TP1: Simple Heuristics

Goals

The objectives of this sheet are to:

- develop the concepts of optimization and heuristics through implementation;
- recall Python and algorithm coding capabilities;
- introduce the `matplotlib` library.

During the TP session, you should at least do Exercises 1.1 – 1.4. You should also complete Exercises 2.1 – 2.3, as the skills they develop are useful for the projects.

Some other information:

- The coding language for this course is Python.
- Always write tests for each exercise to assure yourselves that your code is correct.
- You can discuss in groups, but the code and tests should be written individually.

1 Random Search and Hill Climbing

Exercise 1.1. A simple version of the hill climbing algorithm terminates after the first iteration where it cannot improve. Implement this variant and use the secret binary vector problem to test it. How does it compare to random search and hill climbing?

Note: you can use the `DomainBinVec` class to generate a secret, using:

```
secret = DomainBinVec(n).sample()
```

Exercise 1.2. Implement a variant of the hill climbing algorithm that changes the first index of the solution that leads to an improvement. For this purpose, you can implement and use a method `D.mutate(x,i)` in the class `DomainBinVec` that changes the value at position i.

Test using the secret binary vector problem and compare to the other approaches.

Exercise 1.3. In this exercise, we look at a simplified version of the board game Mastermind (also similar to Wordle). In this game, the first player chooses a secret code (encoded as a sequence of colors), and the second player must guess the code, knowing only how many colors the each guess and code have in common (and also how many are in the right place).

We consider a simplified version, where the code is a sequence of capital letters, and the objective function is the number of positions where the guess matches the code.

Implement the standard hill climbing and random search for this problem and compare them.

Exercise 1.4. Implement a variant of the hill climbing algorithm to minimize a 2D function.

Test it using the Ackley function, defined by the formula below (see also Figure 1):

$$f(x,y) = 13 - 10\exp\left(-0.14\sqrt{x^2+y^2}\right) - \exp\left(\frac{\cos(2\pi x)+\cos(2\pi y)}{2}\right)$$

Test the algorithm using different numbers of iterations and different step sizes. What do you observe?

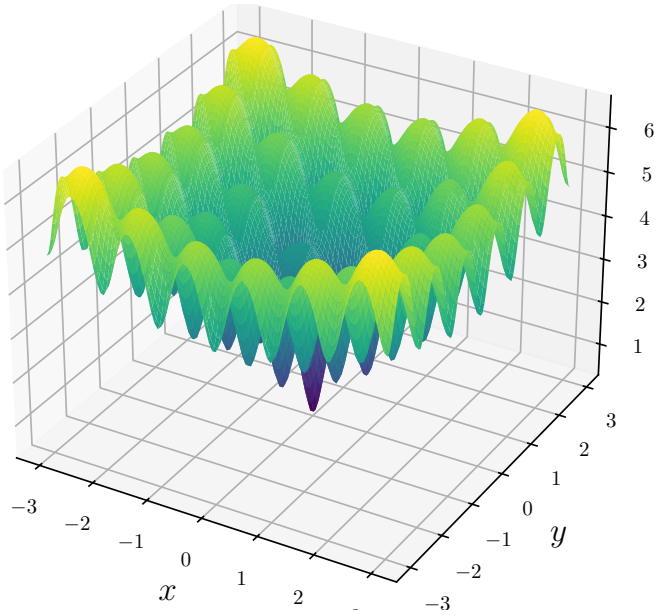

Figure 1: Plot of the Ackley function, seen in Exercise 1.4

Note: To plot a function as in Figure 1, you can use the code in the file `plot.py` that you can find in the zip archive `bioalg-tp01-code.zip`.

Exercise 1.5 (optional). Implement the full version of the Mastermind game (using letters for the code). The objective function must incorporate both the number of correct letters in the right position, and, with less importance, the number of correct letters in the wrong position.

Test it using standard hill climbing. How does hill climbing with the new objective function compare to the simpler version?

2 Shortest Path: Example Mini-Project (Ungraded)

This section and its exercises provide an example of the kind of tasks that you should expect from the projects. It is completely optional and ungraded, and will also skip the less algorithm-oriented tasks, such as input reading, research, and writing the report.

Exercise 2.1. A class to represent a graph in Python is provided to you in the file `graph.py`, contained in the zip archive `bioalg-tp01-code.zip`.

Familiarize yourselves with the structure of the class and the input reading function.

Exercise 2.2. Familiarize yourselves with the two test files provided, `graph-example.txt` and `graph-metro.txt`, and draw the smallest of the two graphs.

Verify what data they contain and reflect on which data might be useful for a shortest path algorithm.

Exercise 2.3. Implement the A* algorithm, using as heuristic the straight line distance between the points v and t.

Test it on both graphs and check that it computes the shortest path correctly.

How many vertices are visited before reaching the destination, compared to the total number of vertices?

Note: The shortest path from *Louise Michel* (171) to *Liberté* (167) is 1340.

Exercise 2.4 (optional). Implement Dijkstra's algorithm and compare the number of vertices visited to the A* algorithm.

Exercise 2.5 (optional). Experiment with other heuristic functions and different ways to weigh d and h to improve the performance of your algorithm on the Metro graph.

Test with different sources and destinations, and check if your improvements still provide a benefit over the standard A* and Dijkstra's algorithms.