

Spencer Shimko

CMSC 471

HW 6

### 1. Planning Representations (25 pts.)

Our agent is hungry and unhappy, and wants to go on a vacation, sunbathe, and eat well. Our agent starts out rich, but only has limited options, some of which will consume a substantial part of its wealth.

The static predicates for the domain are:

- have(car)                    (The agent has a car)
- distance(x,y,d)            (The distance from x to y is d;  
distance is assumed to be reflexive; i.e.,  
distance(x,y,d)  $\Leftrightarrow$  distance(y,x,d))
- temp(x,t)                    (The temperature at location x  
is t)
- in(r,x)                      (Restaurant r is in city x)
- beach(x)                     (There is a beach in location x)
- fastfood(r)                 (r is a fast-food restaurant)
- swanky(r)                    (r is a swanky restaurant)

The dynamic predicates for the domain are:

- at(x,s)                      (The agent is at x in situation  
s)
- happy(s)                     (The agent is happy in situation  
s)
- hungry(s)                    (The agent is hungry in  
situation s)
- rich(s)                      (The agent is rich in situation  
s)

(Note that the situation variables will only be used in part (a) of this problem.)

The constants in the domain are:

-Balt	(Baltimore)
-Berm	(Bermuda)
-OC	(Ocean City)
-McDs	(McDonald's)
-BK	(Burger King)
-CTC	(Chez Tres Cher)
-MC	(Molto Caro).

The initial state is:

at(Balt, s0) ^ ~happy(s0) ^ hungry(s0) ^ rich(s0).

The static predicates in the KB are:

- distance (Balt,OC,150)
- distance(Balt,Berm,500)
- in(McDs,Berm)
- in(BK,OC)
- in(CTC,Berm)
- in(MC,OC)
- fastfood(McDs)
- fastfood(BK)
- swanky(CTC)
- swanky(MC)
- have(car)
- beach(OC)
- beach(Berm)
- temp(Balt,50)
- temp(OC,60)
- temp(Berm,80)

The actions in the domain are as follows:

-drive(x,y) : The agent can drive from location x to location y if it has a car and the distance from x to y is less than or equal to 200.

- fly(x,y) : The agent can fly from location x to location y if it is rich and the distance from x to y is greater than 200. Flights are expensive, so after flying, the agent is no longer rich.
- sunbathe(x) The agent can sunbathe if it is in a location (x) with a beach. The agent will end up happy after sunbathing if the temperature is above 75, but unhappy if the temperature is below 75.
- eat(r): If the agent is hungry [i.e., hunger is a precondition], it can eat at a restaurant r that is In the same location as the agent. The agent will end up not hungry, and will be happy if the restaurant is a swanky restaurant. However, it can only eat at a swanky restaurant if it is rich, and at the end of the action, will not be rich any more. (Fast food restaurants don't require richness, and don't change the status of the richness predicate.)

(a) (7 pts.) Describe *only* the Happy predicate using the situation calculus. You should have one or more possibility axioms (one for each relevant action) and one or more successor-state axioms (one for each relevant action). These axioms should characterize how the state of the Happy predicate changes as a result of the actions in the domain, in terms of the domain predicates listed above.

(You may want to refer to pages 330-333 in the book.)

**Initial State:**

at(Balt, s0) ^ ~happy(s0) ^ hungry(s0) ^ rich(s0).

**Possibility Axioms:**

hungry(s0) ^ At(Balt, s0) ^ rich(s0) => Poss(drive(OC) ^ eat(MC))

At(Balt, s0) ^ rich(s0) => Poss(fly(Balt, Berm) ^ sunbathe (Berm))

At(Berm,s0) ^ rich(s0) => Poss(eat(CTC)) (note this situation results from an impossible start state)

happy(s0) => happy(s1)

**Successor-State Axioms:**

Poss(a,s) => happy(s1) <=> a=(fly(Balt, Berm) ^ sunbathe (Berm)) OR (drive(Balt, OC) ^ eat(MC)) OR (happy(s0) )

**(b) (7 pts.) Describe the actions in this domain as STRIPS operators. Be sure to include all preconditions, add lists, and delete lists. (See pages 377-378.)**

Action(drive(x,y)),

Precond: At(x,s0) ^ have(car)

Effects:

Addlist: At(y,s1)

DeleteList: At(x,s0)

Action(fly(x,y)),

Precond: At(x,s0) ^ rich(s0)

Effect:

Addlist: At(y,s1)

DeleteList: rich(y) ^ At(x,s0)

Action(sunbathe(x)),

Precond: beach(x) ^ At(x,s0)

Effect:

Addlist: happy(s1)

DeleteList: At(x,s0)

Constr: x=Berm

Action(sunbathe(x)),

Precond: beach(x) ^ At(x,s0)

Effect:

Addlist: ~happy(s1)

DeleteList:

Constr: x=OC

Action(eat(r)),

Precond: hungry(x) ^ At(x, s0) ^ rich(s0)

Effect:

AddList: happy(x)

DeleteList: hungry(x)

Constr: r=swanky(r)

Action(eat(r)),

Precond: hungry(x) ^ At(x, s0)

Effect:

AddList: happy(x)

DeleteList: hungry(x)

Constr: r=fastfood(r)

**(c) (5 pts.) Show two different legal plans (sequences of actions) for achieving the goal described above from the given initial state.**

Rich(s0) ^ fly(Balt, Berm) ^ sunbathe(Berm) ^ eat(McD)

Rich(s0) ^ drive(Balt, OC) ^ eat(MC) ^ sunbathe(OC)

Rich(s0) ^ drive(Balt, OC) ^ eat(BK) ^ drive(OC, Balt) ^ fly(Balt, Berm) ^ subathe(Berm)

**(d) (6 pts.) How many legal plans are there for this goal? Explain your answer. Does the answer change depending on whether or not repeated states are allowed?**

Note: although it wasn't explicitly stated in the problem, the results here assume happy() is a goal. It was alluded to by the initial statement that our agent is unhappy and hungry, but never stated.

There are three legal plans.

Explanation: There are three sets of legal actions that don't repeat states and satisfy the goals of alleviating hunger and being happy. There are no other methods which satisfy the constraints placed on the problem by the predicates and actions. The third plan may seem strange, but the predicates allow the agent to drive to OC, eat, and return to Baltimore without repeating states. This is because you are initially hungry in Baltimore but you are not hungry when you return.

If you are allowed to repeat states there is an infinite number of legal plans. You could always perform certain actions which are free *and* meet the given preconditions an infinite number of times. For example, one could drive from Baltimore to OC [drive(Balt, OC)] repeatedly before finally sunbathing and eating. Or perhaps one could drive or fly somewhere, then sunbathe repeatedly, and finally eat.

## **2. Partial-Order Planning (25 pts.)**

Suppose that the agent starts building a partial-order plan to achieve the goal in the domain from problem #2. The agent decides to drive to Ocean City and eat at Burger King. Draw the partial-order plan at this point in the planning process. You do not need to show the dependencies associated with static predicates. Show all dependencies (ordering links and causal links) associated with dynamic predicates. Ordering links should be drawn as a thin, single arrow; causal links, as a double or thick arrow.

Now suppose that the agent decides to satisfy its Happiness goal using the Sunbathe action. Insert this action into the plan, showing all dependencies. Will this plan succeed? If so, complete the partial plan, resolving any threats that arise. If not, complete the partial plan to the point where planning fails, and explain the source of the failure.

The partial order drawing is attached.

The plan will succeed since the dynamic and static predicates allow driving back to Baltimore before flying to Bermuda for sunbathing and happiness. This is reflected in the attached POP.

### 3. Decision tree learning (25 pts.)

The following table gives a data set for deciding whether to play or cancel a ball game, depending on the weather conditions.

Outlook	Temp (F)	Humidity (%)	Windy?	Class
sunny	75	70	true	Play
sunny	80	90	true	Don't Play
sunny	85	85	false	Don't Play
sunny	72	95	false	Don't Play
sunny	69	70	false	Play
overcast	72	90	true	Play
overcast	83	78	false	Play
overcast	64	65	true	Play
overcast	81	75	false	Play
rain	71	80	true	Don't Play
rain	65	70	true	Don't Play
rain	75	80	false	Play
rain	68	80	false	Play
rain	70	96	false	Play

(a) (10 pts.) At the root node for a decision tree in this domain, what are the information gains associated with the Outlook and Humidity attributes? (Use a threshold of 75 for humidity (i.e., assume a binary split: humidity  $\leq$  75 / humidity  $>$  75.)

Note: "lg2()" in these solutions is base 2.

Overall: Plays = 9; Not Play = 5

$$I(P) = -(9/14 \lg_2 (9/14) + 5/14 \lg_2 (5/14) ) = 0.94$$

Outlook: Sunny = (2 = play, 3 = not play);

Overcast = ( 4 = play);

Rain = ( 3 = play, 2 = not play)

$$I(P, \text{Outlook}) = -5/14(2/5 \lg_2 (2/5) + 3/5 \lg_2 (3/5) ) +$$

$$-4/14(4/4 \lg_2 (4/4) + 0/4 \lg_2 (0/4) ) +$$

$$-5/14(3/5 \lg_2 (3/5) + 2/5 (\lg_2 2/5) ) = 0.69$$

Humidity:

$$I(P, \text{Humidity}) = -9/14(5/9 \lg_2 (5/9) + 4/9 \lg_2 (4/9)) +$$

$$-5/14(4/5 \lg_2 (4/5) + 1/5 \lg_2 (4/5)) = 0.9$$

Gain:

$$IG(P, \text{Outlook}) = I(P) - I(P, \text{Outlook}) = .25$$

$$IG(P, \text{Humidity}) = I(P) - I(P, \text{Humidity}) = 0.04$$

**(b) (10 pts.) Again at the root node, what are the gain ratios associated with the Outlook and Humidity attributes (using the same threshold as in (a))?**

$$\text{SplitInfo}(P, \text{Outlook}) = -(5/14 \lg_2 (5/14) + 4/14 \lg_2 (4/14) + 5/14 \lg_2 (5/14) ) = 1.58$$

$$\text{SplitInfo}(P, \text{Humidity}) = -(9/14 \lg_2 (9/14) + 5/14 \lg_2 (5/14) ) = 0.65$$

$$\text{GainRatio}(T, \text{Outlook}) = 0.25 / 1.58 = .158$$

$$\text{GainRatio}(T, \text{Humidity}) = 0.4 / 0.65 = .615$$

**(c) (5 pts.) Suppose you build a decision tree that splits on the Outlook attribute at the root node. How many**

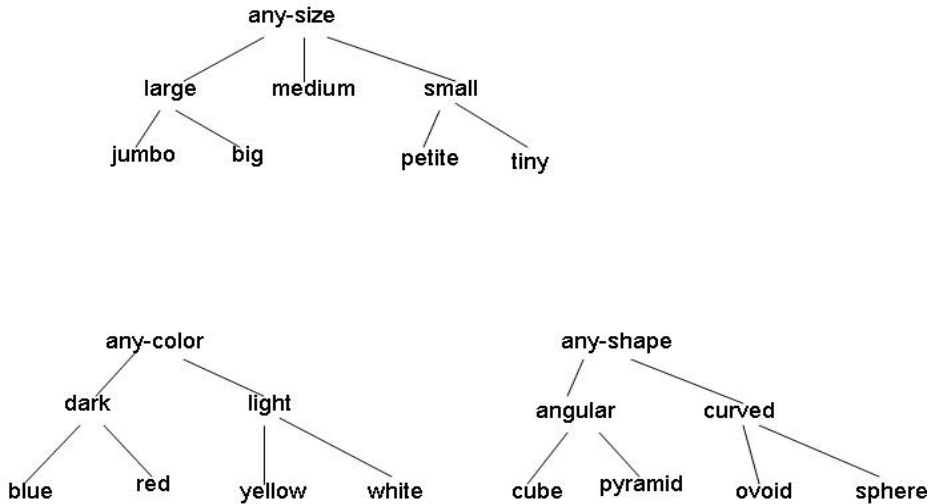
children nodes are there are at the first level of the decision tree? Which branches require a further split in order to create leaf nodes with instances belonging to a single class? For each of these branches, which attribute can you split on to complete the decision tree building process at the next level (i.e., so that at level 2, there are only leaf nodes)? Draw the resulting decision tree, showing the decisions (class predictions) at the leaves.

At the first level of the tree there are three children nodes (sunny, overcast, rainy). Since all overcast conditions result in "Play" no further branching is necessary for this node. However, sunny and rainy need to split. In order to ensure only 2 levels in a tree to fully describe overcast sunny must split on humidity and rainy must split on wind conditions.

See the following page for the drawing.

#### 4. Version spaces (25 points)

Consider the version space defined by the concepts that are the conjunction of the following three attributes:



Examples of legal concepts in this domain include [medium light angular], [large red sphere], and [any-size any-color any-shape]. No disjunctive concepts are allowed, other than the implicit disjunctions represented by the internal nodes in the attribute hierarchies. (For example, the concept [large [red v yellow] curved] isn't allowed.)

(a) (5 points) Consider the initial version space for learning in this domain. What is the G set? How many elements are in the S set? Give one representative member of the initial S set.

$G = \{ [any-size, any-shape, any-color] \}$

Since S cannot contain disjunctive concepts it is limited to the conjunction of: 5 (for size), 4 (for shape), 4 (for

color).  $5 * 4 * 4$  is 80 possibilities and [tiny red pyramid] is one example.

**(b) (5 points) Suppose the first instance I1 is a positive example, with attribute values [jumbo yellow ovoid]. After processing this instance, what are the G and S sets?**

The attribute values of I1 (positive) are all leafs in the version space. This means the resulting S set is the same as the attribute values of I1, [jumbo yellow ovoid]. The G set has not changed and is [any-size, any-shape, any-color].

**(c) (5 points) Now suppose the learning algorithm receives instance I2, a negative example, with attribute values [jumbo red pyramid]. What are the G and S sets after processing this example?**

The G set is reduced. The easiest to attributes to reduce are the general cases affected by the negative example. These include the [ \*size any-shape any-color ] (here \* indicates all 1st level nodes, the more general cases). We can continue down the version space removing nodes as we specialize to eliminate the negative example. This leaves us with:

G = { [any-size light any-shape] [ any-size any-color curved] }

The S set is obviously the same as the attribute values (all leafs once again):

S = [ jumbo red pyramid ]

**(d) (10 points - 2.5 points each) For this question, you**

should start with the version space that remains after I1 and I2 are processed. For each of the following combinations of instance type and events, give a single instance of the specified type that would cause the indicated event, if such an instance exists. If no such instance exists, explain why.

**-A positive instance that causes the version space to collapse.**

Simply choose a positive example that we removed from our G set in the previous question such as [any-size, any-shape, any-color] (translates to anything that isn't light or curved). For example: [ big blue cube]

**-A negative instance that causes the version space to collapse.**

Reversing the previous condition: [jumbo yellow ovoid]

**-A positive instance that reduces the size of the version space to a single concept.**

[tiny white cube] reduces the version space to a single concept.

**-A negative instance that reduces the size of the version space to a single concept.**

It is impossible to reduce the version space to a single concept using a solitary negative instance. It will take several because several specializations still need to occur to reduce it to a single concept.

**Bonus: (e) (5 points) If learning ends after I1 and I2 are processed, how many concepts remain in the version space?**

A total of 24 concepts remain in the version space. See

attachment for a written list (easier to work with pen and paper for this one).

