Md. Mahin, Theodoros Tavoulareas and Christoph F. Eick

COSC 4368: Fundamentals of Artificial Intelligence Spring 2021

Problem Set1 (Individual Tasks[[1]](#footnote-1) Centering on Search)

Sixth Draft



Fig. 1: Finding a Needle in a Large Haystack with Intelligent Search

Submission Deadlines: Problems 1 and 3: Saturday, February 20, 11p; Problem 2: Saturday, February 27, 11p. Problems 1 and 2 have to be submitted in MS Teams and Problem 3 has to be submitted via Kritik. Be aware of the fact that you will also need to perform Kritic peer reviews in the window February 21-24!

Last Updated: February 2, 2021 at 10a

Weight: 25-35% of the points available for the 3 problem sets!

Comment: Points allocated to each of the 3 tasks are tentative and subject to change.

1. Applying Various Search Strategies to a State Space Theodoros

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.

**Submission Guidelines:**

The solution needs to be as follow:

|  |  |
| --- | --- |
| GOAL Reached first |  |
| Expanded states |  |
| OPEN List |  |
| CLOSE List |  |

**Failure to follow above instruction will lead to point deductions**

2) On Probabilistic Search Algorithms: Implementing and Experimenting with Randomized Hill Climbing *Theodoros*

Implement Randomized Hill Climbing and apply it to a minimization problem involving the following function f:

f(x,y) = [1 + (x + y +1)2(19 - 14x + 3x2 -14y + 6xy +3y2)][30 + (2x - 3y)2(18 -32x + 12x2 + 4y - 36xy + 27y2)], where -2 ≤ x,y ≤ 2

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

* sp: is the starting point[[2]](#footnote-2) 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 z=0.5 p neighbors for the current solution s are generated by adding vectors v=(z1,z2) with z1 and z2 being random numbers in [-0.5,+0.5] uniformly distributed
* seed which is an integer that will be used as the seed[[3]](#footnote-3) for the random generator you employ in your implementation.

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

Run your randomized hill climbing procedure RHC twice[[4]](#footnote-4) for the following parameters:

sp = (0.4, -0.5), (-0.5, 0.3), (1, -2) and (0,0)

p = 30 and 120

z = 0.03 and 0.1

For each of the 32 runs report:

a. the best solution (x,y) found and its value for f

b. number of solutions generated during the run[[5]](#footnote-5).

Summarize your results in 4 tables; one for each p and z combination[[6]](#footnote-6). Interpret[[7]](#footnote-7)the obtained results evaluating solution quality, algorithm speed, impact of sp, p, and z 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 (local) minimum for f.

**Submission Guidelines:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| p=30 & z=0.03 | Run1 | | | Run2 | | |
| #sol se#sol searched  S s | sol | f(sol) | #sol  searched | Bes sol | F f(sol) |
| (2.9, (0.4, -0.5) | 11731 | (-0.600227833141095, -0.4000693397804445) | 30.000097576181673 | 11581 | (-0.6000031114262342, -0.3996127999089084) | 30.000207  846213044 |
| (-2.5( (-0.5, 0.3) | 7981 | (-0.601157294163633, -0.3992473412350722) | 30.000922048064023 | 8011 | (-0.600306929341865, -0.399830965607479) | 30.0000652  8439383 |
| ((4.2 (1, -2) | 20701 | (-0.6002459841893858, -0.3997478142117131) | 30.000055  54925506 | 20641 | (-0.6000545737315334, -0.3998964154608278) | 30.00000  887141724 |
| (0,0) ( (0,0) | 7291 | (-0.5992839463659474, -0.400063010384901) | 30.00055  391621  0563 | 7291 | (-0.5996758098990472, -0.40015810427538506) | 30.0000746  46340096 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| p=120 & z=0.03 | Run1 | | | Run2 | | |
| #sol se#sol searched  S s | sol | f(sol) | #sol  searched | Bes sol | F f(sol) |
| (2.9, (0.4, -0.5) | 43201 | (-0.5994524714263054, -0.40040550287398596) | 120.00027725384693 | 43441 | (-0.6000209899311074, -0.39989253737676556) | 120.00003  702790302 |
| (-2.5( (-0.5, 0.3) | 30361 | (-0.5999553085317225, -0.3999319115805202) | 120.0000554137785 | 30241 | (-0.5999877358055552, -0.3999840796283168) | 120.00  00034  5083892 |
| ((4.2 (1, -2) | 72361 | (-0.600279410971383, -0.3998469473716999) | 120.00  0105  88142014 | 72241 | (-0.6005663767683792, -0.39950795723662447) | 120.000  266  31128954 |
| (0,0) ( (0,0) | 26641 | (-0.6000085552625407, -0.3999793106948348) | 120.00  0000  85923753 | 27001 | (-0.5999247546204314, -0.4001352358857347) | 120.00  0026  55978417 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| p=30 & z=0.1 | Run1 | | | Run2 | | |
| #sol se#sol searched  S s | sol | f(sol) | #sol  searched | Bes sol | F f(sol) |
| (2.9, (0.4, -0.5) | 391 | (-0.5865362399652407, -0.40373685753205824) | 30.152942161614618 | 391 | (-0.6128668647840069, -0.3871189765989539) | 30.14  95113  38208786 |
| (-2.5( (-0.5, 0.3) | 301 | (-0.6115427315212936, -0.39418468295972625) | 30.094747046276073 | 301 | (-0.6041537120424917, -0.40347282232708215) | 30.0632  431237594 |
| ((4.2 (1, -2) | 691 | (-0.5907425662906961, -0.4098240894915842) | 30.0815695  2817661 | 691 | (-0.5808377320675376, -0.4056503615436676) | 30.30325  471562054 |
| (0,0) ( (0,0) | 301 | (-0.6089190470328111, -0.39254093325554656) | 30.0604008  87291486 | 271 | (-0.6032286613961372, -0.38740080641216446) | 30.164593  702578724 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| p=120 & z=0.1 | Run1 | | | Run2 | | |
| #sol se#sol searched  S s | sol | f(sol) | #sol  searched | Bes sol | F f(sol) |
| (2.9, (0.4, -0.5) | 1441 | (-0.5745387597356149, -0.4204597229670748) | 120.5463036609161 | 1561 | (-0.5991075360213416, -0.39962102913087333) | 120.0  06993  48813563 |
| (-2.5( (-0.5, 0.3) | 1081 | (-0.5888185550173005, -0.41007426815745596) | 120.10266183824257 | 1081 | (-0.6105027309654939, -0.3872003849748615) | 120.1  49817  81846409 |
| ((4.2 (1, -2) | 2401 | (-0.5966071975618935, -0.4048289441993847) | 120.024  9706  5177181 | 2401 | (-0.5922571127667565, -0.4127682314939357) | 120.21  1661  59520658 |
| (0,0) ( (0,0) | 961 | (-0.610735538914001, -0.3877355219911144) | 120.132320  92513847 | 1321 | (-0.5989934007796562, -0.40114641542656126) | 120.00114  172370765 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| p=138 & z=0.1 | Run1 | | | Run2 | | |
| #sol se#sol searched  S s | sol | f(sol) | #sol  searched | Bes sol | F f(sol) |
| (2.9, (0.7, -0.13) | 4141 | (-0.6003445515887667, -0.40028777220609196) | 120.00172896783057 |  |  |  |

You should summarize your results in 4 tables formatted as the above, for each of the 4 combination of p & z. Don’t forget to summarize the results of the 33th run and to provide the other information asked for in the project specification!

**Failure to follow above instruction will lead to point deductions**

Originally while filling out the tables I had the belief that the solution would tend towards whatever P was set to. However going back and testing out several different values has resulted in the solution trend to stay the same. Going back and trying to get a singular result seems to not work however if I undo my code to when I did a specific table set I end up getting the exact same result. This makes me wonder if it is instead based upon ram memory or if I just am too tired and typing wrong. It is also important to not that even as the solution set changed x and y variables it all still almost always tended towards either 30 or 120 with no solution ever going above or below that threshold (which as stated before was the same as the P value). Also the occurrence of same number of solutions searched across the table shocked me. Reading the assignment I had believed it would be a lot more rare. This could be due to me only doing a 4 digit and 6 digit seed. Possibly higher and lower seeds carry more randomness but it is hard to tell with this sample size. And finally without fail it appears as if the smallest / best solution to the generated vectors was always negative for both the x and y coordinates regardless of if it took 300 iterations or 72000. The coordinates also tended towards -0.6, -0.4. This even spread I believe is due to using a uniform randomizer that even with different seeds and variables will tend towards the uniform for that randomizer.

3) Solving Discrete Constraint Satisfaction Problems peer-reviewed Mahin

Write a program which finds solution to the following 3 hierarchically organized[[8]](#footnote-8) constraint satisfaction problems, involving 15 variables {A,B,C,…,N,O} which can take integer values in {1,…,50}.

1. Problem A: Find a solution to the constraint satisfaction problem involving the six variables A, B, C, D, E and F and constraints C1,…,C4:
   * (C1) A=B+C+E+F
   * (C2) D=E+F+21
   * (C3) D\*\*2=E\*E\*A + 417
   * (C4) E+F<A
2. Problem B: Find a solution to the constraint satisfaction problem involving ten variables A,…,J which satisfy constrains C1,…,C9:
   * (C5) H\*J+E\*12=(G+**I**)\*\*2
   * (C6) A+D=(F-G)\*\*21
   * (C7) 4\*J=G\*\*2+39
   * (C8) (**I**-G)\*\*8=(F-H)\*\*3
   * (C9) (G-C)\*\*2= F\*C\*C 1
3. Problem C: Find a solution to the constraint satisfaction problem involving 15 variables A,…,O which satisfy constrains C1,..,C15:
   * (C10) 2\*M=K\*\*2 6
   * (C11) (N-**O**)\*\*3 + 7= (F-**I**)\*N
   * (C12) N\*\*2=M\*\*2 + 291
   * (C13) **O**\*\*2=G\*H\***I**\*B + 133
   * (C14) M+**O**=K\*\*2 10
   * (C15) L\*\*3 + **I**=(L+B)\*K

Remark: In the above equations the letters ‘I’ and ‘O’ where put into bold face to avoid to be mistaken as numbers 0 or 1. Moreover, the letter ‘J’ looks somewhat similar than the letter ‘I’ but to better distinguish the two letter ‘J’ is never in bold face.

Your program should contain a counter **nva** (“number of variable assignments) that counts the number of times an initial integer value is assigned to a variable or the the assigned integer to the particular variable is changed; in addition to outputting the solution to the CSV also report the value of this variable at the end of the run, and an interface to call your program for CSP Problems A, B, or C. Your program should return the solution or “no solution exists” and the value of nva after the program terminates. Moreover, terminate the search as soon as you found a solution—do not search for additional solutions.

Remark: To guarantee the anonymity of the peer reviewing, the source code and report you submit should not contain your names or other information that could be used to identify your identity. Moreover, this task will need to be submitted via Kritik and not Blackboard.

Submit a report which

* Gives a brief description of the strategy you used to solve the CSP
* Provides Pseudo Code of your CSP solver
* Explains the Pseudo Code in a paragraph
* Describes strategies (if you employed any) you employed to reduce the runtime of your program, measured by the final value of the variable nva.
* If you conducted a mathematical pre-analysis to eliminate variables, to obtain additional ‘<’ constraints to reduce search complexity or came up with other problem complexity reduction strategies based on such a pre-analysis, describe the results of the pre-analysis you conducted, and how the results of this pre-analysis were used for reducing the search complexity.
* If your program takes advantage of the hierarchical structure of the three CSP problems also explain how this was done.
* If the program you developed is generic in the sense that its code could be reused to solve constraint satisfactions which have a similar structure but different constraints, include a paragraph presenting evidence why your program has this property and what you did to make your program ‘generic’…

Moreover, submit the Source Code for the implementation in a separate file and instructions on how to run your code in a Readme File. Attach the Readme file as an appendix to your report.

Notes on grading:

* Sophisticated approaches that lead to lower complexities in solving the respective CSPs—measured by the final value of the variable nva—will get 10-15% higher scores compared to programs that use brute force approaches.
* Serious penalties will be assessed if the value of the variable nva is not properly computed.

1. Collaboration with other students is not allowed! [↑](#footnote-ref-1)
2. A vector (x,y) with x,y in [-2, 2] [↑](#footnote-ref-2)
3. If you run RHC with the same values for sp, p, z and seed it will always return the same solution; if you run is with the same values for sp, p, z and a different seed, it likely will return a different solution and the number of solutions searched is almost always different. [↑](#footnote-ref-3)
4. Make sure you use a different seed for your random generator to get a different sequence of random numbers for the 2 runs! [↑](#footnote-ref-4)
5. Count the number of times function f is called during the search! [↑](#footnote-ref-5)
6. (30, 0.03), (120, 0.1), (30, 0.1), (120, 0.03) [↑](#footnote-ref-6)
7. At least 25% of the available points will be allocated to interpreting the results. [↑](#footnote-ref-7)
8. A solution of the higher numbered problem also represents a solution of the lower numbered problem! [↑](#footnote-ref-8)