Raunak Sarbajna, Md. Mahin and Christoph F. Eick

COSC 4368: Fundamentals of Artificial Intelligence Spring 2024

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



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

Submission Deadlines: Task1: Mo., Feb. 12, 11:59p; Task2: Fr., Feb. 23, 11:59p

Last Updated: February 21, 11a.

Weight: Task weight: T1=35 points, T2=40 points.

1) On Probabilistic Search Algorithms: Implementing and Experimenting with Randomized Hill Climbing *with Resampling* Raunak

Third Draft

Implement a randomized hill climbing algorithm with resampling, called RHCR2 in the following, and conduct a set of experiments, minimizing the following function f:

fFrog(x,y)= xcos(sqrt(|x+y+1|))sin(sqrt(|yx+1|)) +

 (1+y)sin(sqrt(|x+y+1|))cos(sqrt(|yx+1|))

with x,y∈[512,512]

Your procedure should be called RHCR2 and have the following input parameters:

* *sp*: is the starting point[[2]](#footnote-2) of the Randomized Hill Climbing Resampling2 (RHCR2) 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*=10, *p* neighbors for the current solution *s* are generated by adding vectors *v = (z1, z2)* with *z1* and *z2* being random numbers in [-10, +10] 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.



Figure 2: Visualization of fFrog

RHCR2 runs Randomized Hill Climbing 3 times, using sp as the starting position for the first run, the result of the first run sol1 as the start position of the second run which uses a smaller neighborhood size of z/20, and uses the result of the second run sol2 as the starting position of the third run which uses a much smaller neighborhood size of z/400, returning sol3. RHCR2 returns sol1, sol2, and sol3 as well as f(sol1),f(sol2) and f(sol3) and reports how often function f was called in the three runs of RHC and the total number of calls.

**RHCR2(sp,z,p,seed);**

**sol1=RHC(sp,z,p,seed);**

**sol2=RHC(sp,z/20,p,sol1,seed);**

**sol3=RHC(sp,z/400,p,sol2,seed);**

**Return {(sol1,f(sol1)),(sol2,f(sol2)),(sol3,f(sol3))};**

Algorithm1: Pseudo code of RHCR@

Run your randomized hill climbing procedure RHCR2 twice[[4]](#footnote-4) for the following parameters:

*sp* = (-300,-400), (0,0),(-222,-222), (-510,400)

*p* = 120 and 400; *z* = 9 and 50

**REPORT**

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).

1. Summarize your results in 4 tables using the format depicted in Fig. 2; one for each *p* and *z* combination (see example below).
2. Finally, run RHCR2 one more time with “your preferred choice” of values for *sp, p, z*, and report the result; students, who find better solutions in this 33rd run will get more points for the 33rd run subtask.
3. Interpret[[6]](#footnote-6) the obtained results evaluating solution quality, algorithm speed, impact of *sp, p, and z* on solution quality and algorithm speed. Summarize and interpret the complexity of the RHCR2 runs!
4. Did the resampling option lead to better solutions? Do you believe with other values for *p* and *z* better results could be accomplished? Finally, assess if RHCR2 did a good, medium, or bad job in computing a (local) minimum for *f*.

|  |  |  |
| --- | --- | --- |
| *p/z for sp =* (-300,-400),p=120 & z=9 | Run1 | Run2 |
| Number of solutions searched in the first, second and third run of RHC, and the sum of the 3 numbersinS s | <SOLUTION 1, SOLUTION 2,SOLUTION 3> | Result f(sol1),f(sol2),f(sol3) | … | Bes … | F … |
| p=120 & z=50 |   |  |  |  |  |  |
| p=400 & z=9 |  |  |  |  |  |  |
|  p=400 & z=50 |  |  |  |  |  |  |

Fig. 3: Table Template to report the results of the 33 runs of RHCR2

You should summarize your results in 4/8 tables formatted as the above, for each of the 4 combinations of *p* & *z*. Don’t forget to summarize the results of your 33rd run[[7]](#footnote-7) and to provide the other information asked for in the task specification!

*/\* FROG FUNCTION DEFINITION in C++*

*AUTHOR: Raunak Sarbajna\*/*

#include <cmath>

#include <iostream>

double frog (const double *x*[])

{

  double x1 = *x*[0];

  double x2 = *x*[1];

  double term1 =  x1 \* sin (sqrt (fabs (x2 + 1 - x1))) \* cos (sqrt (fabs (x1 + x2 + 1)));

  double term2 =  (x2 + 1) \* cos (sqrt (fabs (x2 + 1 - x1))) \* sin (sqrt (fabs (x1 + x2 + 1)));

  return term1 + term2;

}

int main ()

{

  double x[] = { -100, 100 };

  double result = frog (x);

  std::cout << "f4(x) = " << result << std::endl;

  return 0;

}

**PYTHON:**

import numpy as np

def frog(*x*):

  x1, x2 = *x*

  return (x1 \* np.sin(np.sqrt(np.abs(x2 + 1 - x1))) \* np.cos(np.sqrt(np.abs(x1 + x2 + 1))) +

          (x2 + 1) \* np.cos(np.sqrt(np.abs(x2 + 1 - x1))) \* np.sin(np.sqrt(np.abs(x1 + x2 + 1))))

*# Example usage*

x = np.array([-100,100])

result = frog(x)

print(result)

Fig. 4: C++ and Python Code for fFrog

**Submission Guidelines:**

The followings are expected for submission:

1. A clearly written report. 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 33rd run
1. Source code (Implemented in any language of your choice with a README file of program instructions)
2. Submission will be on MS Teams (Submission link will be available once the assignment is created)

**Failure to follow all instructions will lead to point deductions!**

**Useful Links:**

[Random Number Generator: How Do Computers Generate Random Numbers? (freecodecamp.org)](https://www.freecodecamp.org/news/random-number-generator/)

2) Solving Discrete Constraint Satisfaction Problems (CSP) Mahin

Version 4

![[AI] 3. CSP (Constraint Satisfaction Problems) - Backtracking Search]()

Fig. 5: Constraint Hypergraph for a Letter Equation CSP

Write a program which finds solution to the following 3 hierarchically organized[[8]](#footnote-8) constraint satisfaction problems, involving 13 variables {A, B, C, …, M} which can take integer values in {1, …, 120}; solutions can use the same integer for different variables.

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, …, C5:
	* (C1) A=B\*\*2– C\*\*2
	* (C2) E+F>B
	* (C3) D=B\*\*2-3\*A
	* (C4) (B-C)\*\*2=E\*F\*C-1861
	* (C5) C+D+E+F<120
2. Problem B: Find a solution to the constraint satisfaction problem involving ten variables A, …, J which satisfy constrains C1, …, C12:
	* (C6) (G+**I**)\*\*3= (H-A-1)\*\*2
	* (C7) B\*E\*F=H\*B-200
	* (C8) (C+**I**)\*\*2=B\*E\*(G+1)
	* (C9) G+**I**<E
	* (C10) D+H>180
	* (C11) J < H-C-G
	* (C12) J > B\*G+D+E+G
3. Problem C: Find a solution to the constraint satisfaction problem involving 13 variables A, …, M which satisfy constrains C1, ..., C17:
	* (C13) K\*L\*M=B\*(B+5)
	* (C14) F\*\*3=K\*K\*M\*M\*10 + 331
	* (C15) H\*M\*M=J\*K-20
	* (C16) J+L=**I**\*L
	* (C17) A+D+M=B\*(F-2)

Remark: In the above equations the letter ‘I’ was put into bold face to avoid being mistaken as the number 1. Moreover, the letter ‘J’ looks somewhat similar to the letter ‘I’ but to better distinguish the two letters ‘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 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 develop an interface to call your program for CSP Problems A, B, or C. 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.

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 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.
* Conducting a mathematical pre-analysis to eliminate variables, to obtain additional ‘<’ or ‘>’ constraints to reduce search complexity or developing other problem complexity reduction strategies based on such a pre-analysis, helps 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]](#footnote-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’.

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.

# **Submission Guidelines for Task2**

1. Submit a net, clean, modular code with enough comments + a readme with guideline for running the code (A not runnable code will impact the scores)
2. A well written and structured report
	1. Apart from solution description, the report must also contains any tools used for the assignment. For example, if you have used ChatGPT for report writing or getting any kind of solution, you must mention it why it is used. Similarly, if you used any tool to solve the constraints you should mention it.

# Rubric for Task 2

This rubric is assuming total number of points is 100.

1. Task A: (total 25)
	1. Wrote a valid code (6)
	2. Code produces a solution that satisfies all the given constrains for A (11)
	3. How efficient the code is or number of iterations needed to get to the solution (8)
2. Task B: (total 25)
	1. Wrote a valid code (6)
	2. Code produces a solution that satisfies all the given constrains for A (11)
	3. How efficient the code is or number of iterations needed to get to the solution (8)
3. Task C: (total 25)
	1. Wrote a valid code (6)
	2. Code produces a solution that satisfies all the given constrains for A (11)
	3. How efficient the code is or number of iterations needed to get to the solution (8)
4. Discussion of Approach, (Pseudo) Code, Logic, and Tools used to obtain solutions for Problems A, B, C in the report (15).
5. Generality of Solution Produced (10): The code is reusable for similar tasks and the discussion of reusability in the report.
6. 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 (x1,x2)∈[-512,+512]×[-512,+512]; sample code to visualize f can be found in MS Teams! [↑](#footnote-ref-2)
3. If you run RHCR2 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. [↑](#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. At least 25% of the available points will be allocated to interpreting the results. [↑](#footnote-ref-6)
7. Also briefly explain why you chose the particular input parameters for *sp, p* and *z* for your 33rd run! [↑](#footnote-ref-7)
8. A solution of the higher numbered problem also represents a solution of the lower numbered problem! [↑](#footnote-ref-8)
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 computed 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. [↑](#footnote-ref-9)