TSP Lab Reflection

Ryan Blocker

April 19th 2024

Craig Partridge

CS320

Code: Here is the code that I uploaded to Zybooks:

```
def ga_tsp(initial_population, distances, generations):

    if (initial_population is None or distances is None or
    generations is None):
        return None
    if not valid_path(initial_population):
        return None
    if generations <= 0:
        return None

    path = best_path(initial_population, distances)
    trial = 0
    while trial < generations:
        new_path = two_opt(path, distances)
        if not valid_path(new_path):
            return None
        path = new_path
        trial += 1
    return path
```

Approach: To be honest I was a little confused for a while doing this lab, it was a little daunting to see all of the given files with a ton of interrelated code that was being used in a bunch of different places. It wasn't until I sat down and did the grunt work of going through each file and understanding what each method did and how they were used in other files to manipulate the data. After doing this I realized this might be the easiest lab I have done to date. Partly because the algorithm given by our guest speaker was made available for use. My code is pretty straight forward. I start with the usual checking if certain parameters and given input is `None` or not correct. Then I create a variable called path and I utilize the `best_path()` method given in `util.py` to get the best path out of the given initial population and dictionary of distances.

Then I create a simple counter to run the loop over the amount of generations. Initially I had this as a for loop but I changed it to a while loop for extra readability. Within the loop I create a variable called `new_path` and use the `two_opt()` algorithm given to us which utilizes the `two_opt_round()` and `two_opt_swap()` methods as well. Finally after getting the output from those methods into `new_path` I check its validity with `valid_path` in utils.py and then finally increment the counter and return path.

Modularity: As far as modularity goes this program is already broken up into multiple files with functionality being split up. So this program could be easily swapped with a different algorithm just by swapping out the `two_opt()` program for a separate algorithm.

Big-O Complexity: Now since this program is split up into multiple different files this was a little trick to figure out but I believe this program is $O(n^2)$ time complexity. Since the `ga_tsp()` function has a while loop for the generations and inside the loop it runs the `two_opt()` function which iterates over the pairs of cities we end up with a final complexity of $O(n^2)$