**AUCSC 460 Lab 4**

First Name: Nathan Last Name: Pelletier ID: 1480408

1. **Goal of Lab 4**
   1. To understand hill-climbing search and beam search. To understand simulated annealing search and be able to implement it using a programming language.

**2. Getting** **Credit for the Lab** This lab handout has questions.

2.1 As part of the lab, you are to write the answers to these questions and construct lab4.py.

2.2 Before the class ends, instructor will check your (unfinished) answers to record effort.

Labs are graded on effort and correctness. The instructor may ask you a few questions to

make sure you understand the material and then record your success.

2.3 You then submit **lab4.pdf, lab4.py, and output.txt** after answering the questions onto the

eclass lab assignment page.

In case that you do not finish it during lab, you can submit your lab assignment on the eclass by due

date (Students who missed lab cannot upload lab document.):

by 11:59 pm Tuesday next week.

The full marks are 100 pts.

If your submission date is Wednesday, you will have 10 pts deduction.

If your submission date is Thursday, you will have 20 pts deduction.

If your submission date is after Thursday, your submission will not be marked. You will get

zero mark.

You can discuss all the following problems with classmates, but do not look at others’ solutions or

copy them.

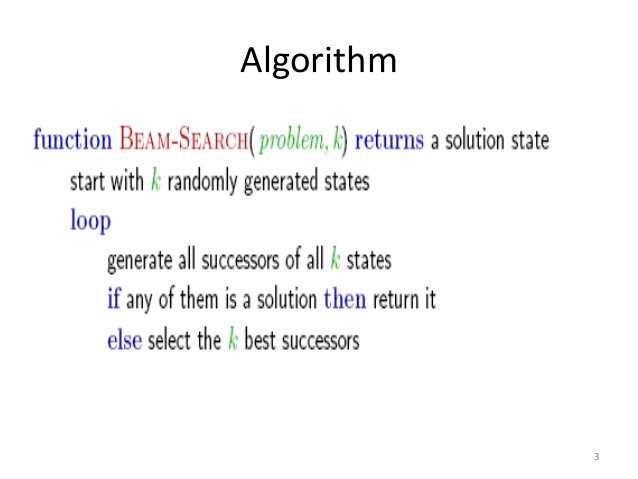
1. The following is the initial state of 8-puzzle. **(a)** **Draw the hill-climbing search tree. (b) Beside each node, write the VALUE of the state.** Manhattan distance will be used as a heuristic function, and a state’s VALUE = **(0-Manhattan distance).** When you compute Manhattan distance, you calculate distances only for the 8 tiles (not the blank space).

You have to follow the algorithm of Figure 4.2 in the textbook. We assume that the action that goes back to the parent node is impossible.

(initial state)

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 | 8 |
| 6 | 7 |  |

1. The following is the beam search algorithm.



Let us assume that beam size k =2. The initial two states will be given as below. **(a) Draw the beam search tree.**  **(b) Beside each node, write the VALUE of the state.** You have to follow the upper pseudo-code algorithm in this document. We assume that the action that goes back to the parent is impossible.

<initial two states>

|  |  |  |
| --- | --- | --- |
| **1** | **2** | **3** |
| **4** | **5** | **8** |
| **6** | **7** |  |

|  |  |  |
| --- | --- | --- |
| **1** | **2** | **3** |
| **4** | **8** |  |
| **7** | **6** | **5** |

1. Understand Figure 4.5 (Simulated annealing algorithm) in textbook and **(a) implement the algorithm in lab4.py.**

(initial state)

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 | 8 |
| 6 | 7 |  |

Your code should follow the algorithm of Figure 4.5. But the following will be changed:

**(b)** *for t =1 to ∞ do*

In your code, the loop will be iterated only finite times. You can decide the iteration number which is greater than 49.

**(c)** A state’s VALUE will **be (0- manhattan distance).**

**(d)** the action that goes back to the parent is impossible.

**(e)** You need to set up *schedule(t)* on your own. Remember that the first *T* will be high (you decide the high value. It can start with 10K or 100 or 10. It’s up to you.) and then *T* should be reduced as time goes by.

**Your code does not need to find an optimal solution.** (Simulated annealing finds an optimal solution when the loop is iterated close to infinity.) **You just need to show that you understood the simulated annealing algorithm and could implement it correctly.**

**(f)** **Your code should print the change of state as follows:**

Initial state: [1 2 3]

[4 5 8]

[6 7 0] (h=6)

Next : [1 2 3]

[4 5 0]

[6 7 8] (h=?(compute on your own) , BAD MOVE)

Next: [1 2 3]

[4 0 5]

[6 7 8] …

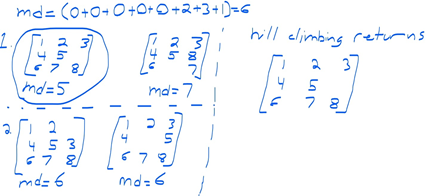
**(g) For each state, display h value, and print ‘BAD MOVE’ if the move was bad (i.e., a new state’s VALUE is not higher).**

**(h) In addition to lab4.py, you have to upload output.txt file which includes the printed output.**

**(i) In lab4.doc, explain how you set up *T*, how many times you iterated the loop, and how your algorithm worked. (Did it work effectively to find a solution?)**

<ANSWER>

1. (Tree with VALUEs)



1. (Tree with VALUEs)

If my algorithm finds multiple children with the same value, it selects the leftmost node.

My algorithm tries moves in the following order: Up, left, right, down.

|  |  |  |
| --- | --- | --- |
| **1** | **2** | **3** |
| **4** | **5** | **8** |
| **6** | **7** |  |

|  |  |  |
| --- | --- | --- |
| **1** | **2** | **3** |
| **4** | **8** |  |
| **7** | **6** | **5** |

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

My algorithm returns:

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 |  |

1. Explain how you set up *T*, how many times you iterated the loop, and how your algorithm worked. (In addition to this answer, you have to upload lab4.py and output.txt.)

I started with T = 50 and loop = 50 but quickly found that no goal states were found, the program continued to make odd moves to the end. I decided to try again with T = 40 and found that the program would stall around move 41. I do not think using probability to make a bad move 4x times will ever work to solve the required problem. Especially not if T = 50 means probabilities of ~90% success and T = 1 means probabilities of ~30%.

Output 1.txt did not work. You can see that it selects a circular path forever, regardless of consequence. I feel as though my probability function is broken.

After testing my probability function, I found that I should stop seeing dumb moves around T = 10 but this is not the case as my code continues to make a loop until T = 0 and then the program stops finding new states. I really don’t understand what went wrong with this one.