Extending Temporal Planning for the Interval Class

Bharat Ranjan Kavuluri
bharatranjan@gmail.com
AIDB Lab, IIT Madras, Chennai
Tamilnadu, India 600036

Abstract

The paper presents a planner called Sharaabi. It is an extension to DRIPS, a planner which tackled required concurrency with overall preconditions requiring continuous support. The paper describes the motivation behind the work and the way the planner operates. The extensions to DRIPS are described in brief. Examples are provided to show that the planner works.

Introduction

The solutions of temporal planning problems more often than not require the concurrent execution of durative actions due to goal deadlines or interactions among actions. The latter case has been labeled as Coordination/Required Concurrency (RC)(Halsey, Long, and Fox 2004; Cushing et al. 2007) and the problems for which it happens have been categorized as temporal planning problems with coordination/required concurrency. The planner CRIKEY (Halsey, Long, and Fox 2004; Coles et al. 2009b) has been specifically designed to handle this class of problems by maintaining envelopes of possibly interacting actions. It searches in the space of split actions- where a durative action D is divided into two point actions D_{start} and D_{end}. The at-start effect and preconditions are added to D_{start} and the at end preconditions and effects are added to D_{end}. The invariants are maintained in a separate queue to check for violations. At each step, CRIKEY decides on actions to add to the plan, If it chooses a start action, it updates the state description with immediate effects and creates a new envelope. If it chooses an end action, it must choose a start action to pair it with and close an envelope. It uses a simple temporal network (Dechter, Meiri, and Pearl 1991) to check for the validity of the current plan under consideration after every such addition of an action. There have been several variants to CRIKEY, notably CRIKEYshe (Coles et al. 2009b), and CRIKEY3 (Coles et al. 2008) and COLIN (Coles et al. 2009a) (which maintains a LP problem to tackle continuous durative planning). POPF (Coles et al. 2010) is a forward state space temporal POCL planner. Instead of committing on the start times of actions as in SAPA, or committing and then lifting a partial order in CRIKEY, LPG.s etc., POPF maintains a partial order from the start and tries to exploit this order for solving RC problems.

The Invariant Class

All these planners except (Coles et al. 2010) do not solve a class of problems we labelled The Invariant Class. The notable feature of this class is that every problem has all its plans requiring atleast one action with an overall precondition that needs to be supported/covered by an overlap of actions providing/establishing the condition. A few motivating examples are shown in the figure 1. Figure 1(a) shows the type of problems which the planners described in the previous paragraph can solve- problems where each producer supports multiple consumers. The problems shown in figure 1(b) require at least one consumer to be supported by multiple producers. The reason the above planners can not tackle these problems is because they do not generate the necessary temporal constraints which force the producers to overlap. In the absence of those constraints, each check for schedulability either takes a significant amount of time and the planner gets stuck or the planner returns invalid plans/no plan messages.

DRIPS

I designed a planner called DRIPS (Kavuluri 2010) to tackle this class of problems. DRIPS is a modification of SAPA (Minh and Kambhampati 2003). SAPA is a metric-temporal planner which searches in the space of time stamped states. It checks for applicable actions whenever an event occurs. Events occur when an action is applied on a state or when delayed effects of actions are applied on a state from its event queue -the time is advanced to the time stamp of the event. SAPA was modified by adding two more types of states after the at start effects of an action are applied.

- A state which signifies the time just before the end of each action
- A state which indicates the passage of one clock tick without any action

This causes the planner to check for actions before the producers finish and also causes the planner to generate all possibilities at each clock tick. In the event the planner fails to

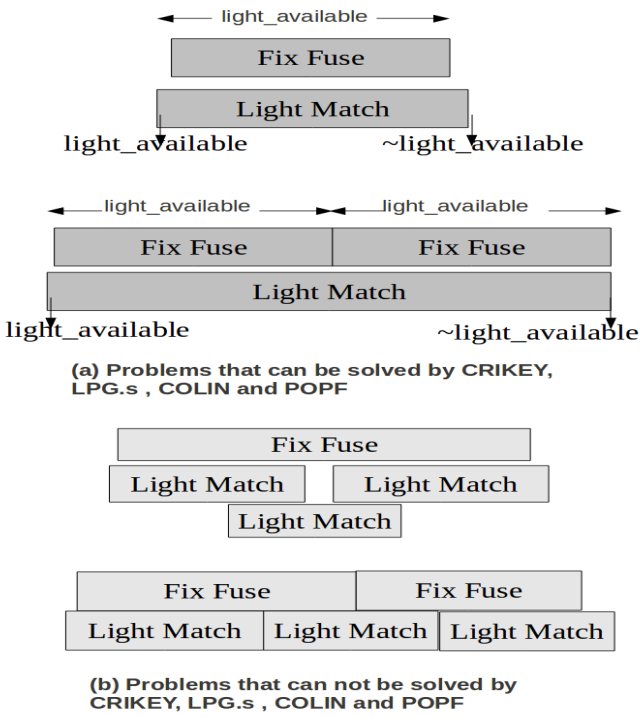

Figure 1: Required Concurrency and Invariant Class

find a plan using the first type of states, the clock tick based states ensure that completeness is not lost. So, given enough time and memory, if the given problem has a plan whose step size is not smaller than the clock tick, DRIPS theoretically should find a plan. More details about DRIPS are available in (Kavuluri 2010).

Sharaabi

The entry for this planning competition Sharaabi consists of modifications that have been made to DRIPS upto this point. The modifications are listed below.

- Sharaabi tackles derived predicates
- Sharaabi tackles problems requiring time-aware temporal planning

Both features of Sharaabi are described here with example problems.

Handling Derived Predicates

Derived predicates were implemented by implementing a class called Predicate formula and associating it with each derived predicate class. The derived predicates are evaluated when testing whether the action is applicable. Once the derived predicate is checked, its truth value is stored in the state so that it need not be computed again. The figures 2, 3 and 7 give an example match problem involving derived predicates where you need either a match or a bulb to be on to mend the fuse. The derived predicate bright becomes true when either of the derived predicates lightavail and bulbavail is true.

```
(define (problem fixfuse)
(:domain matchcellar)
(:objects
match1 - match
bulb - bulb
fuse1 fuse2 - fuse)
(:init
(unused match1)
(working bulb)
(handfree))
(:goal (and (mended fuse1))))
```

Figure 2: Derived Predicate Example- Problem

```
;; Search time 22 milisecs
;; State generated: 9 State explored: 4
0.0: (LIGHT_MATCH match1) [7.0]
0.0: (MEND_FUSE fuse1) [9.0]
6.99: (Put_on bulb) [4.0]
```

Figure 3: Plan for the derived match problem

```
(at start
(assign (convoy-tail-left-at ?v1 ?e) (+ t (/ (convoy-length ?c)
(convoy-speed ?c))))) - effect
(at start
(<= (convoy-tail-left-at ?v1 ?e) (+ (+ t (- 0 (+ (inner-
convoy-distance) (convoy-edge-travel-time ?e)))) (?dura-
tion)))) - precondition
```

Figure 4: Time-aware temporal planning example

```
(define (problem fixfuse)
(:domain TAPmatchcellar)
(:objects
match1 match2  match
fuse1 fuse2 - fuse)
(:init
(unused match1) (unused match2)
(= (cansolve) 7.0) (handfree) (q))
(:goal (and (mended fuse2)))))
```

Figure 5: Time-aware temporal planning example contd..

```
;; Search time 14 milisecs
;; State generated: 12 State explored: 8
0.0: (setTime) [1.0]
5.0: (LIGHT_MATCH match2) [15.0]
5.0: (MEND_FUSE fuse2 match2) [9.0]
```

Figure 6: Time-aware temporal planning example contd..

```
(define (domain matchcellar)
(:requirements :typing :durative-actions :derived-predicates)
(:types match fuse bulb)
(:predicates
(light ?match - match)
(bulblight ?bulb - bulb)
(switchon ?bulb - bulb)
(handfree)
(unused ?match - match)
(working ?bulb - bulb)
(mended ?fuse - fuse))
(:durative-action LIGHT_MATCH
:parameters (?match - match)
:duration (= ?duration 7)
:condition (and
(at start (unused ?match))
(over all (light ?match)))
:effect (and
(at start (not (unused ?match)))
(at start (light ?match))
(at end (not (light ?match)))))
(:durative-action Put_on
:parameters (?bulb - bulb)
:duration (= ?duration 4)
:condition (and
(at start (working ?bulb)))
:effect (and
(at start (not (working ?bulb)))
(at start (bulblight ?bulb))
(at start (switchon ?bulb))
(at end (not (switchon ?bulb)))
(at end (not (bulblight ?bulb)))))

(:durative-action MEND_FUSE
:parameters (?fuse - fuse)
:duration (= ?duration 9)
:condition (and
(at start (handfree))
(over all (bright)))
:effect (and
(at start (not (handfree)))
(at end (mended ?fuse))
(at end (handfree))))
(:derived (lightavail)
(and ((exists (?z) (light ?z)))))
(:derived (bulbavail)
(and ((exists (?z) (bulblight ?z)))))
(:derived (bright) (or ((lightavail) (bulbavail)))))
```

Figure 7: Example for Derived Predicates- Domain

Time-aware temporal planning

Time-aware temporal planning indicates that the current time and action durations are part of actions preconditions and effects. An example of the same is provided in figure 4. The first expression (effect) assigns the value of the current time plus the time the convoy takes to travel a distance equal to its length to the convoy-tail-left-at function. The precondition verifies if the convoy-tail-left-at function satisfies the inter-convoy distance to be maintained if the convoy were to start now.

This was implemented by using the $\#t$ and $?duration$ features in SAPA. This was done to add a function called time which returns the current time of the state when it is called. An example is shown in the figures 5, 6 and 8 to demonstrate this capability. The time when the planner can begin to solve the problem is set by the setTime action and this is verified in the preconditions of the action LIGHT_MATCH. The domain and problem files are given in figures 5 and 6. The plan is shown in Figure 8.

```
(define (domain TAPmatchcellar)
(:requirements :fluents :typing :durative-actions)
(:types match fuse)

(:predicates
(light ?m - match)
(handfree)
(unused ?m - match)
(mended ?f  fuse) (p) (q) )

(:functions (cansolve) )

(:durative-action setTime
:parameters ()
:duration (= ?duration 1)
:condition (and (at start (q)))
:effect (and (at start (assign cansolve 5.0))
(at end (p)) (at start (not (q)))) )

(:durative-action LIGHT_MATCH
:parameters (?m - match)
:duration (= ?duration 15)
:condition (and
(at start (p))
(at start ( == cansolve t))
(at start (unused ?m)) (over all (light ?m)))
:effect (and
(at start (not (unused ?m)))
(at start (light ?m))
(at end (not (light ?m)))))

(:durative-action MEND_FUSE
:parameters (?f - fuse ?m - match)
:duration (= ?duration 9)
:condition (and (at start (handfree))
(over all (light ?m)))
:effect (and (at start (not (handfree)))
(at end (mended ?f))
(at end (handfree)))))
```

Figure 8: Time-aware temporal planning example Domain

Conclusions and Future Work

This paper presents Sharaabi, an extension to DRIPS which deals with Derived predicates and Time-aware temporal

planning. The paper explains in brief how these features were implemented and demonstrates the planners capabilities with brief examples. Currently we are verifying the implementation for scalability and coverage. Future work would entail improving the efficiency of the planner like looking at different heuristic functions to improve speed.

References

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. Planning with Problems Requiring Temporal Coordination. In *AAAI-2008*.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2009a. Temporal Planning in Domains with Linear Processes. In *IJCAI 2009*.

Coles, A.; Fox, M.; Halsey, K.; Long, D.; and Smith, A. 2009b. Managing concurrency in temporal planning using planner-scheduler interaction. *AI Magazine*.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. . In *Twentieth International Conference on Automated Planning and Scheduling (ICAPS-10)*.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. 2007. When is temporal planning really temporal? In *IJCAI*.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal Constraint Networks. *Artificial Intelligence*.

Halsey, K.; Long, D.; and Fox, M. 2004. CRIKEYA planner looking at the integration of scheduling and planning. In *Proceedings of the Workshop on Integration of Scheduling Into Planning at 13th International Conference on Automated Planning and Scheduling (ICAPS '03)*.

Kavuluri, B. R. 2010. Required Concurrency without a Scheduler. In *28th Workshop of UK Planning and Scheduling Special Interest Group*.

Minh, D. B., and Kambhampati, S. 2003. SAPA: A Scalable Multi-Objective Metric Temporal Planner. *JAIR*.