Khadija Khaldi, Romita Banerjee and Christoph F. Eick

COSC 4368: Fundamentals of Artificial Intelligence

Fall 2019

Problem Set1

Version 1

Deadline: Sa. February 23, 11p(3% bonus); Tu., Feb. 26, 11p (the latest)

**1) Applying Various Search Strategies to a State Space Khadija**

Assume that you have the following search graph, where S is the start node and G1 and G2 are goal nodes. Arcs are labeled with the cost of traversing them and the estimated cost to a goal is reported inside nodes. Apply the search strategies listed below to the search graph:, (a) indicate which goal state is reached if any, (b) list, in order, the states expanded, and (c) show the final contents of the OPEN and CLOSED lists. (Recall that a state is expanded when it is removed from the OPEN list.) When there is a tie with respect to which node has to be expanded next, nodes should be expanded in alphabetical order. The used search strategies include;

1. *breadth-first*
2. *depth-first*
3. best-first (using f = h)
4. A\* (using f = g + h)
5. [*SMA\**](https://en.wikipedia.org/wiki/SMA*) (using f=g+h and limiting the open-list to just 3 elements)

#### 

#### General Pseudocode for Depth/Breath/Best First Search

OPEN = { startNode } // Nodes under consideration.

CLOSED = { } // Nodes we're done with.

while OPEN is not empty

{

remove an item from OPEN based on search strategy used

- call it X

if goalState?(X) return the solution found

otherwise // Expand node X.

{

1) add X to CLOSED

2) generate the immediate neighbors (ie, children of X)

3) eliminate those children already in OPEN or CLOSED

4) add REMAINING children to OPEN

}

}

return FAILURE // Failed if OPEN exhausted without a goal being found.

#### General Pseudocode for SMA\*/A\*Search

OPEN = { startNode } // Nodes under consideration.

CLOSED = { } // Nodes we're done with.

while OPEN is not empty

{

remove an item from OPEN based on search strategy used

- call it X

if goalState?(X) return the solution found

otherwise // Expand node X.

{

1) add X to CLOSED

2) generate the immediate neighbors (ie, children of X)

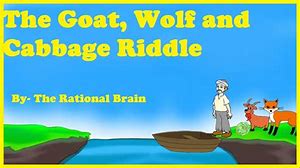
3) add all children to OPEN

}

}

return FAILURE // Failed if OPEN exhausted without a goal being found.

**2) Wolf, Goat, Cabbage Riddle—Describing a State Space Khadija**  
A farmer with his wolf, goat and cabbage come to the edge of a river they wish to cross. There is a boat at the river’s edge. But, of course, only the farmer can row it. The boat also can carry only two things (including the rower) at a time. If the wolf is ever left alone with the goat, the wolf will eat the goat; similarly, if the goat is left alone with the cabbage, the goat will eat the cabbage. Devise a sequence of crossings of the river so that all four characters arrive safely on the other side of the river.

[](https://www.bing.com/images/search?view=detailV2&ccid=V6ZM4FLM&id=AEF5715B9C4BA2E21A98686885AB130497A87797&thid=OIP.V6ZM4FLMbgKyVZcD0vhEnwHaEK&mediaurl=https://i.ytimg.com/vi/0RdKdnVY-Kc/maxresdefault.jpg&exph=720&expw=1280&q=Goat+Wolf+Cabbage+Riddle&simid=607990573965968156&selectedIndex=2)

However, the objective of this problem is not to find a solution but to specify the state space, the applicable operators, and to draw the diagram of the whole state space unambiguously. That is task2, it is about specifying the states and operators of a state space and their semantics precisely.

**3) A\*** Romita

a) Algorithm A\* does not terminate until a goal node is selected for expansion. However, a path to a goal node might be reached long before that node is selected for expansion[[1]](#footnote-1). Why not terminating as soon as the goal node is found? Illustrate your answer with an example!

b) A\* is not guaranteed to find the optimal solution in the case that the function h(s) overestimates the true cost from getting from s to a goal state. Give a state space[[2]](#footnote-2) (describe it similarly to the seach space in problem 2) including a single initial state S and a single goal state G for which A\* will not find the optimal solution; indicate the solution A\* finds for the search search and the optimal solution that A\* does not find for the search space!

4) Programming Problem: Applying A\* to a Road Network Romita

A close up of a map

Description generated with high confidence

You are given a set of cities and the distance between cities. Write a program that finds the path with the smalles distance to travel between the following pair of cities by using your implementation of the A\* algorithm (f=g+h).

1. Minneapolis to Houston
2. San Francisco to Chicago
3. New York City to Los Angeles

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | New York | Los Angeles | Chicago | Minneapolis | Denver | Dallas | Seattle | Boston | San Francisco | St. Louis | Houston | Phoenix | Salt Lake City |
| New York | 0 | - | 713 | 1018 | - | 1374 | - | 213 | - | 875 | - | - | - |
| Los Angeles | - | 0 | - | - | 831 | 1240 | 959 | - | 403 | - | 1374 | 357 | 579 |
| Chicago | 713 | - | 0 | 355 | 920 | 803 | - | 851 | - | 262 | 940 | - | - |
| Minneapolis | 1018 | - | 355 | 0 | 700 | - | 1395 | 1123 | - | 466 | - | - | 987 |
| Denver | - | 831 | 920 | 700 | 0 | 663 | 1021 | - | 949 | 796 | 879 | 586 | 371 |
| Dallas | 1374 | 1240 | 803 | - | 663 | 0 | - | - | - | 547 | 225 | 887 | 999 |
| Seattle | - | 959 | - | 1395 | 1021 | - | 0 | - | 678 | - | - | 1114 | 701 |
| Boston | 213 | - | 851 | 1123 | - | - | - | 0 | - | 1038 | - | - | - |
| San Francisco | - | 403 | - | - | 949 | - | 678 | - | 0 | - | 1645 | 653 | 600 |
| St. Louis | 875 | - | 262 | 466 | 796 | 547 | - | 1038 | - | 0 | 679 | 1272 | 1162 |
| Houston | - | 1374 | 940 | - | 879 | 225 | - | - | 1645 | 679 | 0 | - | 1200 |
| Phoenix | - | 357 | - | - | 586 | 887 | 1114 | - | 653 | 1272 | - | 0 | 504 |
| Salt Lake City | - | 579 | - | 987 | 371 | 999 | 701 | - | 600 | 1162 | 1200 | 504 | 0 |

You use the distances given in the table table to compute g and h is the distance to the target (calculated using latitude longitude values). The latitude and longitude values for the cities are as follows:

New York City: (40.730610 N, 73.935242 W)

Los Angeles: (34.052235 N, 118.243683 W)

Chicago: (41.881832 N, 87.623177 W)

Minneapolis: (44.986656 N, 93.258133 W)

Denver: (39.742043 N, 104.991531 W)

Dallas: (32.897480 N, 97.040443 W)

Seattle: (47.608013 N, 122.335167 W)

Boston: (42.361145 N, 71.057083 W)

San Francisco: (37.7749 N, 122.4194 W)

St. Louis: (38.627003 N, 90.199402 W)

Houston: (29.7604 N, 95.3698 W)

Phoenix: (33.448376 N, 112.074036 W)

Salt Lake City: (40.7608 N, 111.8910 W)

Use can use the following website to calculate the value of h: <https://www.nhc.noaa.gov/gccalc.shtml>

You are free to use any existing library for the implementation but you MUST report any software you use in your ProblemSet1 deliverable. Your program should return the complete path which has the smallest distance and its corresponding distance value.

5) Implementing and Experimenting with Randomized Hill Climbing Khadija

Implement Randomized Hill Climbing and apply it to the optimization problem

* Maximize f(x,y,z)=|x-y-0.2|\*|x\*z-0.8|\*|0.3-z\*z\*y| + (x\*y\*(1-z)\*|z-0.5|) with x,y,z in [0,1]

Your procedure should be called RHC and have the following input parameters:

* sp: is the starting point[[3]](#footnote-3) of the Randomized Hill Climbing run
* p the number of neighbors of the current solution that will be generated
* z neighborhood size; for example if z is set to 0.025 p neighbors for the current solution s are generated by adding vectors v=(z1,z2,z3) with z1,z2, and z3 being random numbers in [-0.025,0.025]
* seed which is an integer that will be used as the seed[[4]](#footnote-4) for the random generator you employ in your implementation.

RHC returns a vector (x,y,z), the value of f(x,y,z) and the number solutions that were generated during the run of RHC.

Run your randomized hill climbing procedure RHC 3 times[[5]](#footnote-5) for the following parameters:

sp= (0.5, 0.5, 0.5), (0,0.5,1), (0.9, 0.6, 0.3)

p= 20 and 100

r= 0.02 and r=0.05

For each of the 24 runs report:

a. the best solution found and its value for f

b. number of solutions generated during the run.

Summarize your results in 4 tables; one for each p and r combination[[6]](#footnote-6). Interpret[[7]](#footnote-7) the obtained results evaluating solution quality, algorithm speed, impact of sp, p, and r on solution quality and algorithm speed. Do you believe with other values for p and r better results could be accomplished? At last, assess if RHC did a good, medium or bad job in computing a maximum for f.

Finally, produce one more run—the 25th run — by choosing parameters for sp, p, r and seed of your own liking and report the solution s found, its value for f(s), and the number solutions searched. Solutions reported for the 25th run that are better will receive more credit.

6) TBDL—*likely something about CSP or Game Theory*

1. That is, a goal state is already in the open-list, but has not been expanded yet… [↑](#footnote-ref-1)
2. For some states h(s) will need to overestimate the true cost as otherwise A\* is guaranteed to find the optimal solution. [↑](#footnote-ref-2)
3. A vector (x,y,z) with x,y,z∈[0,1] [↑](#footnote-ref-3)
4. If you run RHC with the same values for sp,p,r and seed it will always return the same solution; if you run is with the same values for sp, p, r and a different seed, it likely will return a different solution and the number of solutions searched is almost always different. [↑](#footnote-ref-4)
5. Make sure you use a different seed for your random generator to get a different sequence of random numbers for the 3 runs! [↑](#footnote-ref-5)
6. (20,0.02), (100,0.02), (20,0.05), (100, 0.05), and (20, 0.05) [↑](#footnote-ref-6)
7. At least 25% of the available points will be allocated to interpreting the results. [↑](#footnote-ref-7)