MCS 380/590 Intro To Artificial Intelligence
Dr. William C. Regli
Drexel University Fall 2000
Homework Assignment Solutions #2
Artificial Intelligence (Russell / Norvig)

3.3e There is a wide variety of possibilities here. This is one solution:

Initial State: Monkey in a room with a crate, with bananas suspended just out of reach on the ceiling.

Goal test: Monkey has bananas.

Operators: Hop on crate; Hop off crate; Push crate from one spot to another; Walk from one spot to another; grab bananas (if standing on crate).

Path Cost: Number of operations.

3.5 There is a wide variety of possibilities here. This is one:

a No. The algorithm must still try to explore every branch, because any branch can lead to a sequence of negative-cost transitions or a negative-cost loop.

b If there is a loop with net negative cost, a rational agent would enter that loop and never leave it, and therefore generating an infinite reward after an infinite number of steps. The only reason not to enter the loop is if there is a loop with a better reward elsewhere.

c Humans don't get stuck in loops like this because following the loop never leaves you in the exact same state as before you entered. After the first time (or few times), even the most beautiful scenery starts getting boring, and therefore providing less reward. An accurate model of the domain should therefore include effects on the mental state od the agent. There is also the fact that time should be part of a state, and that means that following a loop would take you to a different state in every iteration.

d Some psychiatric disorders, various forms of addiction etc, that result in "looping" behavior.

3.9 A search space in which IDS performs worse that DFS is when the search space is small, the branching factor is also small and the depth of the solution is known. Consider a space with a branching factor of 1 at every move, and a solution path at depth n. A DFS will take exactly n expansions to find a solution, while an IDS search will take

3.10 If we had two trees growing towards each other, it would suggest that the two halves of the search are somehow aware of each other, and were seeking each other out. We could say that in the figure we have two BFSs. If the kind of search chosen was DFS for instance, the search trees could be growing horizontally, but maybe not towards each other - if DFS took a wrong path in a infinite/very large search space, the searches may never intersect.

Most of the time the search moves in directions that are actually moving apart from each other. There is just a small part of the search space where the two searches are approaching each other.

3.12 Bidirectional search is $\bigcirc(b^{d/2})$ under the assumption that hash table lookup is $\bigcirc(1)$. If instead of using a hash table, we use an exhaustive comparison against all the nodes generated by the backward search (all $\bigcirc(b^{d/2})$ of them), then the resulting search is $\bigcirc(b^{d/2} * b^{d/2}) = \bigcirc(b^d)$.

3.13 At least one direction of the bidirectional search must be BFS. A good choice for the other direction would be IDS, because it is optimal and complete, and the search space is large and the depth of the solution is not known. If space is u available, Breadth-first could be considered.

3.20 This is not the only way to set these problems:

a Cryptarithmetic:

Variables: Each letter Domain: {0,...,9} Constraints: Each variable has a different value; The arithmetic sum(or product or whatever) must be correct.

b Channel routing in VLSI layout:

Variables: the wires Domain: set of possible paths from source to destination. Constraints: should limit the # of wires through each channel. Again, other formulations are possible.

c Map coloring:

Variables: Each country Domain: Available colors Constraints: No two adjacent variables have the same value.

d Floor Planning:

Variables: rectangular tiles on the rectangular floor Domain: a 3-tuple: x and y coordinates of the cell where the upper-left hand corner of the rectangle will be placed, plus an indication of whether the rectangle is placed withvthe long side horizontally or vertically, Constraints: No tile overlapping.

or

Assume the floor is divided into a grid of squares, and that each rectangle has sides that are a multiple of the size of each grid.

Variables: each cell in the grid Domain: set of rectangles to be placed. Constraints: if a rectangle r of size m x n is placed in cell c, then some neighboring cells around c forming a retangle of size m x n must also have c as their value.

3.7 A problem still has an initial state, operators (or successor function), and a goal test. There is no path cost function. A solution is a state that passes the goal test. The algorithm is the same, except that there is no need for nodes - you can keep a queue of states.

Of the problems in section in section 3.3, the 8-queens, cryptarithmetic, and VLSI problems are the best seen as pathless.

3.8 If we are given a pathless search algorithm then we have no nodes to keep track of the paths - all we have is state. That suggests that we augment the state to keep trakc of the previous state; from that we can recover the whole path.

For the route-finding problem, a state is comprised of two components: the city we are currently in, and the state we just came from. The operator "Move to city X" takes a state as input as generates a new state in which the current city is X and the previous state is s.