University of Stirling
Computing Science and Mathematics
CSCU9YE - Artificial Intelligence

Assignment

AI Methods in Optimisation. The assignment will cover heuristic search (optimisation) to be implemented in Python as this was the topic developed in more details in the labs.

Heuristic Search

- "Welcome! You're our new employee, right? You should already know that our company specialises in software for editing and managing large collections of photographs. Some of our valued customers want to be able to browse their photograph collection in a visually pleasing fashion with similar coloured photographs following each other. I'm not really sure what that means but you can interpret that as having as little difference as possible in the dominant colour of each consecutive photograph. You have a month to prototype some ideas and report back to me." You start thinking about this task and soon some things become clear to you.

- A colour may be represented by three values that define the levels of red, green and blue (RGB). For example (1,0,0) is red.

- To calculate the difference between two colours, we need to measure the distance between them. Since each colour has 3 numerical coordinates RGB (Red, Green, Blue), we can use the *Euclidean* distance. The Euclidean distance between two colours c1 = (x1,y1, z1) and c2 = (x2, y2 , z2) is given by:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}\ .$$

- Your task is then to find an ordering or permutation of colours that minimises the sum of the distances between adjacent colours. This looks suspiciously like the Traveling Salesman Problem we discussed in the lectures! Actually, it is a little bit simpler since you only need to consider a sequence and not a cycle (ie. returning to the initial city). Specifically:
    - Solutions are represented as permutations (ordering) of colours. Your representation can contain a permutation of indices, not the colour themselves. The indices will refer, then, to a list where you store the colours.
    - The objective function is given by the sum of the distances between all pairs of adjacent colours

- You are given a Python script to help with your assignment. The scripts provide two functions, one to open the file of colours, and another to visualise the colour permutations. The script also generates and plot a random permutation of size 100.

- Below you can see some example visualisations.The first example corresponds to a random permutation of colours and the following two are permutations arranged (optimised) with different algorithms. You can clearly notice the difference between a randomised ordering and an optimised ordering.

What you need to do

1. You are given a file with 1000 colours split into their RGB (red, green, and blue) coordinates, for you to use as input in your assignment.

2. Use a subset of the colours or all them, as needed, to test different approaches. For example, you can run your algorithms for test purposes with the first 10 or 100 colours.

3. Implement the following 3 search algorithms to find the permutation with an objective function which minimises the sum of the distance between adjacent elements of the permutation:
   a. **A first-improvement random mutation hill-climbing algorithm**. This starts with a randomly generated permutation and iteratively tries to improve it. Specifically, you should implement the following algorithm:

```
procedure hill-climbing()
begin
    s = random initial solution
    repeat
        evaluate solution (s)
        s’ = random neighbour of s
        if evaluation(s’) is better than evaluation(s)
            s = s’
    until stopping-criterion satisfied
    return s
end
```

   Stopping-criterion: a fixed number of iterations, a parameter that you can set. Neighborhood or move operator: the simplest move operator for a permutation representation is swapping two colour indices selected at random. For example, given solution [5,6,2,3,1,4], the solution [5,1,2,3,6,4] is in its neighborhood. Another operator is the *inversion* operator. This operator works by inverting the ordering between any two colour indices selected at random. For example, given solution [5,1,2,3,6,4], the solution [5,6,3,2,1,4] is in its neighbourhood.

   b. **A multi-start version of algorithm a.** Implementing a loop and calling the hill-climber algorithm **a** from different starting solutions. Your function should receive a parameter indicating the number of repetitions. Your function should record all the solution values obtained for comparisons purposes.

   c. **An algorithm variant of your choice**: Here you're given the opportunity to design your own algorithm. The goal is to try to improve the performance of algorithms **b**. Can you design an algorithm that improves the performance (finds better solutions) than algorithms **b,** given the same computational time? To do so, you can either use some of the algorithm ideas and metaheuristics discussed in the lectures, or you can try your own ideas.

4. Write a report, containing:
   a. **Hill-climbing algorithm.** Indicate which neighbourhood you implemented, justify your choice. Record the trace of objective function values obtained across the run of the algorithm. That is, for each iteration, store in a list the objective function value of the current solution. Once the algorithm has terminated, plot this list of fitness values using Matplotlib. You should see line decreasing, indicating the progress of the algorithm towards smaller distances, that is, better solutions. Produce a plot for the following two sampling sizes: 100 and 500. Record and report the running time of your algorithm in each case. Using the visualisation script provided, produce the plots of the solution found in each case.
   b. **Multi-start hill-climbing algorithm.** Run your multi-start hill-climbing algorithm with 30 as the parameter for the number of repetitions. Compute the mean, median and standard deviation of the 30 solutions produced over two colour sizes: 100 and 500. Using the visualisation script provided, produce the plots of the best solution found for each sample size. Report also the objective function value of the best solution for each sample size.
   c. **Algorithm variant of your choice**. Briefly describe in pseudo-code and in the text the main idea and motivation behind your algorithm. Indicate which neighbourhood operator (s) you used, justify your choice. If you implemented a stochastic algorithm (i.e an algorithm that uses randomisation), compute the mean, median and standard deviation of 30 runs of your algorithm (starting from different initial solutions) over two colour sizes: 100 and 500. Using the visualisation script provided, produce the plots of the best solution found for each sample size. Report also the objective function value of the best solution for each sample size.
   d. **Comparing algorithm performance**. Was your algorithm **c** able to find better solutions than algorithm **b?** Justify your answer with adequate plots or tables illustrating your results and discussing them.

Submission

- The submission should be done using Canvas
- The submission deadline is Monday 11th November before 3 PM. As at 3 PM we are conducting the Demo in the lab. After this time/date, the official penalty per delayed submissions (1.5 marks over 50) per day will be applied.
- The submission will consist of two separate files:
   a. Heuristic search report, in PDF or Word format.
      - Page limit 8 pages

        - You do not need to introduce/describe the problem in the report
        - You need to describe your design choices for the algorithms implemented and present your results in the form of tables and plots.
        - You should structure your report as four sections as indicated above: (1) Hill-climbing algorithm, (2) Multi-start hill-climbing algorithm, (3) Algorithm variant of your choice, (4) Comparing algorithm performance

    b. Heuristic search source code in Python (compressed in a zip file)

Demo

You will need to give a short demo of your heuristic search implementation on Monday 11th November, in the usual lab slot: Room 4X5, Monday 3:00 PM. The demo is mandatory and will be part of the marking.

Marking Scheme

The assignment is worth **50** marks distributed as follows:

- **Demo** (**10**)**:** Marks will be given for showing an understanding and the correct functioning of the search algorithms implemented.
- **Source code** (**20**)**:** Marks will be given for implementing the algorithms. The quality of the source code (Indentation, meaningful identifiers, comments, modular code) will also be considered
- **Report** (**20**): Marks will be given for both presenting a clear recount of your design and implementation choices, as well as showing your results in the form of tables and plots.