Apollo 13: A Challenge Domain for the Planning Community

C Henrik Westerberg [*] and **John Levine**
Centre for Intelligent Systems and their Applications
School of Informatics, The University of Edinburgh
Appleton Tower, Crichton Street, Edinburgh EH8 9LE, UK
h.c.westerberg@sms.ed.ac.uk
johnl@inf.ed.ac.uk

Abstract

The problem of general purpose planning is vast in terms of variety of domain. PDDL currently represents a subset of all possible domains. One type of domain that may be interesting to include under PDDL are those domains in which actions examine or update more of the current state. That is, in order to update a state, a complicated examination and update occurs each time an action is simulated. Apollo 13 is a new domain in which the whole current state must be considered in order to determine the next state. This kind of domain raises interesting questions about how best to represent these domains and the affect they might have on the language. One solution to the representation problem is for PDDL to support a foreign function interface.

Introduction

The problem of general purpose planning is vast in terms of variety of domain. One definition of planning is:

> "The solution of a problem - any problem - consists in discovering how to transform an existing state of affairs in to a desired one that has not yet come into being (Donaldson 1978)"

Broadly speaking, a vast number of problems can be translated into this current state → desired state scheme. This in turn creates its own problem. How can so many domains be represented using a single language? Or more interestingly, what kinds of domains should we be trying to solve? Currently PDDL and the planning model constrain us to a particular kind of planning.

Most of the current and previous domains from the planning competitions involve movement, delivery, and various types of resource. However this is just one potential shard of AI Planning. It may be interesting to look at domains where actions cause great changes in state, or require a more detailed examination of current the state. This may give rise to a need for state updates to be carried out by complicated simulators, or external function calls to calculate state changes correctly and conveniently.

Space in particular has been a notable source for AI Planning domains. Apollo 13 is a domain based on the task of devising an emergency start up procedure for computer components with only a limited amount of current available. The domain is an example where calculating future states requires examination of the whole current state. This is difficult to do with the current implementation of PDDL (Fox & Long 2001). When representing Apollo 13 in PDDL it is difficult to generalise the domain specification. Though more complicated, this style of domain is meant for domain independent planners and is meant to be tractable with existing technology.

One way to overcome the representational issues is to leave the physical modelling of systems to simulation rather than re-interpreting the domain into a domain language. We advocate the role of using simulators with planning. This could be incorporated into PDDL by allowing a means to call other functions or executables. With this extension it will open a new class of domains for the planning community. However, this has important implications for the current planning model.

In summary we will look at the state of PDDL domains from the competitions. The idea of Apollo 13 will then be given its basis. The domain of Apollo 13 will be outlined, and examples will given about how it could be represented. Finally, we will close with some ramifications.

PDDL Planning Domains

PDDL has been in development for the last five years with the original scope for PDDL being quite broad. More recently its focus has been narrowed on to more "Classical" styles of domain. Current PDDL domains are written so that they have quite simple updates of state. Actions typically contain simple preconditions, meaning very little needs to be verified about the current state in order to derive the next state. Also state updates tend to change just a small amount of the current state.

Due to the language of PDDL and they way it forces you characterise particular domains, it makes most of the domains broadly equivalent. Goals are typically satisfied with sequences like: load → move → unload. With the inclusion of numbers and duration this has been complicated: load (10 mins, used 5 litres fuel) → move (1hr, 20 litres of fuel) → unload (10 mins, 5 litres fuel). For some domains there may

[*]I would like to thank Ruth Aylett,Derek Long and the reviewers for their helpful comments. I would also like to thank EPSRC for support via a quota studentship: award number 00317882

be extra preparation steps, or the sequences may need to be synched with other sequences elsewhere in the domain.

The current crop of PDDL domains are constrained by the modelling language but also by the planning model the language is meant to cover. Though PDDL makes no assumptions about the invertability of actions it is assumed when writing a new domain that all actions within the domain are invertible. They can be reasoned with by both going forwards and backwards from a particular state. Simulation will make this difficult or even impossible to maintain. Adding the simulation aspect will also make the plans more situated, the plans will be run in simulation. Having a more simulated basis for planning will also help with non-deterministic domains. The other aspects of the planning model are the same: deterministic actions, full visibility of state, and a state only changeable by the planning agent.

Apollo 13

Apollo 13 was NASA's 13th flight using Apollo flight hardware and was to be the 3rd Lunar landing. During the flight, the module suffered a massive explosion in one of the oxygen tanks. During the time from the explosion to the splashdown, NASA had to devise many emergency procedures to try and bring the crew safely home. One such procedure we are interested in is an emergency reboot of the module's core components, such that the module could splash down but also not consume too much electrical current. For the interested reader, there are many sources of information on Apollo 13's flight available online[1], in books, and one major film.

Basic Problem

The basic problem is to switch N components on. As each component is switched on the overall drain on the system increases. However the increase in drain, when a component is switched on, is dependent on which other components are on and off. The goal is to come up with an order of switching that minimises the drain on the system or is under some particular threshold.

An example

There are three components: X, Y, and Z. Each of the components must be switched on. How the components behave in relation to the other components can be given using truth tables:

Table 1: Current for Component X

Y	Z	Value
off	off	0.8
off	on	0.3
on	off	0.6
on	on	0.2

[1] http://nssdc.gsfc.nasa.gov/planetary/lunar/apollo13info.html

Table 2: Current for Component Y

X	Z	Value
off	off	0.9
off	on	0.5
on	off	0.5
on	on	0.3

Table 3: Current for Component Z

X	Y	Value
off	off	0.2
off	on	0.1
on	off	0.1
on	on	0.1

From the above example, if all the components were off, then switching on component Z would increase the drain by 0.2. The best sequence is ZXY $(0.2 + 0.3 + 0.3)$, resulting in an overall system drain of 0.8.

Extensions

There are a number of ways to extend the domain. One way would be to introduce dependencies between components. For example, in order to switch Z on, X and Y would also have to be switched on. In addition to this, dummy components can be added that may not have to be switched on to satisfy all the goals but be there to increase the interaction between components. More than one way in which an individual component can be switched on may exist, for example satisfying `(status X on)` may be achieved by turning on either X_1, X_2, or X_3. Additional constraints can be added in the form of safety constraints, X must be on before Y. A further extension would be to simulate the system over time, so that the total drain of the system does not exceed a value over some time period, also some components may need to be on or off at particular time stamps (a component could be switched off with an additional `switch_off` action). The simulation of the system can be made more realistic by determining the real current values based on temperature, and humidity.

Representing Apollo 13 in PDDL

PDDL $-$ Current

The domain can be represented using the current version of PDDL though somewhat inconveniently. One possible representation is:

```
(define (domain Apollo-13)
(:requirements  :typing :fluents)
(:types component bool - object)
(:predicates
   (status ?y - component ?v - bool)
   (table ?x ?y - component ?v - bool
          ?z - component ?w - bool))
(:functions (current))
(:action switch_on
  :parameters (?x ?y ?z - component
       ?a ?b - bool)
  :precondition
     (and (status ?x off) (status ?y ?a)
  (status ?z ?b))
  :effect
     (and (status ?x on)
          (not (status ?x off))
     (increase  (current)
         (table ?x ?y ?a ?z ?b))
)))
```

The three component example outlined above can be represented as:

```
(define (problem Apollo-13-1)
(:domain Apollo-13)
(:objects X Y Z - component)
(:init  (status X off )
(status Y off )
(status Z off )
(= (table X  Y off Z  off) 0.8)
(= (table X  Y off Z  on)  0.3)
(= (table X  Y on  Z  off) 0.6)
(= (table X  Y on  Z  on)  0.2)
(= (table Y  X off Z  off) 0.9)
(= (table Y  X off Z  on)  0.5)
(= (table Y  X on  Z  off) 0.5)
(= (table Y  X on  Z  on)  0.3)
(= (table Z  X off Y  off) 0.2)
(= (table Z  X off Y  on)  0.1)
(= (table Z  X on  Y  off) 0.1)
(= (table Z  X on  Y  on)  0.1)
(= (current) 0))
(:goal (and (status X on )
    (status Y on )
    (status Z on )))
(:metric minimize (current)))
```

This representation suffers from the problem this it does not generalise over the number of components in the system. A new domain must be written for each different number of components. Thought it may be possible to write a domain file that is usable up to a maximum number of components.

PDDL – Function Call

An alternative would be to extend the PDDL language so that a call to an outside function would update the state. This has been used in a couple of cases before: Chemical Flow (Aylett *et al.* 1998), and within O-Plan (Currie & Tate 1991). The domain file could look like:

```
(define (domain Apollo-13)
(:requirements  :typing :fluents)
(:types component bool - object)
(:predicates
   (status ?y - component ?v - bool))
(:functions (current)
    (sim_current))
(:action switch_on
  :parameters (?x)
  :precondition (and (status ?x off))
  :effect (and (status ?x on)
          (increase  (current) (
             call (sim_current))
))))
```

The function, `sim_current`, would have to examine the current state to determine the correct increase in current and return the value to the planner. In this case the same problem file can be used.

This has the advantage of generalising over all possible number of components. It has the negative effect of making the domain less portable, and hides some of the semantics of the domain from the planner. But if some kind of “foreign function interface” was added to the language this would allow various kinds of executables/code segments to be callable from domain files.

Ramifications

We would like to explore a new class of planning domain that require a complex examination of the current state to determine the future state. One example of this is the Apollo 13 domain. Supporting this kind of domain will allow more complicated and more realistic planning domains to be tackled by domain independent planners.

One solution to the representation problem of these domains would be to add a foreign function interface. This would allow a planner to make calls to outside systems to calculate future states. This would ease the strain on the modelling language as it would no longer have to represent every aspect of the physical system. Thus also bringing in a broader range of domains to plan for, and to make them more convenient to represent in PDDL.

Adding simulation to the current planning model would have serious implications for it. For instance how is it possible to reason backwards in such a situation? If all the information is no longer specified in the action header it is impossible to determine prior states. It would also make the planner more situated, as at least the plans are being interpreted by a simulator. It may also help with extending planning to non-deterministic domains.

References

Aylett, R.; Soutter, J. K.; Petley, G. J.; and Chung, P. W. H. 1998. AI planning in a chemical plant domain. In *Proceedings of 4th World Congress on Expert Systems*, 98–105. Mexico City: Cognizant Communication Corporation.

Currie, K., and Tate, A. 1991. O-plan: the open planning architecture. *Artifical Intelligence* 52:49–86. Availible at: http://www.aiai.ed.ac.uk/~oplan/.

Donaldson, M. 1978. *Children’s Minds*. Glasgow: Fontana/Collins.

Fox, M., and Long, D. 2001. PDDL2.1: An extension to PDDL for expressing temporal planning domains. Availible at: http://www.dur.ac.uk/d.p.long/competition.html.