Lab: Search

Instructions

1. Complete Tasks 1 and 2, as detailed below.
2. Re-zip the lab with your changes
3. Submit it to Moodle (strict deadline: before Wednesday 31/01, 10h00).

Important: Do not change any file names or the directory structure. Do not `import` any library other than those already imported in the code provided. Use Python 3.X. Re-zip the files into `lab3.zip` with no directory inside (i.e., as you found the `zip`). Your code must be documented/commented sufficiently so that the logic can be understood at a glance.

Optional: You may check the grading process by looking at the file `check_grade.sh`. If you are running LINUX, you can even run this yourself; or just follow the steps listed to check your work before submission (this file and others such as `generate_graph.py` are provided only for your convenience – you do not need them to complete the tasks).

Task 1

In `run_task1.py` you are given a graph represented by a list of coordinates for each of N nodes (a $N \times 2$ array, loaded from `Nodes.dat` where the n-th row of the array represents the 2D coordinates of the node n; And an $N \times N$ matrix $\mathbf{M}$, loaded from `M.dat`. If $M_{j,k} = c$ then the edge cost from node j to node k is c, except $M_{j,k} = 0$ which implies an infinite cost (not reachable)! Such a graph is shown in Figure 1.

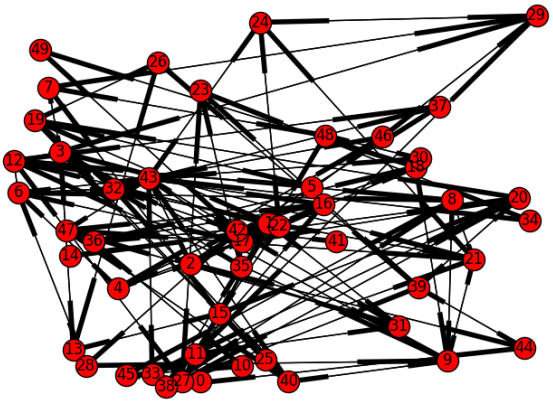

Figure 1: A graph of N nodes $\{n\}_{n=1}^{N}$ (or `0,...,N-1` in Python notation). Edges represent finite costs, and the strength of the cost is represented by the edge tips. Note that edge costs are non-symmetric and in all cases are *at least as* large as the straight-line distance. We could imagine that the graph represents a road network among cities.

The task is to implement the A* search in the function `find_path` in the file `astar.py` to return the shortest path as a Python `list` of nodes, between a given start and the goal nodes (inclusive). Output just `[-1]` if there is no possible path.

Your grade is based on the result obtained from your code on a random graph, and your implementation (of the `find_path` function).

Task 2

In `run_task2.py` there is code which trains a classifier chain $\mathbf{h} = [h_1, \ldots, h_L]$ of L single-label models $h_j : \mathcal{X} \to \{0, 1\}$ (one model for each label) on a given dataset $\{\mathbf{X}, \mathbf{Y}\}$. The classifier chain is implemented in `cc.py`. For each instance $\mathbf{x} = \mathbf{X}_{i,:}$ of the test data, the `predict` function calls a function `explore_paths(self,x,epsilon)` which (in the implementation you are given) carries out a greedy search on the probability tree, returns the branches explored, the predicted path $\mathbf{y}$ to the goal node (the best path of those explored), and the payoff incurred (note: higher payoff is better) according to payoff function $P(\mathbf{y}|\mathbf{x}) = \prod_{j=1}^{L} P(y_j|\mathbf{x}, y_1, \ldots, y_{j-1})$. This is the equivalent to ϵ-greedy search with $\epsilon > 0.5$ (note: the `epsilon` parameter is not used *yet*).

The list of `branches` returned by this function can be used to draw the exploration. Indeed, there is a function in `utils.py` which converts this to GRAPHVIZ markup and writes it to a file that may be later converted to a graph. You may view it by installing GRAPHVIZ or using an online tool like `http://www.webgraphviz.com/` (simply copy and paste the contents of the file there). An example of the result for a particular $\mathbf{x}$ is given in Figure 2 (and also as a `pdf` file). The greedy search tree is actually just a single path.

The task is to re-implement the function `explore_paths` in `cc.py` as ϵ-greedy search, to explore the probability tree. Use the `epsilon` provided for the function. Make sure to also return the best path (wrt $0/1$-loss) and its score as suggested in the current implementation – such that the `predict` function works correctly.

Your grade is based on

- The predictions of the method under your implementation of `explore_paths` (for a given random `seed` and `epsilon`)
- The resulting path (for a given instance and `epsilon`)
- Your implementation (of the `explore_paths` function)

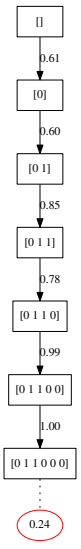

Figure 2: The paths explored by greedy search (just one; unexplored paths are not shown). Each node is represented by the accumulated path in a rectangle, and the value of each edge is an associated payoff. The red oval node attached to the goal node displays the overall path cost (it is not part of the tree).