CS 4700: Foundations of Artificial Intelligence
Homework 2 Solutions

1. (30 points) Imagine you have a state space where a state is represented by a tuple of three positive integers (i,j,k), and you have three actions: decrease i by 1 (as long as i > 0), decrease j by 1 (as long as j > 0), and decrease k by 1 (as long as k > 0). The goal state is (0,0,0). Assume that a given search method does not revisit states it has already seen, and that whenever there are multiple successors for a given state it first expands the state you get from the action of decreasing i by 1 (if possible), then the action of decreasing j by 1, then k.
   a. (1 point) What is the branching factor for this problem?

      3. You can decrease by 1 either i, j, or k.

   b. (2 points) Is this state space a graph or a tree?

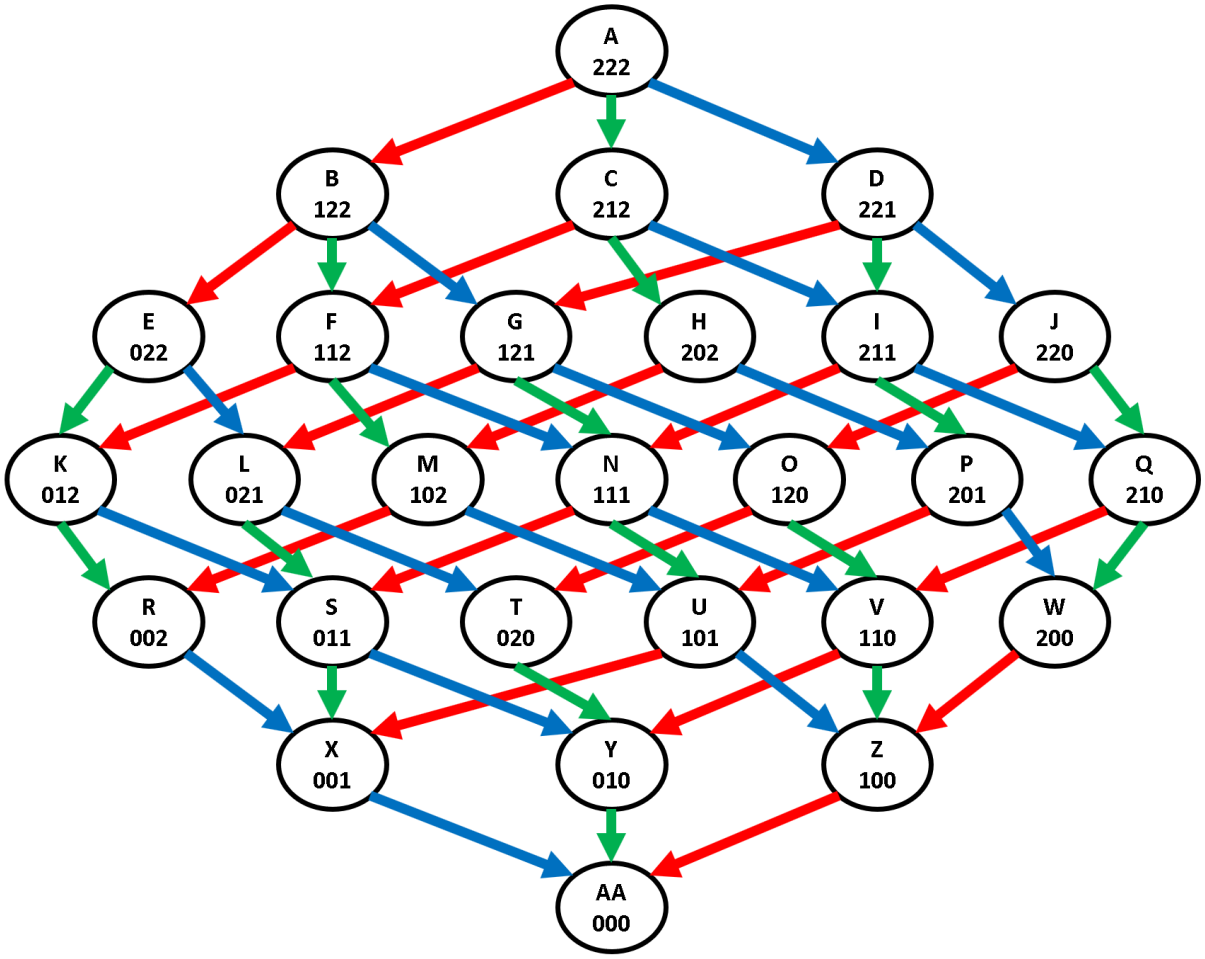
      Graph. You can get to the same single state through multiple paths.

   c. If the initial state is (2,2,2):
      i. (3 points) Draw the subset of the state space that you can reach from this state.
      ii. (3 points) Label the states with the numbers 1, 2, 3, …, to show the order in which they would be expanded by depth-first search.
      iii. (3 points) Label the states with the upper-case letters A, B, C, …, to show the order in which they would be expanded by breadth-first search.
      iv. (3 points) Label the states with the lower-case letters a, b, c, …, to show the order in which they would be expanded by iterative deepening search.

      The answers here refer to the figure that follows. I'm displaying things a bit differently than asked for, to help clarify the answers. In particular:

      - A red/green/blue arrow refers to the action of decrementing the i/j/k coordinate by 1. The arrows emerge from each state in the order red/blue/green to visually depict the order in which successors get expanded.
      - Each level contains the states with the same i+j+k value. These correspond to the set of states reachable in the same number of steps. Within each level states are listed in increasing value: 022 is to the left of 112, and so on left to right till 22.
      - Instead of labeling the figure with the order in which states are expanded by each search method I've labeled each state with a letter, and the answer lists the letters of states in the order in which they would be expanded.
      - Full credit whether or not AA was explicitly listed as the final state. For iv a vertical bar is used to separate the states visited in each iteration of DFS.

      i. See the figure.
      ii. A B E K R X AA
      iii. A B C D E F G H I J K L M N O P Q R S T U V W X Y Z AA
      iv. A | A B C D | A B E F G C H I D J | A B E K L F M N G O C H P I Q J | A B E K R S L T F M U N V G O C H P W I Q D J | A B E K R X S Y L T F M U Z N V G O C H P W I Q D J | A B E K R X AA



d. If the initial state is (i,j,k), what is the time complexity, as a function of i, j, and k, of:

i. (3 points) Depth-first search

O(i+j+k). It goes straight to the goal in i+j+k steps. First it decreases the i coordinate by 1 i times, then the j coordinate j times, then the k coordinate k times.

ii. (3 points) Breadth-first search

It will need to visit all of the states. There are i+1 possible values for the i coordinate, j+1 for the j coordinate, k+1 for the k coordinate, making the result (i+1)(j+1)(k+1). This is O(ijk).

iii. (3 points) Iterative deepening search

There are multiple ways to approach this. I will give three.

Approach 1: Consider the case where the three coordinates are the same number, let's call it n (in other words, i=j=k=n). This is the worst case scenario.

Let's give the complexity as a function of n. (If you're in a situation where they're not equal this still gives an upper bound on the complexity if you use n = max(i,j,k) since the i=j=k=n case counts a superset of the states.) Note this means:

1) The topmost node is (n,n,n).
2) It will take 3n steps to get to (0,0,0).
3) There are 3n+1 levels (one level for each step you take, plus one level for (n,n,n), which you're at before you take any steps).
4) The total number of states is $(n+1)^3$ because each of i, j, and k can take on one of (n+1) values.

Let's consider the case when n is odd. We'll return to the even case in a moment. This means the number of level is even (if n is odd, 3n+1 is even). The halfway point, $\frac{3n+1}{2}$, falls between two levels. Nodes in the upper half of the graph are between 0 and $\frac{3n+1}{2} - 1 = \frac{3n-1}{2}$ steps from the top. Nodes in the bottom half of the graph are between $\frac{3n+1}{2}$ and 3n+1 steps from the top, or, equivalently, between 0 and $\frac{3n-1}{2}$ from the bottom.

Consider a state that's in the upper half of the graph, namely that is $d \leq \frac{3n-1}{2}$ steps from the top. Let's use the notation F_d to refer to the total number of states from the top of the graph down to level d. The total number of states that iterative deepening visits in the first half of the iterations is the sum of these F_d's as d ranges from 0 to $\frac{3n-1}{2}$, or in other words $\sum_{d=0}^{\frac{3n-1}{2}} F_d$.

Let's consider the second half of the iterations. Notice that the structure of the graph is symmetric, so that the number of states d steps up from the *bottom* is also F_d. This means that if we do a depth-first search up to but not including a level that's d steps up from the bottom (where $d \leq \frac{3n-1}{2}$) the search will visit all $(n+1)^3$ in the state space *except* for the F_d states that are d steps or less up from the bottom, on other words $(n+1)^3 - F_d$. The number of states visited in this second half of the iterations is the sum of these quantities as d ranges from 0 to $\frac{3n-1}{2}$, namely $\sum_{d=0}^{\frac{3n-1}{2}} [(n+1)^3 - F_d]$.

Adding these two together gives the total number of states visited,

$$\sum_{d=0}^{\frac{3n-1}{2}} F_d + \sum_{d=0}^{\frac{3n-1}{2}} [(n+1)^3 - F_d] = \sum_{d=0}^{\frac{3n-1}{2}} (n+1)^3 = \frac{(3n+1)(n+1)^3}{2}$$

That is the formula for the exact number of states. It simplifies to $O(n^4)$.

If n is even you can upper bound the value by adding 1 to n, which results in an odd number, making it possible to then use the analysis above. This still gives $O(n^4) = O(\max(i,j,k)^4)$.

Approach 2: Notice that the F_d's cancel out in Approach 1, so we didn't need to figure out how many states are at any level! If you missed this you might have instead figured out a formula for F_d and gone ahead with the summation (without realizing that everything cancels out at the end). It turns out $F_d = \binom{d+2}{3}$. (There's a name for this: F_d is the d-th *tetrahedral number*.) You would then need to sum up the F_d's using the summation in Approach 1, the part that we ultimately side-stepped due to the cancellation of terms. The sum of the first c tetrahedral numbers is $\binom{c+3}{4}$. (There's a name for this: it's the c-th pentatope number.) With these formulas you can do the algebra and get the same answer as Approach 1.

Approach 3: Each iteration of iterative deepening visits a subset of the nodes in the space. This is upper-bounded by the total number of nodes in the space, (i+1)(j+1)(k+1). There are i+j+k+1 iterations of iterative deepening (including the "iteration" that just looks at the root node), so the total number of states visited is upper-bounded by (i+1)(j+1)(k+1) times the number of iterations [(i+1)(j+1)(k+1)], or in other words $(i+2)^2(j+2)^2(k+2)^2$. This is $O(i^2j^2k^2)$. Notice that if i=j=k=n (the worst-case scenario) this simplifies to $O(n^6)$, which is inferior to the first approach.

e. (3 points) Would your answers to c change if the search method did *not* check for revisiting states (in other words, if you get to a state you've already been to you don't realize it and simply expand it again)?

Depth-first search would still reach the goal state after i+j+k steps. Breadth-first and iterative deepening search would treat the state space as if it were a tree exponential in the depth (i+j+k), resulting in time exponential in each, rather taking time polynomial in i, j, and k.

f. (3 points) Give an admissible heuristic for this problem.

If you're at state (a,b,c) using the function a+b+c is not only admissible, it equals h*. (Note that h=0 for all states is also admissible, albeit far more inferior.)

2. (10 points) Formulate Sudoku as a search problem. What are the states, actions, initial state, and goal condition?

There are multiple ways to represent this.

- One way is to have states be a 9x9 grid/array whose squares are each filled with a single digit 1-9 or a blank; the initial state is a grid whose values correspond to the blanks and filled-in numbers in a particular Sudoku problem; actions correspond to placing a digit 1-9 in an empty square of the grid; and the goal condition is that a single digit does not appear twice in any row, column, or the relevant 3x3 subregions.

- Another way is to have states be a 9x9 grid/array whose squares are each filled with a single digit 1-9 (but *not* a blank); the initial state is a grid with a digit in each of the 81

grid locations; actions correspond to switching the digit in a grid location to a new value as long as it is not a square that was given at the start as part of the problem definition; and the goal condition is that a single digit does not appear twice in any row, column, or the relevant 3x3 subregions.

3. (10 points) If h(s) is an admissible heuristic:
   a. (5 points) Is $h_1(s) = h(s)^{0.7}$ an admissible heuristic?
   b. (5 points) Is $h_2(s) = h(s)^{1.7}$ an admissible heuristic?

   Please explain your answers.

   Neither is admissible. Even though h is admissible, there are problems for which $h(s)^{0.7}$ will overestimate h^* and problems for which $h(s)^{1,3}$ will overestimate h^*. In particular, imagine in the first case that for all s $h_1(s) = h^* < 1$ (i.e., a problem where the cost of each action is a number less than 1). In this case raising h to an exponent less than 1 generates a *larger* number, violating admissibility. A symmetric thing happens for $h_2(s)$ if $h^* > 1$.