlog Simple Time Domain

	This Planner		LPG		MIPS	
	Time	Quality	Time	Quality	Time	Quality
1	340	302.05	10	302.008	100	302.1
2	610	341.03	15130	246.023	140	246.22
3	460	173.02	10	173.011	110	173.1
4	660	392.03	800	249.017	349	250.18
5	1130	306.04	12130	102.022	149	163.2
6	800	260.04	30	168.011	330	238.14
7	690	268.01	13820	200.02	250	287.15
8	2610	527.06	104800	206.029	3470	320.28
9	309450	1065.15	4420	345.026	330	403.24
10	2760	259.04	48640	93.037	409	231.23
11	3980	430.01	50	232.024	400	306.23

Table 2 - Driverlog Time Domain

As these are only initial results, and are only indicative, no formal analysis has been preformed on them. In the simple time domain, this planner would seem to scale better than LPG although at first slower. It is slower than MIPS, but scales at a similar rate. Whilst no optimisation has been performed on any of the code I have written, the time to execute this code is insignificant, with most of the time being spent planning by FF. This planner consistently produces better quality plans than MIPS, but equal or worse in quality than LPG.

In the time variant, the picture is less clear. Whilst it is generally slower than MIPS, its performance varies compared to LPG. With regard to quality, it is generally the slowest out of the three planner, although is still competitive.

Conclusions

I have achieved writing a version of "TemporalFF" for Simple Time, Time and, as described below, soon, Complex domains. The idea is similar to MIPS (Edelkamp and Helmert 2000) as it uses pre and post processing of the domain and plan. But the planner described here is

potentially more powerful as it has the ability to exploit start effects of actions. This is because the translation phase is structure preserving. Originality in this planner lies in its 'plug-ability' as the language translator gives generality. As can be seen from Figure 1, it is possible to replace any of the components, most importantly the classical planner, with a functionally similar program. This distinguishes it from MIPS which any changes affect the whole algorithm. As is described next, replacing the planner with one with increased capabilities results in this planner inheriting those capabilities without any other changes required. If the planner is replaced by a partial order planner, there would be no need to lift a partial order plan, reducing the effort needed.

Opportunities for Improvement

There are two main opportunities for improving the quality of the plans produced. The first is in the algorithm which lifts the partial order from the total order. As already observed, whilst this greedy approach is sound and complete, it may be the case that better partial order plans, which better exploit concurrency, could be found with some search. There would obviously be a speed trade off here.

The second opportunity is that of withdrawing the times from the STN. Rather than setting the latest possible action to the same time as its earliest possible time, it could be set to the next latest possible time in the network. A shorter plan could be found but this may take longer to find.

One idea to improve the efficiency of the system is to discard the need to send the invariant checking actions to FF without a reduction of expressiveness or soundness. The advantage of this would be two fold. Firstly, the search space would be smaller, and secondly, FF would need to do less work instantiating the actions (one observed problem with this planner). This would require more post processing as conditions and effects of the invariant would have to be moved to the start action and then removed afterwards.

Potential Further Work

As already observed, one potential change to the system is simply to change the classical planner. There are two obvious choices here. One is to use MetricFF, a variant of FF which would allow the use the resources and other numeric values in the domain, and so tackle Complex domains. The second is to use a partial order planner. This would of course get rid of the need to lift the partial order plan as it could be taken straight from the planner.

Currently, as all temporal information is taken out the problem for FF, it cannot know the cost of the actions it is using. It may well be the case that it is better to use a few short duration actions, rather than one long one. However, FF's heuristic will choose the longer one as it tries to minimise the total number of actions, not the total duration. By incorporating the durations into the heuristic, this could improve the quality of the plans.

References

Dechter R., Meiri J. & Pearl J. 1989. Temporal Constraint Networks. In *Proceedings from 83-93 Principles of Knowledge Representation and Reasoning*: 83-93. Toronto, Canada.

Edelkamp S. & Helmert M. 2000. On the Implementation of Mips. In *Proceedings from the Fourth Artificial Intelligence Planning and Scheduling (AIPS), Workshop on Decision-Theoretic Planning*. 18-25 Breckenridge, Colorado: AAAI-Press.

Fox M. & Long D. 2001. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains, Technical Report, Department of Computer Science, University of Durham.

Fox M. & Long D. 2002. The third International Planning Competition: Temporal and Metric Planning. In *Proceedings from the Sixth International Conference on Artificial Intelligence Planning and Scheduling*. 115-117

Gallo G. & Pallottion S. 1988. Shortest Path Algorithms. In *Annals of Operations Research* 13:38-64.

Gerevini A. & Serina I. 2002. LPG: a Planner based on Local Search for Planning Graphs. In *Proceedings of the Sixth Int. Conference on AI Planning and Scheduling (AIPS'02)*. AAAI Press.

Howey R. & Long D. 2003 VAL's Progress: The Automatic Validation Tool for PDDL2.1 used in The International Planning Competition. Forthcoming.

Long D. & Fox M. 2002. Fast Temporal Planning in a Graphplan Framework. In *Proceedings from the Sixth International Conference on Artificial Intelligence Planning and Scheduling*.

McDermott D. & the AIPS'98 Planning Competition Committee 1998. PDDL – The Planning Domain Definition Language. Technical Report, Department of Computer Science, Yale University.

McDermott D. 2000. The 1998 AI Planning Systems Competition. *AI Magazine* 2(2):35-55.

Moreno D., Oddi A., Borrajo D., Cesta A. & Meziat D. 2002. Integrating Hybrid Reasoners for Planning and Scheduling. In *Proceeding of the twenty-first workshop of the UK Planning and Scheduling Special Interest Group*, 179-189.

Weld, D. S. 1999. Recent Advances in AI Planning. *AI Magazine* 20(2).