Session 1

Metaheuristics

Academic year 2020/2021

This first lab session is focused on the development of both local and global search techniques. The assignment is divided into two parts: Hill Climbing; Simulated Annealing. The student must carry out all that is requested in this document, and to generate a detailed report in which the answers and analyzes carried out are justified. Both **the new code generated and the detailed report must be submitted to Moodle before the deadline**, taking into account that <u>this first practice lasts 3 sessions</u>.

1. *Hill Climbing*. In this first study we are going to work with a local search using the Python code HillClimbing.py. The provided code corresponds to a generic Steepest-Ascent Hill Climbing algorithm to solve the TSP problem. A neighbor will be one having two different cities interchanged. For example, the path [0, 2, 1, 3] will be a neighbor of [0, 1, 2, 3], but it will not be a neighbor of [1, 3, 2, 0]. Please, analyze the code and verify that it is correct and meets the requirements. You can then use the TSP generator provided in the TPGenerator.py file to vary the number of cities that will be used by the Hill Climbing algorithm.
   Answer the following questions in a justified manner and with the help of graphs when necessary:
   - How does this algorithm behave as we increase the complexity of the problem (number of cities in the TSP)?
   - Do you always get the best solution? Why? What does it depend on?
   - Modify the code to start the search again from another initial solution (Iterated local search). Have you managed to improve? Why?

2. *Simulated Annealing.* In this second study we are going to work with a global search using the Python code SimAnnealing.py. The provided code corresponds to a generic simulated annealing algorithm to solve the TSP problem. The algorithm includes an initial temperature of 10 and a cooling function in which the temperature drops 1% in each iteration. The stop criterion is a temperature equal to or less than 0.05. A neighbor will be one having two different cities interchanged. For example, the path [0, 2, 1, 3] will be a neighbor of [0, 1, 2, 3], but it will not be a neighbor of [1, 3, 2, 0]. Please, analyze the code and verify that it is correct and meets the requirements. You can then use the TSP generator provided in the TPGenerator.py file to vary the number of cities that will be used by the algorithm.

   Answer the following questions in a justified manner and with the help of graphs when necessary:
   - How does this algorithm behave as we increase the problem (number of cities in the TSP)?
   - Do you always get the best solution? Why? What does it depend on?

- Analyze how the behavior of the algorithm varies as we change the stop criteria and the initial temperature.
- Modify the code to use different cooling functions:
    - Logarithmic. To is the initial temperature; α is the cooling rate (values less than 1 accelerate cooling); k is the iteration you are in.

      $$T_k = \frac{\alpha\ To}{\ln(1+k)}$$

    - Geometric. To is the initial temperature; α is the cooling rate (values less than 1); k is the iteration you are in.

      $$T_k = \alpha^k\ To$$

  How do these functions affect the final results? Why? Help yourself by representing the values of these functions. Look for new features and compare them to the old ones.
- How would you improve the algorithm? For example, reheating so often. Modify the code with this and any other improvement you guess is appropriate. Analyze the results.