Planning with Ill-defined Resources

Laurence A. Kramer

The Robotics Institute
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
lkramer@cs.cmu.edu

Abstract

Many real world planning problems involve resources that may not be completely specified at the time of plan generation and that may be utilized opportunistically during plan execution. What is needed in addressing such problems is the ability to reason about hypothetical resources and to generate bounds for other resources to ensure feasible choices during plan execution. We present a slight twist to the well known "travel domain" and a solution model to illustrate this point.

Introduction

Perhaps second only to the (in)famous "blocks world" problem as a demonstration domain for planning engines is the "travel domain." The travel domain has taken many incarnations, but in essence involves an agent that needs to get from point A to point B, and must choose from several means of transport to do so. We investigate the scenario where the planning agent would like to travel across town, and may do so by either walking, taking a taxi, taking a bus, or some combination of these modes of transportation. This problem is handled in a straightforward manner by any number of planners, but becomes somewhat more difficult by introducing limited resources such as money and walking distance. We consider the further complication -- quite common in the real world -- that at plan generation time the location of taxis is unknown. The plan(s) we generate should give the agent foreknowledge, for instance, that if they start walking and hail a cab too early, they may end up having to walk again when their money runs out. The generated plan, then, gives the agent the tools to be able to operate opportunistically in response to changing real world conditions.

A Simple Travel Domain

(Nau et al. 1999) describe a simple transportation planning domain to demonstrate the workings of the SHOP planner. In this domain the planning agent is downtown and would like to either travel to the park or the suburb. The initial state of the world is specified in terms of how much money the agent has, whether the weather is bad or good, and the locations of bus routes and taxi stands. If the weather is good the agent can travel 3 miles, but if it is bad the agent can travel 0.5 mile. The bus costs a flat rate of $1.00, and a taxi costs $1.00 per mile plus a constant $1.50. Figure 1 presents the full SHOP model for this domain and Figure 2 an example initial state of the world. SHOP syntax for axioms, methods and operators is explained fully in (Nau et al. 2000). Briefly, the axioms are represented as Horn clauses, the methods as tasks, preconditions, and task lists, and the operators as tasks, and add and delete lists.

```
A1 (:- (have-taxi-fare ?distance)
       ((have-cash ?m)
        (eval (>= ?m (+ 1.5 ?distance)))))
A2 (:- (walking-distance ?u ?v)
       ((weather-is 'good)
        (distance ?u ?v ?w)
        (eval (<= ?w 3)))
       ((distance ?u ?v ?w)
        (eval (<= ?w 0.5))))
M1 (:method (pay-driver ?fare)
       ((have-cash ?m)
        (eval (>= ?m ?fare)))
       `((!set-cash ?m ,(- ?m ?fare))))
M2 (:method (travel-to ?q)
       ((at ?p) (walking-distance ?p ?q))
       '((!walk ?p ?q)))
M3 (:method (travel-to ?y)
       (:first (at ?x)
        (at-taxi-stand ?t ?x)
        (distance ?x ?y ?d)
        (have-taxi-fare ?d))
      `((!hail ?t ?x) (!ride ?t ?x ?y)
        (pay-driver ,(+ 1.50 ?d)))
       ((at ?x) (bus-route ?bus ?x ?y))
       '((!wait-for ?bus ?x)
         (pay-driver 1.00)
         (!ride ?bus ?x ?y)))
O1 (:operator (!hail ?vehicle ?location)
       ()
       ((at ?vehicle ?location)))
O2 (:operator (!wait-for ?bus ?location)
       ()
       ((at ?bus ?location)))
O3 (:operator (!ride ?vehicle ?a ?b)
       ((at ?a) (at ?vehicle ?a))
       ((at ?b) (at ?vehicle ?b)))
O4 (:operator (!set-cash ?old ?new)
       ((have-cash ?old))
       ((have-cash ?new)))
O5 (:operator (!walk ?here ?there)
       ((at ?here))
       ((at ?there)))
```

Figure 1: SHOP Simple Transportation-planning Domain

```
((distance downtown park 2)
 (distance downtown uptown 8)
 (distance downtown suburb 12)
 (at-taxi-stand taxi1 downtown)
 (at-taxi-stand taxi2 downtown)
 (bus-route bus1 downtown park)
 (bus-route bus2 downtown uptown)
 (bus-route bus3 downtown suburb)
 (at downtown)
 (weather-is 'good)
 (have-cash 12))
```

Figure 2: Example world state for Transportation-planning

By varying the initial conditions it is seen that SHOP is able to generate a number of feasible plans to travel to the park, uptown or the suburb. Given the ability to embed Lisp code in the axioms and methods of the model, SHOP is able to "reason" about limited resources such as money and the ability to walk certain distances.

Travel in the Real World

The SHOP simple travel domain model was never intended to demonstrate travel planning in the real world, but it is interesting to prod the model to see what it would take to make it more lifelike. First of all, it is quite common in the real world that you may not have enough money to take a taxi all the way across town, but you could take a taxi part way and take the bus the remainder. Or, you could plan to walk part of the way, and then catch a taxi. The SHOP model in Table 1 is not powerful enough to solve this problem.

Looking at methods **M2** and **M3** in Figure 1, the assumption is that the planning agent is always at one location trying to get directly to another location. In order to solve this problem we add **M4** (see Figure 3) which states that to get from location1 to location3, travel to location2 and then to location3.

```
M4 (:method (travel-to ?loc3)
       ((at ?loc1)
        (distance ?loc1 ?loc2 ?dist1)
        (distance ?loc2 ?loc3 ?dist2))
       '((travel-to ?loc2) (travel-to ?loc3)))
```

Figure 3: A method for traveling indirectly

With the addition of **M4**, SHOP is able to solve the problem in Figure 4, where the goal is (travel-to park).

```
((distance downtown park 2)
 (distance uptown downtown 8)
 (distance downtown suburb 12)
 (at-taxi-stand taxi1 uptown)
 (at-taxi-stand taxi2 downtown)
 (bus-route bus1 downtown park)
 (bus-route bus2 uptown downtown)
 (bus-route bus3 downtown suburb)
 (at uptown)
 (weather-is 'good)
 (have-cash 11)))
```

Figure 4: A travel domain that requires two modes of transportation.

The following two plans are generated:

```
((!HAIL TAXI1 UPTOWN)
 (!RIDE TAXI1 UPTOWN DOWNTOWN)
 (!SET-CASH 11 1.5)
 (!WAIT-FOR BUS1 DOWNTOWN)
 (!SET-CASH 1.5 0.5)
 (!RIDE BUS1 DOWNTOWN PARK))

((!HAIL TAXI1 UPTOWN)
 (!RIDE TAXI1 UPTOWN DOWNTOWN)
 (!SET-CASH 11 1.5)
 (!WALK DOWNTOWN PARK))
```

Real World Taxis are Different than Buses

To represent many real world cities the model we have built is still too contrived. Taxi stands are few and far between or nonexistent. The model of waiting at a taxi stand for a taxi as you would wait at a bus stop for a bus, is not the one we want. Somehow we'd like to represent the idea that just as the distances one can walk don't fall into a set of discrete bins, it might be possible to flag a cab just about anywhere and get off anywhere.

Given this insight, we propose a planning domain that is initially simpler than the one we've been considering, but which is more true to life. In this domain we are uptown and would like to get to the park. We have some money, but not enough to take a taxi all the way. Depending on the weather, we are able to walk a certain distance, but not all the way from uptown to the park. We would like to generate a feasible plan to get to the park if one exists. The world state for this problem is represented in Figure 5, and the goal is (travel-to park).

```
((distance uptown park 10)
 (at uptown)
 (weather-is 'good)
 (good-weather-distance 3)
 (bad-weather-distance 0.5)
 (have-cash 10))
```

Figure 5: A simplified, but lifelike planning domain.

The "Hail a Cab" Domain

It is not difficult to see that unless the amount of money we have is just enough to take a taxi to the limit of the distance we're able to walk, there is either no solution to the problem or an infinite number of solutions, since the distance we can walk is a continuous resource. In reality, taxi meters only turn over at regular intervals, so we could enumerate solutions corresponding to those discrete intervals, however the number of possible solutions could still be huge.

Another way to look at this problem, though, is to design a model such that we plan for the boundary solutions, and at plan execution time we can be assured that any of the myriad solutions between these bounds will be feasible ones.

The “Hail a Cab” Model

Corresponding to our intuition, then, we would like to build into our model the ability to generate two hypothetical waypoints *during the planning process*, call them waypoint1 and waypoint2. Given the simplicity of our starting world state, we expect that the planner will either fail to find a plan, or will generate exactly two plans: one at the limit of our money to waypoint1 and one at the limit of our walking distance to waypoint2.

After some trial and error we were able to represent such a model in SHOP. We reuse the model (Figure 1) from the simple travel domain (with a few improvements to make it more maintainable) and add an "assert" operator **O6** as described in (Nau et al. 2000). The assert operator has no delete-list, and adds all of it’s arguments to the current world state. During planning we assert the existence of waypoints, and their distance from an origin and destination.

Two new planning methods **M5**, **M6** are added to represent the “take the taxi to the extreme” and the “walk to the extreme” actions. In addition, we found it necessary to add an axiom **A5** which filtered waypoints from non waypoints. Before addition of this axiom, the planner would thrash while attempting to travel between the two hypothetical waypoints or add an ever increasing number of new waypoints between the original ones. The full model for the "hail a cab" domain is listed in figure 6.

```
                ;;****** AXIOMS ******

;; Fetch the max walking distance from the
;; database depending on whether the weather is ;;
bad or good.
A1 (:- (possible-walking-distance ?wd)
       ((weather-is 'good)
        (good-weather-distance ?wd))
       ((bad-weather-distance ?wd)))

;; Two points are within walking distance if
;; distance is less than possible walking
;; distance.
A2 (:- (within-walking-distance ?u ?v)
       ((distance ?u ?v ?w)
        (possible-walking-distance ?wd)
        (eval (<= ?w ?wd))))

;; A taxi costs $1.50 + $1.00/mile
A3 (:- (have-taxi-fare ?distance)
       ((have-cash ?m)
        (eval (>= ?m (+ 1.5 ?distance)))))

;; It's worth considering a taxi only if you have
;; at least the $1.50 fixed rate, plus another
;; $.50 to go a half mile (we can assume the
;; taxi's meter clicks over every half mile).
A4 (:- (worth-considering-taxi ?m)
       ((have-cash ?m) (eval (>= ?m 2.0))))

;; This helper axiom is needed to restrict the
;; search from attempting to travel from one
;; waypoint to another.
A5 (:- (not-waypoint ?w)
       ((not (is-waypoint ?w))))
```

```
                ;;****** METHODS  ******

;; Paying the driver reduces your cash on hand by
;; the fare.
M1 (:method (pay-driver ?fare)
            ((have-cash ?m) (eval (>= ?m ?fare)))
            '((!set-cash ?m ,(- ?m ?fare))))

;; Attempt to walk.
M2 (:method (travel-to ?q)
            ((at ?p)
             (within-walking-distance ?p ?q))
            '((!walk ?p ?q)))

;; Attempt to take a taxi.
M3 (:method (travel-to ?y)
            ((at ?x)
             (distance ?x ?y ?d)
             (have-taxi-fare ?d))
            '((!hail taxi ?x) (!ride taxi ?x ?y)
              (pay-driver ,(+ 1.50 ?d))))

;; Taking a taxi "script."
;; Only to be used when it is certain that
;; preconditions for taking a taxi are satisfied.
M4 (:method (take-taxi ?x ?y ?m)
            ()
            '((!hail taxi ?x) (!ride taxi ?x ?y)
              (pay-driver ,?m)))

;; Attempt to combine taking one form of
;; transportation with another by hypothesizing a
;; waypoint, waypoint1. This waypoint represents
;; spending all your money on a taxi, and then
;; traveling the rest of the way by some other
;; means.
M5 (:method (travel-to ?z)
            ((not-waypoint ?z)
             (worth-considering-taxi ?m)
             (at ?x) (not-waypoint ?x)
             (distance ?x ?z ?dist))
            '((!assert
               ((is-waypoint waypoint1)
                (distance ?x waypoint1
                          ,(- ?m 1.50))
                (distance waypoint1 ?z
                          ,(- ?dist ?m -1.50))))
              (take-taxi ?x waypoint1 ?m)
              (travel-to ?z)))

;; Attempt to combine taking one form of
;; transportation with another by hypothesizing a
;; waypoint, waypoint2. This waypoint represents
;; traveling as far as you can walk, and then
;; taking the taxi (or another means) from that ;;
point.
M6 (:method (travel-to ?z)
            ((not-waypoint ?z)
             (at ?x) (not-waypoint ?x)
             (distance ?x ?z ?dist)
             (possible-walking-distance ?wd))
            '((!assert
               ((is-waypoint waypoint2)
                (distance ?x waypoint2 ,?wd)
                (distance waypoint2 ?z
                          ,(- ?dist ?wd))))
              (!walk ?x waypoint2)
              (travel-to ?z)))

            ;;****** OPERATORS  ******

;; Operators O1-O5 are the same as in Figure 1
O6 (:operator (!assert ?g)
              ()
              ?g
              0)))  ;zero-cost to apply
```

Figure 6: “Hail a Cab” Model

Given the start state in Figure 5, and the goal of (travel-to park) SHOP was able to find the following two plans. (Execution time was 0.010 cpu seconds on a Sun Ultra 1 running Allegro Common Lisp 5.0.1).

```
((!ASSERT ((IS-WAYPOINT WAYPOINT2)
           (DISTANCE UPTOWN WAYPOINT2 3)
           (DISTANCE WAYPOINT2 PARK 7)))
 (!WALK UPTOWN WAYPOINT2)
 (!HAIL TAXI WAYPOINT2)
 (!RIDE TAXI WAYPOINT2 PARK)
 (!SET-CASH 10 1.5))

((!ASSERT ((IS-WAYPOINT WAYPOINT1)
           (DISTANCE UPTOWN WAYPOINT1 8.5)
           (DISTANCE WAYPOINT1 PARK 1.5)))
 (!HAIL TAXI UPTOWN)
 (!RIDE TAXI UPTOWN WAYPOINT1)
 (!SET-CASH 10 0)
 (!WALK WAYPOINT1 PARK))
```

From this result we can see that any plan where we walk at least 1.5 miles and at most 3 miles will be a feasible one. Having access to the plan in terms of two boundary solutions allows an intelligent agent to act opportunistically when executing the plan (hailing a cab at the most opportune time). Alternatively, these plan bounds could be handed off to a scheduler that may better deal with the optimization problem of trading off conserving money, conserving walking power, and other factors.

Critique of this Approach

Although we were successful in modeling the "Hail a Cab" problem and generating plans for it, it is far from being a satisfactory solution. First of all, the model is very specifically crafted to solve this particular problem. Although the model still performs relatively well when we add a number of the locations and modes of travel that were present in the original model, we found that sometimes the plans produced were very counter-intuitive, for instance involving travel to "intermediate" waypoints that were actually further than the overall distance to the goal location, thus resulting in negative distances. Although negative distances can be interpreted as a less than optimal plan, where some physical "backtracking" is taking place, it is probably preferred that these plans not be generated.

Adding axioms and additional preconditions to some methods succeeded in avoiding these "negative distance" solutions, however it took quite a level of effort to produce a model that generated all feasible plans without getting stuck in an endless loop.

This can be taken as a criticism of the modeling paradigm in SHOP, or it can be ascribed to the novice status of this author in working with the SHOP modeling language.

In any case, it seems that a more desirable approach to this problem would allow us to deal with it at a higher level of abstraction. For instance, it would be nice to be able to just specify that a taxi is a type of resource that behaves in a certain way, and have the planner be able to reason with the attributes of that resource type. This is the sort of capability that SIPE advertises (Wilkins 88), and it would be interesting to see a SIPE model for the "Hail a Cab" domain. In addition, it would be nice to model “waypoints” as a generic type of unfixed location resource, as opposed to hard-coding two waypoints in the domain model.

Future Work: Extensions to the Domain

The “hail a cab domain” attempts to capture some of the complexity and uncertainty of catching a cab in a real world city. There are a number of dimensions along which it could be extended to make it even more lifelike and challenging. A few that come to mind are:

Time Limits. Quite often it’s important to get across town before some deadline. Add in timing considerations to the model.

Restrictions on Origins and Destinations. It’s not possible to take a cab *anywhere*, and in most cities there are places you wouldn’t want to be dropped off by a cab, just for safety’s sake. Similarly, there are locations where it is not likely to catch a cab. Factor location constraints into the model.

Replanning. Often the weather can quickly change from good to bad. Does this necessitate replanning from scratch, or is it possible to build some look-ahead into the model?

Constraint Relaxation. In the real world we’d never stop walking in bad weather one block away from where we saw a cab, simply because we’d reached our “bad weather walking limit.” What’s a good way to relax this and other constraints?

Benchmark Problems for Real World Planning

In (Wilkins and desJardins 2001) the authors claim that benchmark problems for planning either ignore or filter out many of the qualities that typify real world planning problems, thus causing research to tend in nonproductive directions. We agree that this is a danger, but feel that a place remains for benchmark problems which test the adequacy of a planner to handle real world problems, even if these tests make some simplifying assumptions along one or more dimensions.

We present the "Hail a Cab" problem as a challenge problem for other planners to tackle. We propose the following additional metrics (besides execution time) with which to evaluate solutions to the problem:

Expressivity. Does the model express the problem in a full and intuitive way?

Extensibility. Is the model easily extensible to new real world requirements or along new dimensions?

Maintainability. How hard is the model to develop, debug and improve over time?

Scalability. How does the model scale in performance as the problem grows in the number of variables? Are there natural ways to apply heuristics to mitigate scalability

problems? How long does it take for the planner to fail to find a plan when there is none?

These metrics, except for possibly the last, are hard to quantify, but evaluating them for benchmark problems for real world planning can drive research in interesting and productive directions.

Conclusions

Investigating the "Hail a Cab" model suggests solution approaches for planning and scheduling in real world transportation domains where differing modes of transportation may be employed to achieve the goal of getting from an origin to a destination. Features of this model include the generation of hypothetical waypoints during plan generation, and reasoning about time bounds for non homogeneous actions. Providing these bounds can help ensure execution of a feasible plan. We feel that this model can serve as a useful test bed for real world planners.

Acknowledgements

We thank David Hildum for insights into a successful solution to this problem, and Dana Nau and his collaborators for making the SHOP planner freely available to others in the community.

References

Nau, D.; Cao, Y.; Lotem, A.; and Munoz-Avila, H. 1999. SHOP: Simple Hierarchical Ordered Planner, Technical Report, CS-TR-3981, UMIACS-TR-9904, Department Of Computer Science, and Institute for Systems Research, University of Maryland.

Nau, D.; Cao, Y.; Lotem, A.; and Munoz-Avila, H. 2000. SHOP and M-SHOP: Planning with Ordered Task Decomposition, Technical Report, CS-TR-4157, Department Of Computer Science, and Institute for Systems Research, University of Maryland.

Wilkins, D. 1988. *Practical Planning: Extending the Classical AI Paradigm*. San Mateo, CA: Morgan Kaufmann Publishers.

Wilkins, D. and desJardins, M. 2001. A Call for Knowledge-Based Planning. AI Magazine 22(1):99-115.