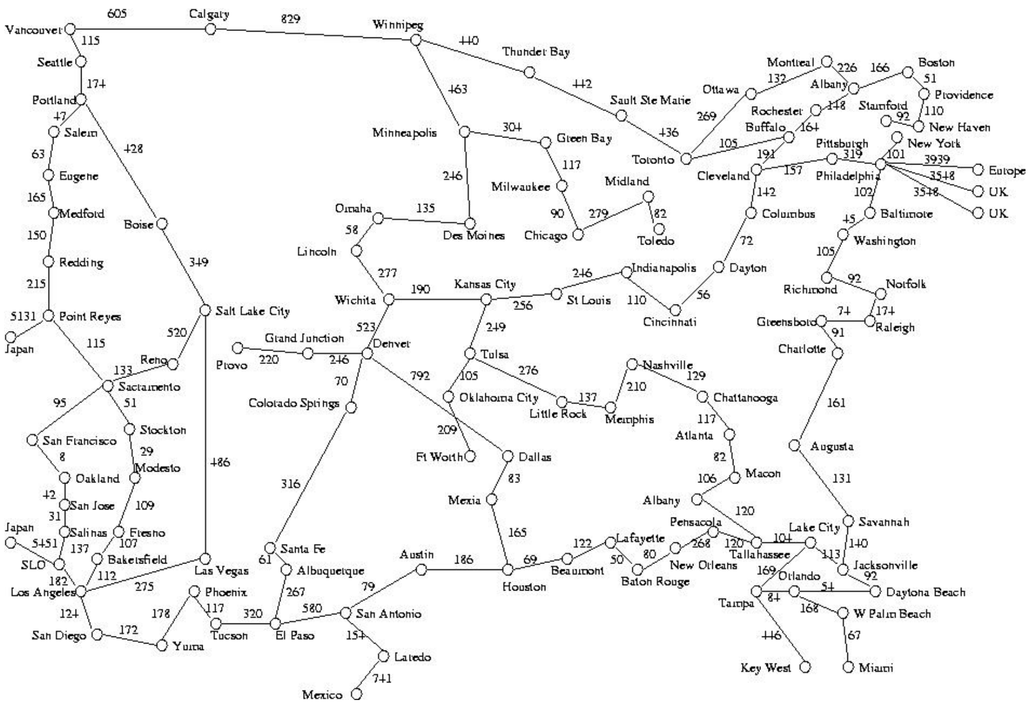
Course: CSC 520, Introduction to Artificial Intelligence
Homework 3
Student: Xusheng Xiao
Unity ID: xxiao2
Email: xxiao2@ncsu.edu

---

1. (80 pts.) This question concerns route-finding, with comparison of several search algorithms. This time, we're in the U.S. Here's jpg of the map below. The solution consists of the series of cities the agent must pass through, each city connected to one or more others by roads of the indicated length. There are no other roads.



The road system is implemented as Prolog procedures in usroads.pl.
A node is said to be expanded when it is taken off a data structure and its successors generated. In this code, the structure is a priority queue implemented as a sorted list. (There are more efficient ways to implement such queues.)
The straight-line distance between cities is computed using decimal degrees of latitude and longitude by heuristic.pl.
Using straight-line distance as the heuristic, and starting from the astar.pl

implementation, modify the code into a working Prolog implementation of A*, then perform some experiments.
The following modifications to the code will be necessary before you start:

(a) The map indicates distances in miles, while the heuristic code uses kilometers as the units. Change the heuristic code to use miles.

(b) The heuristic uses 45 degrees North latitude for all node pairs, which doesn't work well in North America. Change the heuristic code to use the average latitude of the two cities instead of 45 degrees.

(c) Name this new heuristic source file heuristic2.pl. For ease of use, you can keep the procedure name heuristic/3.

The subproblems:

(a) (10 pts.) Now experiment with executing the combination of astar.pl, usroads.pl, and heuristic2.pl to find various paths, until you understand the meaning of the output. Are there any pairs of cities (A,B) for which the algorithm finds a different path from B to A than from A to B? Are there any pairs of cities (A,B) for which the algorithm expands a different total number of nodes from B to A than from A to B?

Answer: There are no such pairs of cities (A,B) for which the algorithm finds a different path from B to A than from A to B. The reason is that the heurisic of computing straight line distance in the heuristic2.pl is admissible.

(b) (10 pts.) Compare the working of astar.pl with the two heuristics (heuristic1.pl) and (heuristic2.pl) for the same map usroads.pl. Are there differences between the performance of the two heuristics (either the length of the paths or the number of nodes expanded before reaching the final solution)?

Answer: I found a difference when apply these two heuristics to find path from orlando to lakeCity. heuristic1: $[orlando, daytonaBeach, jacksonville, lakeCity]$ (cost 259). heuristic2: $[orlando, tampa, lakeCity]$ (cost 253). The reason is that heuristic1 uses km instead of mile, which overestimates the straightline distance between cities. As a result, heuristc1 is not admissible, thus not able to find optimal path every time.

(c) (10 pts.) Change the astar.pl code and/or the heuristic2.pl code so as to implement branch-and-bound search, as discussed in the web notes.

(d) (10 pts.) Do enough exploration to find at least one path that is longer using branch-and-bound than that found using A*, or to satisfy yourself that there are no such paths. Find at least one path that is found by expanding more nodes than the comparable path using A*, or satisfy yourself that there are no such paths. If there is such a path, list the nodes in the path and the total distance.

Answer: There are no such paths found by branch-and-bound that is longer using branch-and-bound than that found using A*, since both of these two searches find optimal paths. From orlando to lakeCity, branch-and-bound expands more nodes than A*. The nodes expanded by branch-and-bound is $[orlando, daytonaBeach, tampa, jacksonville, westPalmBeach, miami]$, while the nodes expanded by A* is $[orlando, daytonaBeach, tampa, jacksonville]$. The path found by branch-and-bound and A* is $[orlando, tampa, lakeCity]$ (cost 253).

(e) (10 pts.) Change the astar.pl code and/or the heuristic2.pl code so as to implement greedy search, as discussed in the web notes.

(f) (10 pts.) Do enough exploration to find at least one path that is longer using greedy search than that found using A*, or to satisfy yourself that there are no such paths. Find at least one path that is found by expanding more nodes than the comparable path using A*, or satisfy yourself that there are no such paths. If there is such a path, list the nodes in the path and the total distance.

Answer: From orlando to lakeCity, greedy found a longer path than A*. The path found by greedy is $[orlando, daytonaBeach, jacksonville, lakeCity]$ (cost 259), while the optimal path found by A* is $[orlando, tampa, lakeCity]$ (cost 253).

(g) (10 pts.)Change the astar.pl code and/or the heuristic2.pl code so as to implement dynamic programming search, as discussed in the web notes.

(h) (10 pts.)Do enough exploration to find at least one path that is longer using dynamic programming than that found using A*, or to satisfy yourself that there are no such paths. Find at least one path that is found by expanding more nodes than the comparable path using A*, or satisfy yourself that there are no such paths. If there is such a path, list the nodes in the path and the total distance.

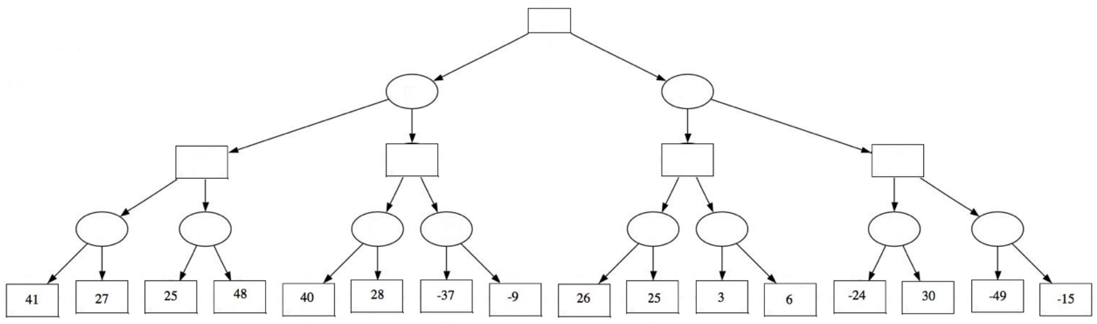
Answer: There are no such paths found by dynamic programming search that is longer using branch-and-bound than that found using A*, since both of these two searches find optimal paths. From orlando to tallahassee, dynamic programming expands more nodes than A*. The nodes expanded by dynamic programming is $[orlando, daytonaBeach, tampa, jacksonville, westPalmBeach, miami, lakeCity, savannah]$, while the nodes expanded by A* is $[orlando, daytonaBeach, tampa, jacksonville, lakeCity]$. The path found by dynamic programming and A* is $[orlando, tampa, lakeCity, tallahassee]$ (cost 357).

Submit the following:

(a) Submit your modified code for the heuristic.

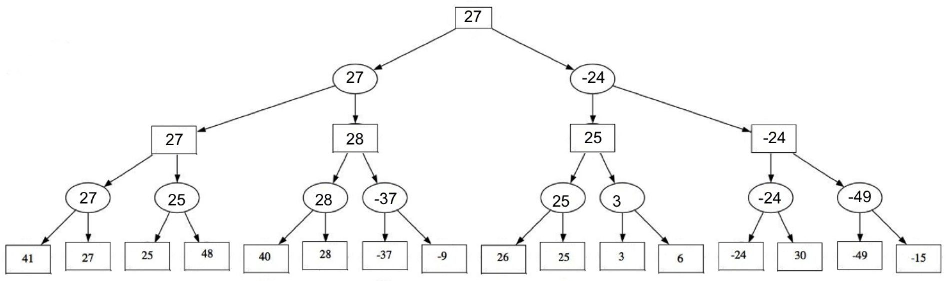
(b) Submit your modified code for branch-and-bound, greedy, and dynamic programming As part of your answer, compare the solution paths and explain what happened, especially any weird behavior you might detect.

2. (35 pts.) Consider the game-tree in the figure below. The leaf node contain the values generated by some unknown static evaluation function. The root node (level 0) represents the current state of the game and agent must pick the next best state to move to.



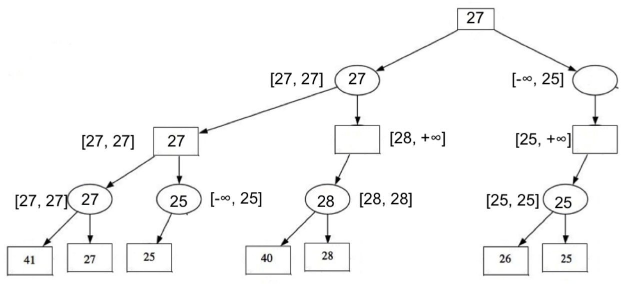
Now, answer the following questions:

(a) (8 pts.) Explore the game-tree just as the agent would using Min-Max algorithm discussed in class. At each level i, indicate the value of the nodes. Indicate the next best state picked by the agent.



(b) (12 pts.) Explore the game-tree just as the agent would using Alpha-beta pruning discussed in class. At each level i, indicate the value of the nodes and indicate the correct set of nodes that will be pruned.

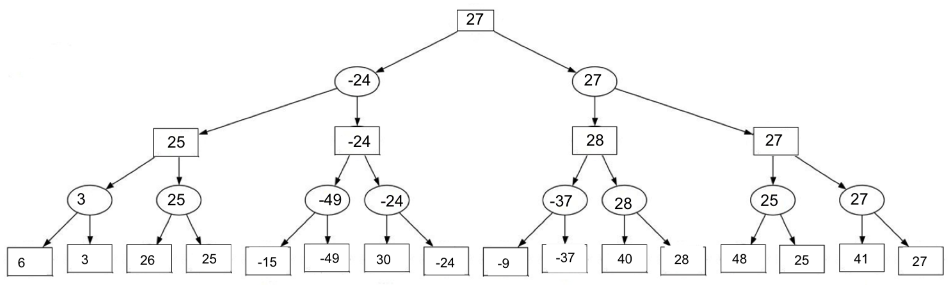
You do not have to label alpha and beta. Indicate the next best state picked by the agent.



(c) (5 pts.) What is the worst case for alpha-beta pruning? In the worst case, is the performance of alpha-beta pruning the same as min-max?

Answer: The worst case for alpha-beta pruning is to expand compute the values of all the nodes. In the worst case, its performance is the same as min-max.

(d) (10 pts.) Assuming that the values in the leaf nodes, the depth and branching factor remain the same, rearrange the leaves so that the new configuration represents the worst case behavior for alpha-beta pruning. Show the resulting graph and repeat 2b.



3. (20 pts.) Ex. 10.4, p. 397 in the textbook. The original STRIPS planner was designed to control Shakey the robot. Figure 3 shows a version of Shakey's world consisting of four rooms lined up along a corridor, where each room has a door and a light switch. The actions in Shakey's world include moving from place to place, pushing movable objects (such as boxes), climbing onto and down from rigid objects (such as boxes), and turning light switches on and off. The robot itself could not climb on a box or toggle a switch, but the planner was capable of finding and printing

out plans that were beyond the robot's abilities. Shakey's six actions are the following:

- $Go(x, y, r)$, which requires that Shakey be Atx and that x and y are locations In the same room r. By convention a door between two rooms is in both of them.
- Push a box b from location x to location y within the same room: $Push(b, x, y, r)$. You will need the predicate Box and constants for the boxes.
- Climb onto a box from position x: $ClimbUp(x, b)$; climb down from a box to position x: $ClimbDown(b, x)$. We will need the predicate On and the constant $Floor$.
- Turn a light switch on or off: $TurnOn(s, b)$; $TurnOff(s, b)$. To turn a light on or off, Shakey must be on top of a box at the light switch's location.

Write PDDL sentences for Shakey's six actions and the initial state from Figure 3. Construct a plan for Shakey to get Box_2 into $Room_2$.

Assume that each room has $5 * 3$ locations (5 columns and 3 rows). Corridor has 4 locations for 4 doors. Each door is at location 10 in each room.

$Constants : shakey, b_1, b_2, b_3, b_4, r_1, r_2, r_3, r_4, d_1, d_2, d_3, d_4, floor, c$
$Predicates :$
$corridor(X)$ - X is corridor
$room(X)$ - X is room
$box(X)$ - X is box
$switch(X)$ - X is switch
$on(X, Y)$ - X is on Y
$at(X, Y, Z)$ - X is at location Y in Z, Z can be room or corridor
$lightOn(X)$ - X is turned on
$lightOff(X)$ - X is turned off
$between(D, X, R, Y, C)$ - D is a door between location X in room R and location Y in corridor C

$Init(box(b_1) \wedge box(b_2) \wedge box(b_3) \wedge box(b_4) \wedge room(r_1) \wedge room(r_2) \wedge room(r_3) \wedge$
$room(r_4) \wedge door(d_1) \wedge door(d_2) \wedge door(d_3) \wedge door(d_4) \wedge switch(s_1) \wedge switch(s_2) \wedge$
$switch(s_3) \wedge switch(s_4) \wedge corridor(c) \wedge at(b_1, 5, r_1) \wedge at(b_2, 14, r_1) \wedge at(b_3, 13, r_1) \wedge$
$at(b_4, 1, r_1)) \wedge at(d_1, 10, r_1) \wedge between(d_1, 10, r_1, 1, c) \wedge at(d_2, 10, r_2) \wedge between(d_2, 10, r_2, 2, c) \wedge$
$at(d_3, 10, r_3) \wedge between(d_3, 10, r_3, 3, c) \wedge at(d_4, 10, r_4) \wedge between(d_4, 10, r_4, 4, c) \wedge$
$at(shakey, 7, r_3) \wedge at(s_1, 15, r_1) \wedge at(s_2, 15, r_2) \wedge at(s_3, 15, r_3) \wedge at(s_4, 15, r_4) \wedge$
$lightOn(s_1) \wedge lightOff(s_2) \wedge lightOff(s_3) \wedge lightOn(s_4)$
$goal(at(b_2, 10, r_2))$

% shakey go from location X to location Y in room R
$action(go(X, Y, R),$
$PRECOND : room(R) \wedge at(shakey, X, R)$
$EFFECT : at(shakey, Y, R))$

% shakey go from location X in room R to location Y in corridor C
$action(go(X, R, Y, C),$
$PRECOND : room(R) \wedge corridor(C) \wedge door(D) \wedge at(shakey, X, R) \wedge at(D, X, R) \wedge between(D, X, R, Y, C)$
$EFFECT : at(shakey, Y, C))$

% shakey go from location X in corridor C to location Y in room R
$action(go(X, C, Y, R),$
$PRECOND : room(R) \wedge corridor(C) \wedge door(D) \wedge at(shakey, X, C) \wedge at(D, Y, C) \wedge between(D, Y, R, X, C)$
$EFFECT : at(shakey, Y, R))$

% push a box from location X to location Y in room R
$action(push(B, X, Y, C),$
$PRECOND : room(R) \wedge box(B) \wedge at(B, X, R) \wedge at(shakey, X, R)$
$EFFECT : at(B, Y, R))$

% push a box from location X to location Y in corridor C
$action(push(B, X, Y, C),$
$PRECOND : corridor(C) \wedge box(B) \wedge at(B, X, C) \wedge at(shakey, X, C)$
$EFFECT : at(B, Y, C) \wedge at(shakey, Y, C))$

% push a box from location X in room R into location Y in corridor C
$action(push(B, X, R, Y, C),$
$PRECOND : room(R) \wedge corridor(C) \wedge box(B) \wedge door(D) \wedge at(B, X, R) \wedge at(shakey, X, R) \wedge at(D, X, R) \wedge between(D, X, R, Y, C)$
$EFFECT : at(B, Y, C) \wedge at(shakey, Y, C))$

% push a box from location X in corridor C into location Y in room R
$action(push(B, X, C, Y, R),$
$PRECOND : room(R) \wedge corridor(C) \wedge box(B) \wedge door(D) \wedge at(B, X, C) \wedge at(shakey, X, C) \wedge at(D, Y, C) \wedge between(D, Y, R, X, C)$
$EFFECT : at(B, Y, R) \wedge at(shakey, Y, R))$

% climb up onto a box B at location X
$action(climbUp(X, B),$
$PRECOND : room(R) \wedge box(B) \wedge at(shakey, X, R) \wedge at(B, X, R) \wedge on(floor)$
$EFFECT : on(B))$

% climb down from box B onto location X
$action(climbDown(B, X),$
$PRECOND : room(R) \wedge box(B) \wedge on(B) \wedge at(B, X, R)$
$EFFECT : on(floor) \wedge at(shakey, X, R))$

% turn on switch S on box B
$action(turnOn(S, B),$
$PRECOND : room(R) \wedge box(B) \wedge switch(S) \wedge at(S, X, R) \wedge at(B, X, R) \wedge$
$lightOff(S)$
$EFFECT : lightOn(S))$

% turn off switch S on box B
$action(turnOff(S, B),$
$PRECOND : room(R) \wedge box(B) \wedge switch(S) \wedge at(S, X, R) \wedge at(B, X, R) \wedge$
$lightOn(S)$
$EFFECT : lightOff(S))$

A plan that moves box b_2 into room r_2 is:

$$go(7, 10, r_3), go(10, r_3, 3, c), go(3, 1, c), go(1, c, 10, r_1), go(10, 14, r_1),$$
$$push(b_2, 14, 10, r_1), push(b_2, 10, r_1, 1, c), push(b_2, 1, 2, c), push(b_2, 2, c, 10, r_2)$$

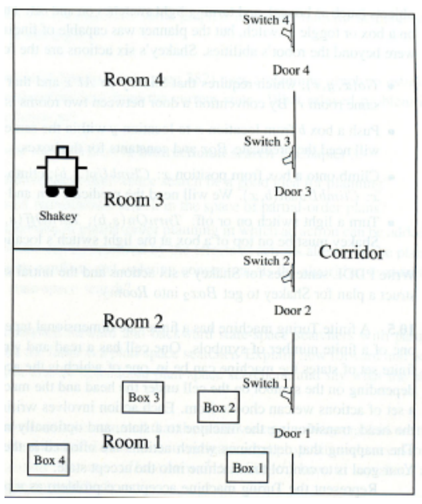


Figure 1: Shakey's World. Shakey can move between landmarks within a room, can pass through the door between rooms, can climb climbable objects and push pushable objects, and can flip light switches.