Optimization II: TSP Coding Checkpoint #1 Writeup

Neelay Chakravarthy , Ariana Lerena, and Will Cote

Motivation for choosing our approach:

Our group has chosen to begin our approach to the Traveling Salesperson Problem (TSP) by applying a genetic algorithm (GA) modification for the given dataset. We were first motivated to choose to tackle TSP using a genetic algorithm over ant colony as ant colony optimization (ACO) requires larger memory usage through the process of storing the pheromone route. In some initial testing of ACO, shown in the notebook, we found that parallelization of the ants was imperative to good runtime of the algorithm. Additionally, the genetic algorithm boasts a relatively straightforward implementation; we were motivated to leverage this by adjusting its multiple possible population/mutation parameters. We approached GA in TSP by the addition of hyperparameters, with the hopes of achieving a more accurate optimal tour for our TSP.

Modifications made to genetic algorithm in our code:

- **Smart-Crossover**
    - We realized once working on stitching together the tours for different clusters that the tours were not completely valid: that is, they had duplicate cities. This was because the naive-crossover breeding strategy discussed in class is susceptible to gene-duplication. To resolve this is simple: we add the half from a single tour that we wanted, then adding all the genes that haven't already been added to our chromosome after. In terms of TSP, this means checking if a city has been added to the tour already or not.
- **Messiahs**
    - The introduction of messiahs and the two hyperparameters it adds allow us to shift our simple genetic algorithm to be a hybrid between genetics and simulated annealing, with the parameters dictating the strength of the annealing. With that being said, the algorithm does not have a lowering/changing temperature. Rather, we keep track of the best tour from all the generations up until the current point in the algorithm, which we store anyways as that is the answer to TSP we are trying to return. In each generation, we always compare the best tour to the best up until that point in order to update as well. It is here that we make our modification: we additionally check if the difference between the best up until this point and the best in this current generation is greater than messiah_margin, a hyperparameter. If it is, we then replace messiah_percent, our second hyperparameter, of the population with perfect clones of the best up until this point tour, what we call the *messiah*. We then continue the algorithm from there. The effectiveness and results of this change we discuss in the following sections, however note that graphs and metrics other than our final ones do not have smart-crossover implemented. That is, they contain duplicate cities, and the performance of the algorithm on this

corrupted population may be skewed. However, testing with smart-crossover enabled has shown similar improvements over time as before, likely due to the extensive cloning rather than breeding.

Analysis of the current strengths and weaknesses of our model:

- **Strengths:** Our implementation does make gradual and significant improvements. The randomization in each generation runs the risk of picking worse performing tours than the ones we were transferring over from the previous generation. However, utilizing smart-crossover and messiah ideas we were able to generally assure that our algorithm had a low probability of not getting any better in the next generation.
- **Weaknesses:** It was difficult to run the algorithm on maximum length tours so we instead ran it on smaller clusters of cities. The random aspect implemented into our algorithm did also create instances where new generations were not getting any better and in some instances got worse. This was especially the case when the starting population was created using the simulated annealing tour that was found, a much better starting point. The algorithm when faced with this better starting point failed to make significant reductions in length, however dramatically increasing the SA effect by tuning the messiah parameters accordingly resulted in better deltas between generations.

What we hope to add/change and implement in our approach:

- **Variable Messiah Parameters:**
    - One thing we may experiment with is not keeping our messiah_margin and messiah_percent constant throughout generations and between different clusters. Some starting intuition for this is that the greater the messiah_percent, the greater the simulated annealing effect is when it is triggered. Contrastly, the smaller the messiah_margin, the more often the simulated annealing kicks in. Knowing this, there can be some soft-tuning per cluster/generation as to what our margins and percents are for the messiah, i.e. set the margin to some percent of the current stored messiah's tour length.
- **Roulette Selection:**
    - For the selection of the parents used for breeding, we are using truncate_select, where we take the top x% of performers in our population of tours and select them for breeding. Instead, we may look into implementing a different type of selection method, for example roulette, where we increase the likelihood of picking parents with lower tour lengths.
- **Drifting away from the genetic metaheuristic:**
    - In addition to utilizing some of the ideas Matthew brought up in his uncrossing algorithm, we are thinking of drifting away from the genetic algorithm, as what the messiahs in general have shown is that genetics isn't necessarily useful other than making small improvements to a already decent solution (since starting at

worse solutions takes too long to get to what SA can get to in a fraction of the time)
    - We are hoping to implement some version of the genetic algorithm with the results from Matthew's greedy algorithm to try and find and interpret areas of improvement on top of that algorithm.
- **Clustering:**
    - The manner in which we are clustering cities currently has k set to 300, in other words, we are effectively running with 300 clusters of cities with the hopes of running this algorithm on the maximum number of cities. Experimenting with this k value can be very time consuming, however, there may be merit to considering larger/smaller clusters of cities, especially if we can find a good approach to linking the optimal tour from cluster to cluster.
    - To this point, we may experiment with linking these clusters together not from centroid to centroid, but perhaps by linking together the two closest cities from 2 adjacent clusters, and following a possible "nearest neighbor" method.
- **Breeding method:**
    - In our current approach we are using the smart-crossover method suggested by the initial coded solution. There are, however, a series of more nuanced crossover methods specifically designed for GA with possible TSP applications. We could, for example, implement the partially mapped crossover method or the order crossover method. Within combinatorics problems, such as TSP, there is a chance that an offspring contains the same genome twice, or is missing some others. The aforementioned crossover methods implement genetic repair, "e.g. by replacing the redundant genes in positional fidelity for missing ones from the other child genome." By implementing a more nuanced approach to the crossover step within our GA, we may hope to find faster, more accurate optimal tours.
    - Implementing Partially Mapped Crossovers in TSP has special advantages as, "PMX maintains the same order of genes across parents, important when there are disorders that require a sequence of genes' order, like the Travelling salesman problem."

Results:

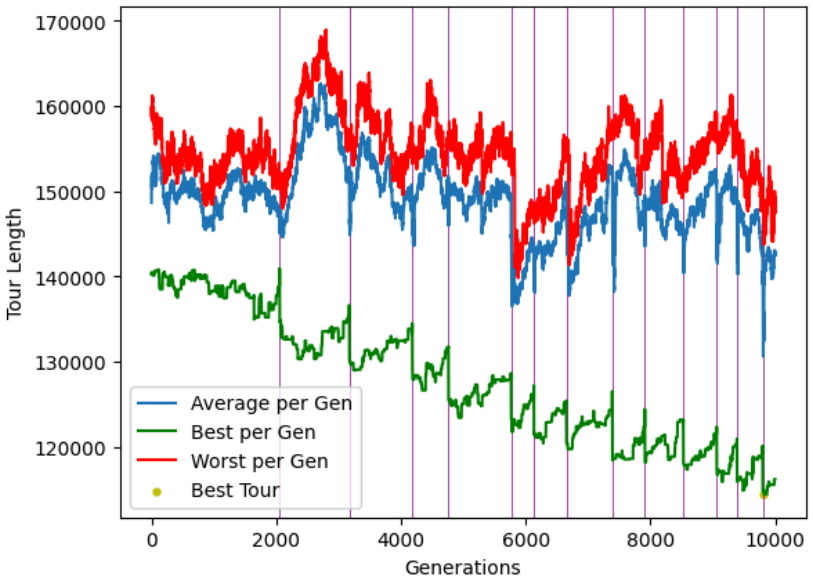
- **Parameter Explanations**:
    - *gens* : number of generations to run
    - *population_size* : size of each population (kept constant throughout generations)
    - *darwin* : top percentage of parents used in truncate select for parent selection
    - *mutation_chance_child* : the chance for a mutation (single random swap) to occur during breeding
    - *elite_clone_percent* : top percentage of parents (multiplied by *darwin*) that are used for cloning (called the *elites*)

- *mutation_chance_elite* : the chance for a mutation (single random swap) to occur during cloning of elites
- *messiah_margin* : the maximum distance margin allowed between the stored messiah and the current generation's best individual. If exceeded, triggers a messiah injection into the population.
- *messiah_percent* : the percent of the population to be replaced by clones of the messiah being injected.

- **Initial Results:**
  - Parameters
    - *gens=10000*
    - *population_size=100*
    - *darwin=0.3*
    - *mutate_chance_child=0.2*
    - *elite_clone_percent=0.1*
    - *mutate_chance_elite=0.05*
    - *messiah_margin=5000*
    - *messiah_percent=0.05*

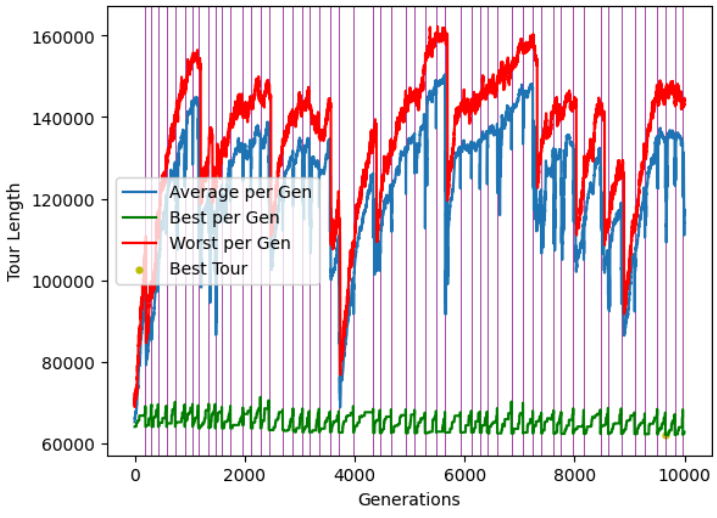
Graph 1

- Here we can see the graph for the parameters above running on a single cluster, with the purple lines denoting messiah injections. (Note, smart crossover was not yet implemented). We see that the best tour was found in generation 9820 with a length of roughly 114.4k. It took 443 seconds.



Graph 2

- Here, we ran the same parameters on a simulated annealing initial population. Very, very little improvement per generation can be seen (though we get better results later). The best tour was found in generation 9654 with a length of roughly 62k. It also took 443 seconds.

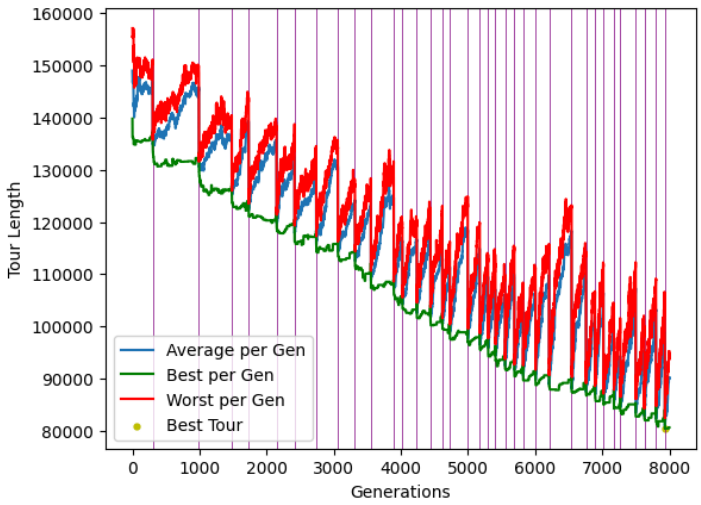


```
gens = 8000 #number of generations
darwin = .30 #top percentage of population used as parents for breeding
mutate_chance_child = .20 #chance for a mutation to occur in each child
mutate_chance_elite = .05 #chance for a mutation to occur in each cloned elite
population_size = 100 #size of each generation
elite_clone_percent = .10/darwin #top percent of parents used for breeding that are also used for cloning (multiplied by darwin percent)
messiah_margin = 2000 #distance margin between best tour found and best tour in current generation (if met, messiah is reintroduced to populat
messiah_percent = .50 #when reintroducing a messiah, the percent of the population that should be reintroduced as clones of that messiah
messiah_num = int(messiah_percent * population_size)
```

We then made the following changes to the parameters

Decreased generations
Decreased messiah margin
Increased messiah num

We had the following results:

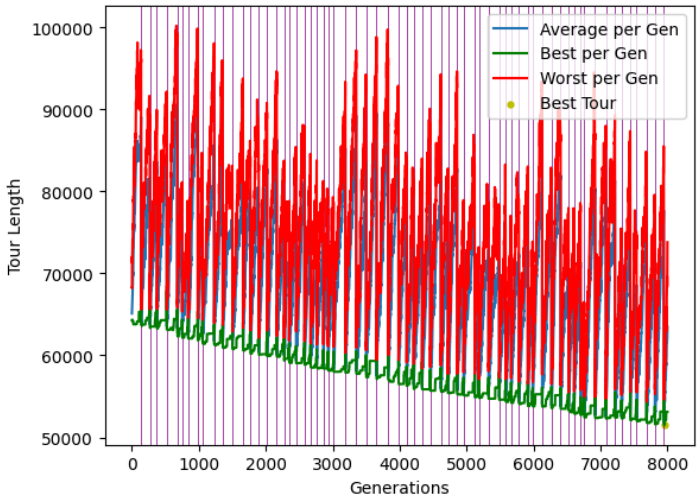


Graph 1:

- Here, we start again with a random population on a small cluster, but this time with a dramatically increased messiah presence. We see greater improvements in even less generations. The best tour was found in generation 7936 with a length of roughly 80.5k It took 567 seconds.

Graph 2:

- Here, we start with all the same conditions, but with a simulated annealing starting population. We see much better delta between generations compared to before. The best tour was found in generation 7949 with a length of roughly 51.5k. It took 570 seconds.

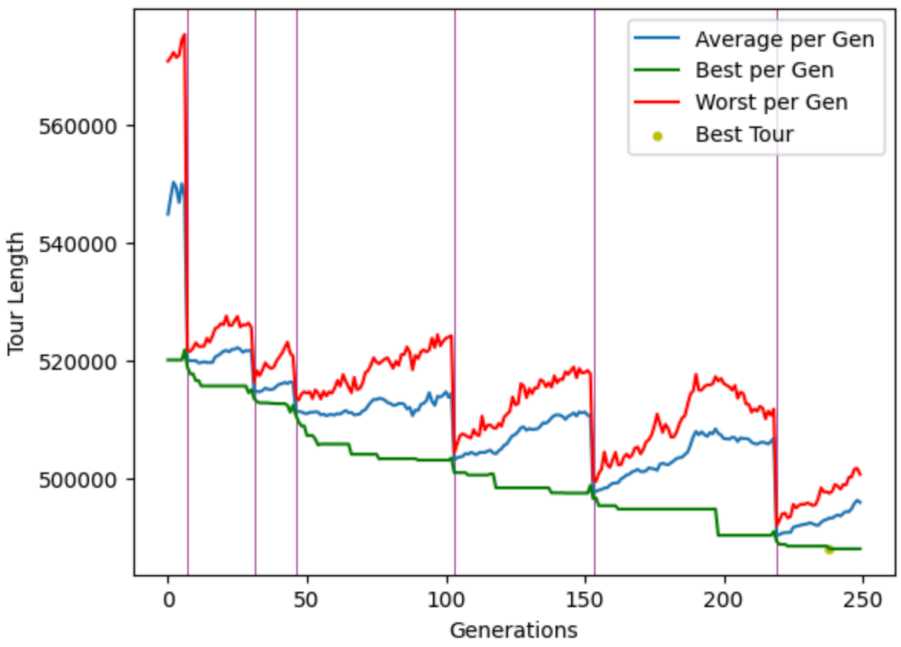


We then ran the algorithm on every cluster, with the goal of simply concatenating the tours together naively for now, with the following parameters (and smart-crossover implemented and enabled):

```
2]: #parameters for running on all clusters
    gens = 250
    darwin = .25
    mutate_chance_child = .25
    mutate_chance_elite = .08
    population_size = 100
    elite_clone_percent = .10/darwin
    messiah_margin = 300
    messiah_percent = .75
    messiah_num = int(messiah_percent * population_size)
```

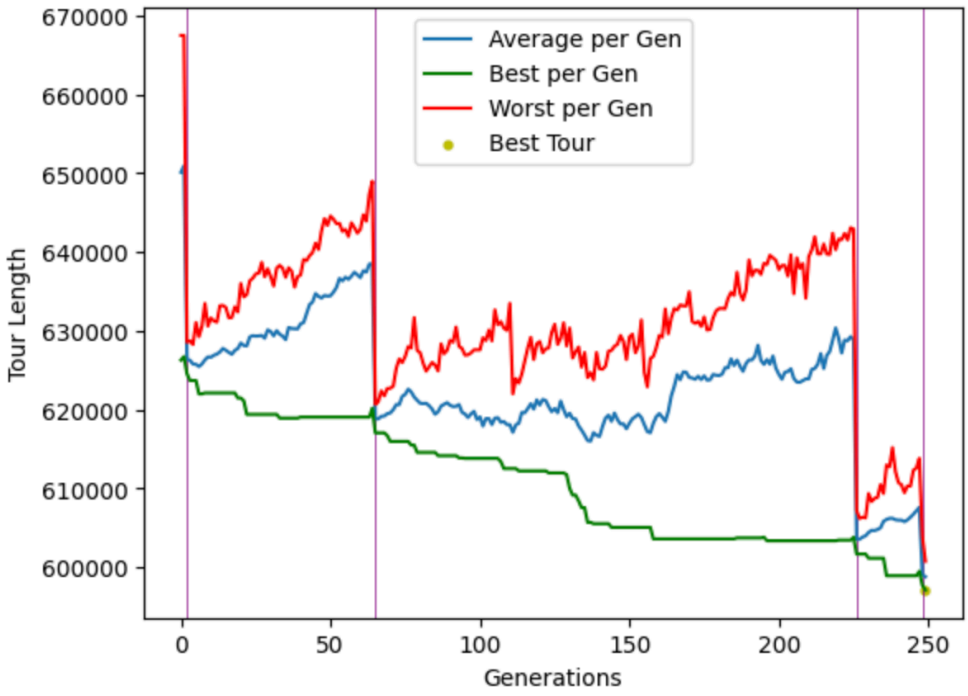
Here were some of the graphs from a couple clusters to give a feel for its performance:

```
54.667248010635376 seconds
```



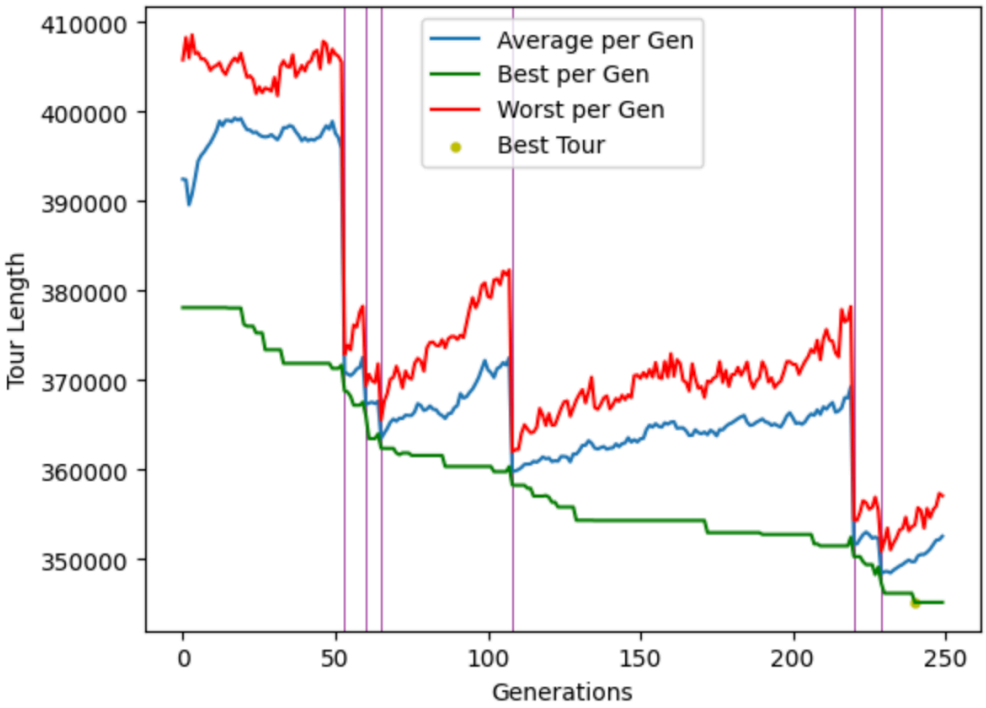

```
Best Tour Found in generation 238 with length of 488035.08558360103
Completed sub-tour for cluster number 66 ----------------------------------------
```

51.62140679359436 seconds



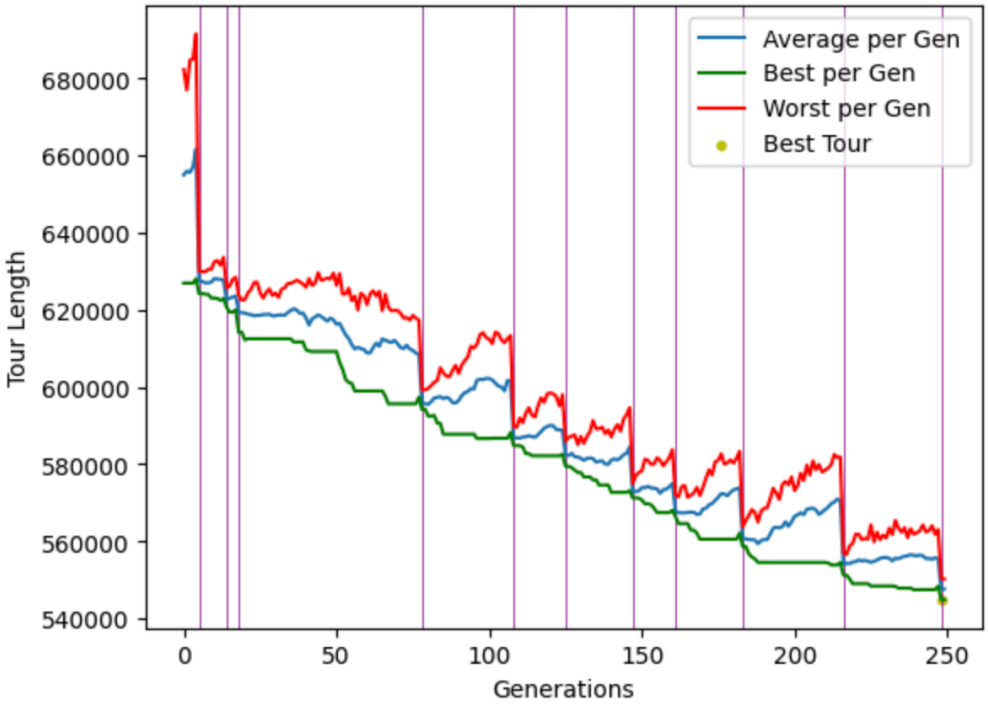
Best Tour Found in generation 249 with length of 597113.0434616617
Completed sub-tour for cluster number 137 ----------

34.31043720245361 seconds



Best Tour Found in generation 240 with length of 345204.2953008356
Completed sub-tour for cluster number 194 ----------

```
43.74957895278931 seconds
```




```
Best Tour Found in generation 248 with length of 544883.5484842244
Completed sub-tour for cluster number 293 ----------------------------------------
```

In the end, to run on the full dataset, it took 7671 seconds, or just over 2 hours (dumb-crossover took 1.5 hours - results not included). The final concatenated tour for the whole dataset had a length of 82017282.85044158 and is displayed on the submitted notebook, along with graphs for every cluster.