CS 3445 – Spring, 2022
Project 3

Ant Colony Optimization for the Traveling Salesman Problem

Due: April 28

For this project, I want you to implement the basic Ant System algorithm plus two of the variants (your choice) that were suggested in class last week, and investigate their performance on non-trivial instances of the Traveling Salesman Problem (TSP).

The Algorithms

Ant System

In Ant System, the next leg in a tour is selected probabilistically. For ant k at city i, the probability that it will choose city j to go to next is:

$$p_{ij}^k = \frac{\tau_{ij}^\alpha \eta_{ij}^\beta}{\Sigma_{l \in N_k} \tau_{il}^\alpha \eta_{il}^\beta}$$

where τ_{ij} is the pheromone level on leg (i, j), $\eta_{ij} = 1/d_{ij}$, where d_{ij} is the distance from city i to city j, α and β are positive constants, and N_k is the set of cities ant k has not visited yet. If a city has been visited, of course, the probability of choosing it is 0.0.

The pheromone update is:

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \Sigma_{k=1}^m \Delta\tau_{ij}^k$$

where m is the number of ants, τ_{ij} is the pheromone level on leg (i, j), ρ is the evaporation factor, $\Delta\tau_{ij}^k = 1/L_k$, if leg (i, j) is in ant k's tour; 0.0 otherwise, and L_k is the length of ant k's tour.

Variants

Here are the four variants we decided to look at. I've just called them Variants 1, 2, 3, and 4. See if you can come up with better names.

Variant 1

Pheromones can be substracted on legs of "bad" tours. Suggestions for how this might be operationalized:

- Look at the distribution of tour lengths and subtract pheromones from legs in tours whose length is greater than x standard deviations above the mean.
- Compare the worst tour length to the average tour length and subtract pheromones on the worst tour legs based on the size of the difference.
- In every tour, subtract pheromones on every leg that is not in the best tour so far.

Variant 2

Introduce more randomness in earlier iterations. Suggestions for how this might be operationalized:

- Add pheromones from the best tour so far with a certain probability p_{bsf}. This probability would start low and gradually increase.
- With some probability p_{rand}, choose the next city randomly; otherwise, choose it using the same formula as in Ant System. This probability would start high and gradually decrease. I'm not sure if I'm interpreting this one accurately. Was it that the next city would be chosen randomly, or the entire tour chosen randomly? Either seems reasonable.

Variant 3

Allow different evaporation rates for different legs. Suggestions for what this could depend on:

- the number of "successful" tours the leg was in, or
- the average length of the tours the leg was in.

Variant 4

Legs where the *rate* of increase in pheromone level is higher are more likely to be chosen and, possibly, legs where the *rate* of decrease in pheromone level is higher are less likely to be chosen.

For any of the four variants, if my description does not match your understanding of the idea, feel free to ignore my description and implement your version.

Required ACO Code

You should implement two of these four variants. It's up to you to decide exactly how to implement them. I want you to be able to focus on operationalizing the two variants you choose. For running Ant System, you can use the following parameter settings:

- m (number of ants) $= 20$,
- $\alpha = 1.5$
- $\beta = 3.5$
- $\rho = 0.5$

For the variants you decide to implement, it's up to you to decide what the parameters are and to try to find good values for them. This time I am not going to require that these be specifiable on the command line. Implement this in whatever way makes your testing and data collection easiest.

Given the kinds of tests I want you to run (see below), you will probably want to write your code so that a run can be terminated:

- after a maximum number of iterations is reached, or
- after a tour has been found that is no more than a specified percentage over the optimal (0.0 would mean you will not settle for anything less than the optimal), or
- both, whichever comes first.

You may also want to write your code so that, in addition to these termination conditions, a run will terminate if a specified amount of time has elapsed.

Test Problems for the Experiments

I want you to test your ACO algorithms on instances of the symmetric TSP problem. In the symmetric TSP problem, the distance between any two cities is the same in both directions. I've given you a set of test problems in the directory `ALL_tsp`.

You'll want to use smaller problems to debug your code, but I want you to conduct tests on at least one problem in each of the following ranges of problem size: 2,000-2,999 cities, 3,000-3,999 cities, 4,000-4,999 cities, and 5,000-5,999 cities. My testing using an ACO software package developed by Thomas Stuetzle indicates that these problems are a good test set:

- 2,000-2,999 cities

  `d2103.tsp`

  `u2152.tsp`

  `u2319.tsp`

  `pr2392.tsp`

- 3,000-3,999 cities

  `pcb3038.tsp`

  `fl3795.tsp`

- 4,000-4,999 cities

  `fnl4461.tsp`

- 5,000-5,999 cities

  `rl5915.tsp`

  `rl5934.tsp`

Of course, you're welcome to run tests on larger problems, e.g., `pla7397.tsp` and `rl11849.tsp` seem to be doable, given my tests. There are also a few *much* larger problems, e.g., `pla85900.tsp`, but even the Stuetzle software was not able to solve this one.

Problem files begin with some lines that provide information about the problem. In the example below: the problem name, a comment about the origin of the problem, the type of problem, how many cities, what type of TSP it is (in this example, a 2-d map using Euclidean distances), and a line specifying the beginning of the coordinates section, which is a list of coordinates of the cities (ending with `EOF`). You will need to calculate distances.

```
NAME : d2103
COMMENT : Drilling problem (Reinelt)
TYPE : TSP
DIMENSION : 2103
EDGE_WEIGHT_TYPE : EUC_2D
NODE_COORD_SECTION
1 0.00000e+00 0.00000e+00
2 6.91100e+02 8.56700e+02
3 7.41900e+02 8.56700e+02
4 7.92700e+02 8.56700e+02
5 8.68900e+02 8.63100e+02
6 7.92700e+02 9.07500e+02
7 6.91100e+02 9.07500e+02
8 8.68900e+02 9.13900e+02
          .
          .
2101 3.96750e+03 3.24430e+03
2102 4.01830e+03 3.24430e+03
2103 4.06910e+03 3.24430e+03
EOF
```

To evaluate the performance of these algorithms, you're going to need to know what the optimal tour lengths are. These are available in the file `tsp-optimal-tour-lengths.pdf` I've given you and at:

`http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/STSP.html`

Experiments

We want to test the performance of these algorithms, but what do we mean by "performance?" Of course, we'd like to get the optimal tour, but since that may not happen, we want to be able to do the following:

1. specify some number of iterations and measure how well it does, in terms of percentage over the optimal tour length, given that many iterations, and

2. specify how good you want the answer to be, in terms of maximum percentage over the optimal tour length (so 0.0 would be insisting that you get the optimal tour) and measure the number of iterations needed to get there (subject to some maximum number of iterations).

You want to be able to make statements like:

- “For the `fl3795.tsp` problem, the Ant System algorithm with 100 iterations was able to find a solution that was no more than 1.21 times the optimal tour length (average over 10 runs).”
- “For the `fnl4461.tsp` problem, the Variation 1 algorithm with parameter settings X and a target of no more than 1.05 times the optimal tour length, was able to reach that target in 89 iterations (average over 10 runs).”

Report

Your report should evaluate the performance of Ant System and the two variants you decided to implement. At this point you've had experience writing two reports and gotten some feedback, so you have a good idea of what a report should look like. Here are some things I have mentioned in previous project handouts and in class, but that I want to remind you of:

1. In your section on TSP, be sure to explain why TSP is important.
2. In your section describing ACO algorithms, be sure to include all of the equations that describe how the algorithms operate. When you write a paper of this nature, you should always provide enough detail about the algorithms you used so that if someone wanted to try to duplicate your results, they would be able to do so. This is particularly important for algorithms, like ACO algorithms, that have many different versions.
3. Your text and graphs are both important. Someone should be able to read your text and get a very good sense of your results without looking at the graphs, and your graphs should be clear enough that someone who knows the problem you are addressing (i.e., using ACO algorithms for TSP problems) could look at them and get a pretty good idea of what they are saying without reading the text. To this end, your graphs should be clearly drawn (a title, axis labels with units, a legend if more than one series of values is being shown), and captioned. The graphs should be integrated with the text (not grouped at the end of the report) and the text should refer to the graphs so that the reader can easily connect the text and graphs, e.g. “As Figure 8 shows, the effect of...”

Deadlines

Thursday, April 28:

- Submit a folder containing a copy of your code and your finished report on BlackBoard. All the files I need to run your program should be in a directory that includes a `README` file containing clear instructions for running your program. Your code should be documented well enough that I can easily see what you are doing.
- Submit a hard copy of your report.

- Submit, **individually**, a confidential report assessing each group member's contribution to the project.

May 2–6:

- A project review session with me. These sessions should take between half an hour and an hour. I will ask you questions about your code, e.g. where the code is that does a particular thing, why you did things in a particular way, etc. and your report. I will expect anyone on the team to be able to answer any questions, so even if, for example, someone did not work on the code, they should still understand it well enough to be able to answer questions about it. And everyone should be able to talk about what's in the paper. I may ask you to run your program for me on some test problems.

Grading

The grading will be split among your code, experiments, and report approximately as follows:

35%: the correctness, clarity, and documentation of your code,

15%: the quality and, to a lesser extent, the quantity of your experiments,

50%: the content, organization, and clarity of your report.