HW2: Classical Planning and Constraint Satisfaction

CS6300: Artificial Intelligence, Spring 2018 **Tucker Hermans**
University of Utah

1 Classical Planning

1. **Define the preconditions and postconditions (add and delete lists) for the two actions. Assume the agent can only pick up a single object with nothing on it, that it can only hold a single object at a time, and that it can only place the object on a horizontally-flat, open surface.**

   We will introduce three new fluents. The first *flat(x)* is true if x is a horizontally-flat surface. The second *in_hand(x)* is true if x is in the agent's hand. Finally *empty(hand)* indicates the agent's hand is empty.

   Additionally, *clear(x)* indicates there is nothing on top of x and it's not in the agent's hand, but it is not necessarily flat. Finally, *on(x, y)* will be used to indicate object x is on object y.

   Using these, we can define the preconditions, add list, and delete list for *pick_up(x, y)* and *place(x, y)*. Note we expanded the definition of *pick_up(x)* to *pick_up(x, y)* to specify the object under x.

PC	**D**	**A**
clear(x)	*on(x, y)*	*clear(y)*
empty(hand)	*empty(hand)*	*in_hand(x)*
on(x, y)	*clear(x)*	

Table 1: Preconditions, add list, and delete list for *pick_up(x, y)*.

PC	**D**	**A**
in_hand(x)	*in_hand(x)*	*empty(hand)*
clear(y)	*clear(y)*	*clear(x)*
flat(y)		*clear(table)*
		on(x, y)

Table 2: Preconditions, add list, and delete list for *place(x, y)*.

   Note that *clear(table)* is included in *place(x, y)'s* add list in the case that y is the table.

2. **How would you express a goal that there are two towers, each of height 3 blocks, with pyramids on top of each tower?**

Let B_1, B_2, B_3, B_4, B_5, and B_6 be blocks. The goal state can be defined generically as a conjunction of fluents (broken into two lines for readability):

$$on(B_1, Table) \ \wedge \ on(B_2, B_1) \ \wedge \ on(B_3, B_2) \ \wedge \ \neg flat(B_3)$$
$$\wedge \ on(B_4, Table) \ \wedge \ on(B_5, B_4) \ \wedge \ on(B_6, B_5) \ \wedge \ \neg flat(B_6)$$

It is excessive to check if B_1, B_2, B_4, and B_5 are *flat* because the *on(x, y)* operations aren't possible if y isn't flat.

3. **Suppose we introduce a new type of *plank* object into the environment, which is long and skinny and must be supported by either the table or two blocks. What necessary changes would you have to make to your planning description to accommodate these changes?**

   First we will assume the supporting blocks must be flat, have equal height, and planks cannot be placed on a plank (must be placed on the table or blocks).

   To accommodate the plank, we will add new fluents and actions.

   The fluent *clear(plank)* represents there being room to fit another block on the plank (similar to the table). We will add *plank_on(x, y, z)* which is true when plank instance x is on y and z. In the case of the plank being on the table, both y and z are the table to avoid making too many new fluents. To determine which actions are valid on which objects, a $plank(x)$ fluent is added that is true when x is a plank object. We will add $same\_height(x, y)$ (returns true if x and y have equal heights) to enforce blocks under a plank being the same height.

   A variation of *pick_up(x, y)* and *place(x, y)* are added as actions that can be performed on a plank. They are defined as *pick_up_plank(x, y, z)* where x is the plank instance and y and z are its supports and *place_plank(x, y, z)* where x is the plank instance and y and z are the destination supports. For both of these actions, y and z can both be the table.

   The actions *pick_up(x, y)* and *place(x, y)* need slight modifications to restrict x from being a plank and we will define the preconditions, add list, and delete list for the new actions.

PC	**D**	**A**
clear(x)	*on(x, y)*	*clear(y)*
empty(hand)	*empty(hand)*	*in_hand(x)*
on(x, y)	*clear(x)*	
$\neg plank(x)$		

Table 3: Update preconditions, add list, and delete list for *pick_up(x, y)*.

PC	**D**	**A**
in_hand(x)	*in_hand(x)*	*empty(hand)*
clear(y)	*clear(y)*	*clear(x)*
flat(y)		*clear(table)*
¬plank(x)		*on(x, y)*

Table 4: Updated preconditions, add list, and delete list for *place(x, y)*.

PC	**D**	**A**
clear(x)	*plank_on(x, y, z)*	*clear(y)*
empty(hand)	*empty(hand)*	*clear(z)*
plank_on(x, y, z)	*clear(x)*	*in_hand(x)*
plank(x)		

Table 5: Preconditions, add list, and delete list for *pick_up_plank(x, y, z)*.

PC	**D**	**A**
in_hand(x)	*clear(y)*	*empty(hand)*
plank(x)	*clear(z)*	*clear(x)*
clear(y)	*in_hand(x)*	*clear(table)*
clear(z)		*plank_on(y, z)*
flat(y)		
flat(z)		
same_height(y, z)		

Table 6: Preconditions, add list, and delete list for *place_plank(x, y, z)*.

We won't consider goal states that contain a plank as a result of a conversation with the course TA Amanda.

2 Crossword Puzzles

1. **Formulate this problem as a CSP where the variables are words. List all the variables and constraints.**

   For this CSP we have two variables: words and spans of blank squares that need to be assigned words. Let each w_i be a word in the provided dictionary and each span s_i is provided by the problem.

$$w_i \in \mathit{Dictionary}$$
$$s_i \in \mathit{Grid}$$

   The goal is then to assign a word w_i to each span s_i such that the constraints of the grid are met. There are three constraints that must be met: each span must be assigned a word of matching length and any intersecting words must agree at their intersection.

   First, the length and assignment constraints:

$$\forall_i \ \mathit{Length}(w_i) = \mathit{Length}(s_i)$$
$$\forall_i \ \mathit{Assigned}(s_i)$$

   Each span on the grid is assigned a word and that word is of equal length to the span.

   For the intersecting constraint, we will define a constraint *IntersectMatch(x, y)*. It is defined as: if x and y intersect on the grid, then they must have the same letter at that intersection. We will check that this is satisfied for each pair of words:

$$\forall_{i,j} \ IntersectMatch(w_i, w_j)$$

   This implication could be rewritten as:

$$\forall_{i,j} \ \neg Intersect(w_i, w_j) \lor (Intersect(w_i, w_j) \land Match(w_i, w_j))$$

2. **Formulate this problem as a CSP where the variables are letters. List all the variables and constraints.**

   Next we will consider the problem at a character level. To keep things simple we will assume all the words in the provided dictionary only contain lower-case letters and no punctuation. Let each character be $c_{i,j}$ and each cell in the grid provided by the problem to be $g_{i,j}$.

$$c_{i,j} \in \{a, b, c, ..., x, y, z, \epsilon\}$$
$$g_{i,j} \in \{blank, shaded\}$$

   With ϵ indicating no assignment. First we will ensure that each spot in the grid $g_{i,j}$ is assigned a character $c_{i,j}$. If $g_{i,j}$ is shaded then $c_{i,j}$ must equal ϵ. If $g_{i,j}$ is blank then $c_{i,j}$ must *not* equal ϵ.

$$\forall_{i,j}\ (c_{i,j},\ g_{i,j}) \in \{(\epsilon, \mathit{shaded}), (a, \mathit{blank}), ..., (z, \mathit{blank})\}$$

Next, we must constrain each span of characters to being a word present in the dictionary. A span of characters is defined by a sequence of blank cells followed by a sequence of shaded cells or the edge of the board. Spans can run left-to-right or top-to-bottom.

Consider the functions *IsVertSpan($g_{x,y}$, $g_{z,y}$)* and *IsHorizSpan($g_{x,y}$, $g_{x,z}$)*. These functions are similar in that they return true if a valid span on the grid (as defined in the above paragraph) exists between the start and end cells inclusively. They differ in that they are both fixed in either the vertical or horizontal direction as words cannot run on a diagonal. A final function $\mathit{InDictionary}(c_{x,y}, c_{z,y})$ returns true if a word created by the character span is in the provided dictionary.

$$\forall_{x,y,z}\ \neg \mathit{IsVertSpan}(g_{x,y}, g_{z,y}) \vee (\mathit{IsVertSpan}(g_{x,y}, g_{z,y}) \wedge \mathit{InDictionary}(c_{x,y}, c_{z,y}))$$
$$\forall_{x,y,z}\ \neg \mathit{IsHorizSpan}(g_{x,y}, g_{x,z}) \vee (\mathit{IsHorizSpan}(g_{x,y}, g_{x,z}) \wedge \mathit{InDictionary}(c_{x,y}, c_{x,z}))$$

That is, all spans are checked and for all legal spans their assigned words must be present in the dictionary.

3. **Suppose we wish to make sure the the crossword puzzle we generate doesn't contain multiple words that are "too similar." We'll say that words w_1 and w_2 are too similar if *either* one can be obtained from the other by changing exactly one character *or* one is a substring of the other. For instance "dog" and "dogs" are too similar (by the second constraint) as are "dog" and "dog" (again, by the second constraint). Similarly, "cats" and "cots" are too similar (by the first constraint). How would you specify these additional constraints in both the by-word and by-letter formulations?**

   Additional constraints for by-word formation: Each word chosen for the crossword puzzle must have an edit distance of greater than 1 with all other words. Edit distance is how many operations must be performed on a word to arrive at another word.

$$\forall_{i,j}\ i \neq j\ \mathit{EditDistance}(w_i, w_j) > 1$$

   Additional constraints for by-character formation: The constraint is similar as for words but we must determine which spans of characters are considered words. In part 2 we showed that a grid span must be valid (a span of cells terminated by a shaded square or the edge of the board) and the assigned character span is in the dictionary. We will capture that logic in a function called $Valid$ to create valid spans:

$$s_i \in \{x : \forall_{a,b,c,d}\ Valid(c_{a,b}, c_{c,d})\}$$

   Now that we have a collection of valid character spans that constitute words, we just need to make sure those words have the proper edit distance with all other words:

$$\forall_{i,j}\ i \neq j\ \mathit{EditDistance}(s_i, s_j) > 1$$