CS 3445 – Fall, 2019
Project 3

Ant Colony Optimization for the Traveling Salesman Problem

Due: November 12 (recommended), but no later than November 14

For this assignment, I want you to implement two Ant Colony Optimization (ACO) algorithms and investigate their performance on non-trivial instances of the Traveling Salesman Problem (TSP).

Required ACO Code

You should implement and test:

- Elitist Ant System, and
- Ant Colony System.

Your code should allow you to vary the following parameters:

- the number of ants,
- the number iterations,
- α, the degree of influence of the pheromone component,
- β, the degree of influence of the heuristic component,
- ρ, the pheromone evaporation factor,
- e, the elitism factor in Elitist Ant System, and
- the factors controlling "wearing away" of pheromones in Ant Colony System (ϵ and τ_0), and
- q_0, the probability in Ant Colony System that an ant will choose the best leg for the next leg of the tour it is constructing, instead of choosing probabilistically.

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 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.

Although not required, you may 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`. Also, a `.tar` file containing them is available at:

`http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/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 suggests the following as good candidates, but you are not limited to these:

- 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 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`

Although not necessary for this assignment, the following website:

`http://www.tsp.gatech.edu`

has a lot of TSP trivia and describes the TSP grand challenge: finding a shortest tour for 1,904,711 cities in the world.

Designing Experiments

You want to compare performance and, since the performance of ACO algorithms is dependent on the parameter settings, you want to see how the performance of the algorithms changes with respect to the parameter values. And you want to compare performance on larger problems (almost any reasonable parameter setting will find solutions to small problems quickly). You'll have to decide, for a given set of experiments, which parameters you want to vary, over what range, and how many tests you can manage. Then choose specific values for each parameter that cover that parameter's range, such that it is feasible to test all possible combinations of the values you have specified. Of course, it's a tradeoff between thoroughness and practicality. A possible approach is to set some number of parameters to reasonable fixed values, and restrict your attention to the remaining parameters. For example, it would be fine to say that setting the number of ants to the number of cities is reasonable, and turn your attention to the other parameters. Or you may decide that the impact of the number of ants is something you want to test.

Given a set of parameter settings that you want to test, a reasonable thing to do would be, for each of the two algorithms, to test each setting of parameters on each problem and measure the performance. But what do we mean by "performance?" Of course, we'd like to get the optimal tour in less than a second, but since that's not going to happen, we want to measure how close we get to the optimal tour length and/or how long it takes to get that answer. Since the time per iteration can vary considerably, depending on factors like the algorithm and the number of ants, you probably want to measure the time taken rather than the number of iterations.

So, for example, you could measure the time needed to find the optimal tour, or the time needed to get to within 1.05 times the optimal tour length (or 1.10, or...). Or you could measure how well it does (in terms of the fraction of the optimal tour length) given a fixed number of seconds (if you write your program such that you can terminate after a given number of seconds). You will need to do multiple runs and calculate some summary statistic, e.g. mean and standard deviation and/or median.

Now you're in a position to make statements like: "For the `fl3795.tsp` problem, the Elitist Ant System algorithm with the following parameter settings was able to find a solution that was no more than 1.10 times the optimal tour length in 43.5 seconds (average over 10 runs), and this was better than any other setting of parameters." Then, for example, you could look at what the best parameter settings were for each of the problems and be able to say something about whether there is a single setting that is best over all the problems you tested. Since you won't be testing on a lot of problems, that statement loses some force, but this is, after all, a *smaller* project.

As in the first two projects, the parameter settings affect each other, so it is sometimes a good idea to first test the effect of changing parameter values *one parameter at a time*, and then see what happens when you vary multiple paramters. Do you get the effect you would expect if you just "added" together the effects of the single parameter changes, or do they interact in an unexpected way?

Here are some questions/issues that would be worth looking at:

- How sensitive to parameter changes is each algorithm? Is one algorithm more sensitive than the other? For each algorithm, is there a set of parameter values that work well across all the problems?

- For the more difficult problems, what's the best way to get better results? increase the number of ants? increase the number of iterations? change some parameter values, e.g. α and β? Keep in mind that it could be the case that one approach gives a shorter tour but takes longer, whereas another approach gives a decent answer faster.

- What is the effect as the relative strength of pheromones and heuristic information changes?

- Do your results confirm the following rule-of-thumb parameter settings?
  - 10-30 ants
  - $\alpha = 1$
  - $\beta =$ 2-5
  - $\rho = 0.1$
  - $e =$ number of ants (for Elitist Ant Colony),
  - $\epsilon = 0.1$ and $\tau_0 = 1/(n \cdot L_{nn})$ (for Ant Colony System), where L_{nn} is the length of a nearest neighbor tour, a tour constructed by starting at an arbitrary city and selecting the closest unvisited city to visit next (note that nearest neighbor tours are not unique), and
  - $q_0 = 0.9$ (for Ant Colony System)
- In Elitist Ant System, how important is e, the elitism factor? How does performance vary with the value of e?
- In Ant Colony System, how important is q_0, the probability of choosing the best city next in the tour construction (instead of choosing probabilistically)? How does performance vary with the value of q_0?

These are examples, not a list of requirements. And there are many other things you could look at. You can't possibly test everything – you need to focus your experiments. I'd be happy to discuss your ideas for doing this before you actually start testing.

Report

At this point you've had experience writing two reports and gotten feedback on them, so you have a good idea of what a report should look like, but I am including two excerpts from the results sections of reports submitted in the past (`report-excerpt-1.pdf` and `report-excerpt-2.pdf`) as examples of types of graphs that are nice ways of presenting results.

Here are some things I have mentioned in previous project handouts and in class, but that I want to remind you of:

1. Whatever experiments you run, you will be reporting results for both algorithms. If you have separate graphs for each algorithm, use the same scale for both graphs so that it is easier to make comparisons visually.
2. In your section on TSP, be sure to explain why TSP is important.
3. 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.

4. Since one of the things you should measure is the amount of time taken for an algorithm to find a solution, and not just the number of iterations, you should describe the machine you ran your tests on (processor speed, etc.), the operating system, and the language in which your program is written. Ideally, all the tests would be run on the same machine, but, under the circumstances, it's fine if you use more than one machine, and report that.

5. 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..."

Grading

The grading will be split among your code, experiments, and report 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.

Deadlines

As discussed in class, I am not requiring you to submit a partial draft report. You are welcome to do so, however, and I will provide feedback. Keep in mind that I need 2-4 days to return your draft with feedback.

Tuesday, November 12, but no later than Thursday, November 14:

- 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.

Week of November 18:

- 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 will also ask you to run your program for me on some test problems.