Project #6: Travelling Salesperson (Your Algorithm)

Teams: This will be a group project on which you will all work in teams of typically 3-4 students. Here are team assignments for sections 1, 2, 3, and 4. You will each submit the exact same report using LS. Your team will all receive the same base grade for the project. To augment this, each team member will send an e-mail to Prof. Martinez for sections 1 and 2 and to byucs312ta@gmail.com for sections 3 and 4. Include your section and group number. The e-mail should include a thoughtful and honest evaluation of the contributions of your team members, including **yourselves**. For each individual, this evaluation should include a score from 0 to 10 indicating your evaluation of their work (10 meaning they were a consistent fully participating member of the team that made significant contributions to the project and were good to work with, 0 meaning they contributed nothing). If you would like, you may also include any clarifying comments, etc. (especially for low scores). If a person receives consistently low evaluations from peers, then their score will be proportionally decreased.

As with project 5, there are three difficulty levels that govern the city distributions: Easy (totally connected symmetric), Normal (totally connected asymmetric), Hard (asymmetric and some infinite distances, not totally connected). You can play with all of them during testing but just use the Hard level for all of your reporting below.

Framework: You will use the same framework used for the TSP Branch and Bound project 5, implementing methods for (at least) two additional solver techniques. See the project 5 link for details on the framework and GUI. Since this is a team project, you can just choose one team member's project 5 code and use it for testing.

Your Algorithm: Your goal is to implement an algorithm which can solve the largest possible city problems with the most possible accuracy (approximation) in "reasonable" time. You can use any algorithm you come up with. It is all right to look on the web or elsewhere to find algorithms, but you should write your own code. Note that since we are requiring "reasonable" runtimes, your algorithm will not find optimal length tours for larger numbers of cities. Do an approximation algorithm, NOT an algorithm which is guaranteed optimal as that will let you do very few cities. You may not use branch and bound (with different bounding functions, or adding a beam, etc.) as that would not give you as much of a new experience.

To Do

1. Implement a greedy algorithm which starts from an arbitrary city, picks the shortest path available to an unvisited city, then picks the shortest path from that city to an unvisited city, etc., until it returns to the origin city. Note that for Hard problems, it is possible to reach a city which has no paths to any remaining unvisited cities (don't forget to check for a path from the last city back to the first to complete the cycle). In such cases, just restart with a different random seed.
2. Implement your own algorithm to solve the TSP. We give extra love at grading time if you try to be somewhat creative in your approach, by either coming up with your own algorithm, or trying to extend or modify a published algorithm. The best algorithms will have a good combination of:
   - % improvement in solution quality (tour length) over random and greedy
   - size of problem (# of cities) that can be handled

   You will use the supplied random algorithm and your greedy algorithm as baselines to see how much better you can do. Your grade will in part be based on these three components:
   a. % improvement over greedy
   b. the size of problems your algorithm can handle
   c. the creativity of your algorithm

   As long as you make a good effort in these areas you can get full credit. If you do your own creative algorithm which may not be as good as published algorithms, you will still get rewarded for your creative effort. There will also be some extra credit for those groups who get exceptional results or who demonstrate significant effort and/or creativity on their algorithm.

Report: There is no graded design experience for this project, but your group should do the design experience together before starting to code.

1. [10] Correct implementation of the greedy TSP algorithm. Brief discussion **and complexity** of the algorithm assuming it will always find a legal path (like with the easy city distribution).
2. [35] Correct implementation of your own TSP algorithm including a discussion of the algorithm, why you chose it, and its pros and cons. (Note that not always returning optimal is NOT a con, as that is expected). Give proper attribution (references) for ideas you find externally. Clearly point out which ideas came from other sources, and which ideas are original contributions that you made. Include a discussion of the theoretical big-O time and complexity of your algorithm. Discuss how the empirical complexity matched your theoretical complexity. Include screenshot examples of typical results for your algorithm.
3. [30] Include a table with columns (see below) for each of the TSP algorithms including the Branch and Bound TSP algorithm you implemented in Project 5 (use one of your individual project 5 implementations as a representative version of B&B; however, do not set your time limit at 60 seconds as you previously did). For the random algorithm, use the default algorithm provided with the original framework (this acts as a baseline for the greedy algorithm). You can play with all three levels of problem difficulty (Easy, Normal, Hard) during testing, but just use the Hard level for all of your reporting. For the greedy algorithm report the improvement over random as a fraction [calculate this as `greedy_path/random_path`]. This will help calibrate and make sure your greedy algorithm is implemented correctly. For B&B and your own algorithm, report the improvement over the greedy algorithm [calculate this as `your_path/greedy_path`]. You should create a table just like the one below with the city sizes shown below (15, 30, 60, 100, 200). Round average tour lengths to the nearest integer. Round time and % improvement to two significant digits beyond the decimal. The results for each cell (all 4 algorithms) should be the average of 5 runs with different random seeds for that number of cities. Do not try to find a particular set of seeds on which your implementation does well. They should be randomly chosen runs. You will fill in average time (seconds) and average tour length for the different numbers of cities. If an algorithm takes more than "reasonable time" (more than about 10 minutes or so) to solve for that number of cities for the majority of your trials, then just fill in that cell for that algorithm with "TB" (Too Big). For B&B and your algorithm, if a minority of the trials do not finish, take the average of the majority that does. The numbers in the example table below are made up. (Note: We give random and greedy algorithms an unfair advantage in our comparison as we allow them an arbitrary number of restarts when they fail in the hard version, which we do not grant to B&B and your algorithm. This points out another way in which they are inferior to your algorithm and B&B.)

	Random		**Greedy**			**Branch & Bound**			**Our Algorithm**		
# Cities	Time (sec)	Path Length	Time (sec)	Path Length	% of Random	Time (sec)	Path Length	% of Greedy	Time (sec)	Path Length	% of Greedy
15	0.001	7743	.004	4100	.53	3.26	3527	.86	1.47	3649	.89
30											
60											
100											
200	.11	100761	.3	15798	.16	TB	TB	TB	165.1	14244	.90
…											

4. [15] To complement the required five rows shown in the table above, you will add to your table some additional rows with different numbers of cities and maintained in increasing city order. Make sure you have some rows around the "sweet spot" for your algorithm, which is around the number of cities where it shows the best balance of lots of cities with good costs. Include a row in your table at the number of cities where it is "Too Big" for your algorithm to solve in a reasonable time limit. Discuss and analyze the overall results from your table.

5. [10] Your write-up should look sharp and follow the form of a brief conference paper with abstract, introduction, algorithm explanation, complexity, analysis of results, and future work. Examples of good work and reports from previous semesters can be found here and here.

Oral Report: Each group will give an oral presentation of their project. You will prepare a short slide presentation (length of presentation is found on your section's team link). Do not take time in the presentation to discuss the basic greedy and Branch and Bound approaches, as everyone will already be aware of that. The presentation should:

- Introduce and explain your algorithm. Along with this briefly review why you chose this algorithm, how much is original, while giving appropriate attribution, any challenges/detours along the way, and the pros and cons of your approach.
- Present and discuss the theoretical big-O complexity of your algorithm.
- Show and discuss your version of the table above.
- Briefly describe what next steps you would try if you had more time to work on the project.

You will need to practice your presentation to make sure it works correctly and fits the time constraint.