COSC 4368: Fundamentals of Artificial Intelligence
Problem Set1 (Individual Tasks[1] Centering on Search)
Fall 2023

Weight of Task 1	10 Points
Weight of Task 2	30 Points
Weight of Task 3	20 Points

Submission guidelines:

1. **Deadlines: 25 September 2023 (until 11:59 pm)**
2. **Report and Code Submission:** Submit a report for all three tasks together the source codes
3. Submission will be on **CANVAS** (Submission link will be available)
4. Source codes (Implemented in any language of your choice with a README file of program instructions)
5. **Failure to follow all instructions will lead to point deductions!**

---

1) On Uninformed Search

Consider the vacuum-world problem where the agent seeks to move the vacuum machine to clean all locations. Assume discrete locations, discrete dirt, reliable cleaning, and it never gets any dirtier.

a) Which of the uninformed search algorithms would be appropriate for this problem?
b) Apply your chosen algorithm to compute an optimal sequence of actions for a 3×3 world whose initial state has dirt in the three bottom squares and the agent in the center square.

[1] Collaboration with other students is not allowed!

2) On Probabilistic Search Algorithms: Implementing and Experimenting with Randomized Hill Climbing (RHC)

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

Implement randomized hill climbing (RHC) to maximization the following function f:

f(x,y)= $(1.5+x + x*y)^2 + (2.25+x - x*y*y)^2 + (2.625+x - x*y*y*y)^2$ *with* $-4.2 \leq x,y \leq +4.2$

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

- ***sp***: is the starting point[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 $v = (z_1, z_2)$ with z_1 and z_2 being random numbers in [-0.5, +0.5] uniformly distributed
- ***seed***: which is an integer that will be used as the seed[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 of solutions that were generated during the run of RHC.

a) Run your randomized hill climbing procedure RHC twice[4] for the following parameters:

- ***sp*** = (2,2), (1, 4), (-2,-3), (1,-2)
- ***p*** = 65 and 400
- *z* = 0.2 and 0.01
- **seed** = 42 and 43

[2] A vector (x, y) with x, y in [-4.2, +4.2]

[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 if 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.

[4] Make sure you use a different seed for your random generator to get a different sequence of random numbers for the 2 runs!

If you run the program using these parameters, you will find the program is running for 32 iterations. For each of the 32 runs report:

i) the best solution (x, y) found and its value for *f*
ii) number of solutions generated during the run[5].

Summarize your results in the following tables:

For p = 65 and z = 0.2

(x, y)	Seed = 42			Seed =43		
(2, 2)						
(1, 4)						
(-2, -3)						
(1, -2)						

For p = 400 and z = 0.2

(x, y)	Seed = 42			Seed =43		
(2, 2)						
(1, 4)						
(-2, -3)						
(1, -2)						

For p = 65 and z = 0.01

(x, y)	Seed = 42			Seed =43		
(2, 2)						
(1, 4)						
(-2, -3)						
(1, -2)						

For p =400 and z = 0.01

(x, y)	Seed = 42			Seed =43		
(2, 2)						
(1, 4)						
(-2, -3)						
(1, -2)						

b) Finally, run RHC one more time with "your preferred choice" of values for *sp, p, z* (it can be random or some value from your observation) and report the result. Interpret[6] 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 *z* better results could be accomplished? Finally, assess if RHC did a good, medium, or bad job in computing a (local) maximum for *f*. Don't forget to summarize the results of your 33[rd] run[7] and to provide the other information asked for in the project specification!

---

[5] Count the number of times function f is called during the search!
[6] At least 25% of the available points will be allocated to interpreting the results.
[7] Also briefly explain why you chose the particular input parameters for *sp, p* and *z* for your 33[rd] run!

Submission Guidelines:
The followings are expected for submission:

The report should include the followings:

- All 4 tables of obtained results
- Random seed used for your experiments
- Expected results interpretation and conclusions as described above
- Summary of your 33^{rd} run

3) Solving Discrete Constraint Satisfaction Problems

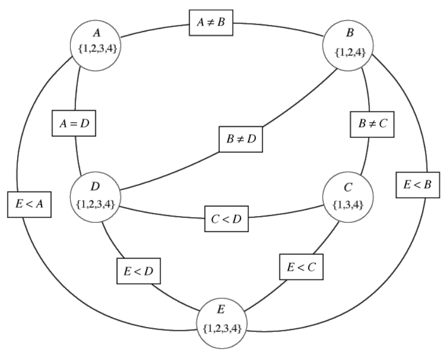

Fig. 2: Example of Constraint Graph

Write a program which finds solution to the following hierarchically organized[8] constraint satisfaction problem, involving 9 variables {A, B, C, D, E, F, G, H, I} which can take integer values in {1, …, 125}. Find a solution to the constraint satisfaction problem involving the six variables A, B, C, D, E and F and constraints C1, …, C5:

- (C1) A=B+C+E+F
- (C2) A-C=(H-F)**2+4
- (C3) D=E+E+21
- (C4) G**2=E*E + 694
- (C5) E+**I**<D

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 assigned integer to the particular variable is changed; Your program should return a 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.

The report should include the following:

- 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 or two
- Describes strategies (if you employed any) you employed to reduce the runtime of your program, measured by the final value of the variable nva.

[8] A solution of the higher numbered problem also represents a solution of the lower numbered problem!

- Conducting a mathematical pre-analysis to eliminate variables, to create an efficient solution. Describe the results of the pre-analysis you conducted, and how the results of this pre-analysis were used for reducing the search complexity.
- Explain how your program takes advantage of the hierarchical structure[9] of the three CSP problems.
- Developing a generic program in the sense that its code could be reused to solve other constraint satisfaction problems which have a similar structure, but different constraints is expected. Include a paragraph presenting evidence why your program has this property and what you did to make your program 'generic'.

[9] If your approach uses solutions of a lower problem to solve the higher problem, e.g. uses solutions of problem A to solve problem B then the proper value for the variable nva should be computer by adding the cost of creating the solutions for A and the cost of finding a single solution for B based on the solutions obtained for A.