Final Project

The code in this final project can be found here.

Feedback and Grading

Not sure if there's anything in particular I want feedback on - I'm pretty proud of the work!

Executive Summary

This project attempts to find the shortest path in a waste disposal system by comparing several methods, namely between truck routing in a fixed path with different optimization (minimize distance, minimize distance to dropoff, minimize distance to fuel, and maximize waste collected) versus truck routing in a flexible path. The findings suggest that the flexible path is superior even when comparing to the best algorithm in fixed-path truck (minimizing distance), with statistical significance (p-value = 0.0) and large effect size of 2.26. It further discusses the scaling behavior of these two implementations for large-scale network, suggesting that the flexible path algorithm while optimal, might not be the best choice in densely-connected graphs.

Model

This project attempts to model a waste disposal system, where a company needs to visit different farms, collect their waste and drop off at a waste site. The truck would follows a road system represented by a network, where each node is a farm, a waste removal company, or a drop-off site, and each edge represent the length of each road. Each farm has varying quantity of waste, centering around a given mean and standard deviation. We want to find the optimal algorithm such that we can minimize the time travelling between farms, thus maximizing the profit. To do that, we simulate a network and apply different types of algorithm and calculation to find the most optimal routing.

Assumptions & Rules

In order to simulate the problem, I simplified a lot of other variables so that we can focus on only one change, which is the algorithm used to decide the route for the truck. A lot of assumptions in this model might not hold in the real world, but suffices for the purpose of the simulation:

1. The network is fully connected, so we can travel to any nodes from any given node. However, it is not complete, so not every node is connected to one another. This is so that we can travel to every place to collect all the waste.
2. The network is a random graph, following the Erdos-Renyi model (with modification that it is fully connected, as per 1). We can use other models for our network, however, road network is more likely to follow Erdos-Renyi model than small-world or power-law distribution network.
3. The edges between each node are also randomly distributed within a defined range. This is like the real world where road connections are within a certain range, but most likely follow a uniform distribution.

4. Each farm produces waste that follows a normal distribution with a mean and standard deviation picked randomly from a given range. The waste fluctuates daily. The truck has access to this mean, but not the actual waste production on the day itself.
5. There is only 1 truck, 1 drop-off site, and 1 fuel site. This is to simplify our model and focus on our variable of interests.
6. The truck always refuel to maximum fuel capacity when at the fuel station, and remove all waste when at the waste dropoff station.
7. The truck spends negligible time at each node, so the only variable of interest would be distance travelled. We simplify this further by assuming that distance travelled = time travelled = fuel used, thus there is only one quantity of interest we need to measure.

Additionally, there are some rules the truck needs to follow:

1. The fuel level must never be below 0.
2. The truck can never carry more waste than the maximum capacity.
3. If the truck goes to a place and they have more waste than it can carry, it can carry up to its maximum capacity, and then leave. The farm would still have the leftover waste.
4. At the end of the day, all wastes from all farms need to be removed.

Variables

With all of the assumptions above, we keep all variables constant, including the network structure, number of trucks, number of dropoff sites, maximum fuel capacity, maximum waste capacity, and so on. Our only quantity of interest is distance travelled. The independent variable that we want to test is the routing. In particular, since this is essentially a travelling salesman problem with complication, we can test two types of truck.

The first type is a fixed-route truck. This truck already has a predetermined route, and follows this route every day. The route is determined by a standard algorithm solving the travelling salesman problem. We vary the algorithm to have 4 different weights: minimizing distance travel, minimizing distance to dropoff, minimizing distance to fuel, and maximize waste collected. We then choose the best algorithm for the fixed-route truck.

The second type is a flexible-route truck. Instead of calculating the route in advance, the route is decided as the truck is traveling: at every node, the truck will greedily choose the closest node to travel to (that is not visited), and repeat until it travels the entire network.

This is essentially comparing a global optimum versus a local greedy algorithm, given the heuristic function for the global optimum function is not known. While we might expect the fixed truck to do better since it's following a global optimum, because our heuristic function is flawed as at any one time, the truck is constrained by several things besides distance (namely waste and fuel), thus the greedy flexible-route truck ends up being better. This is what our simulation found.

Implementation

Parameters

The graph is initialized to have 50 nodes, each of them is, on average, connected to 5 other nodes, and the distance between two nodes are between 1 and 100 km. The farm has a waste capacity range between 100 and 500 kg per day, with a standard deviation range between 1 and 50 kg.

The truck has a fuel capacity to travel maximum of 1000 km, and a waste capacity to carry 3,000 kg of waste.

All of these are reasonable assumptions of the farm systems. Further, note that all of this is defined on the second code cell, and can be changed as needed.

Code structure

Our code has 5 main classes: `Network`, `Truck`, `FixedTruck`, `FlexibleTruck` and `Simulation.`

Class `Network` makes use of the `networkx` library in Python to create the network of interest. Each node in the network has 6 attributes: `mean`, `std` , `is_farm`, `is_dropoff` and `is_fuel_station` and `waste`. The first and second attribute, `mean` and `std` is the parameter for each farm, and is determined by getting a random value from the waste range specified above. If it's a dropoff site or a fuel site, then the `mean` and `std` is 0. The next 3 attributes help check whether the node is a farm, a dropoff site, or a fuel site. Finally, the `waste` value is updated daily so we get a different waste value each day.

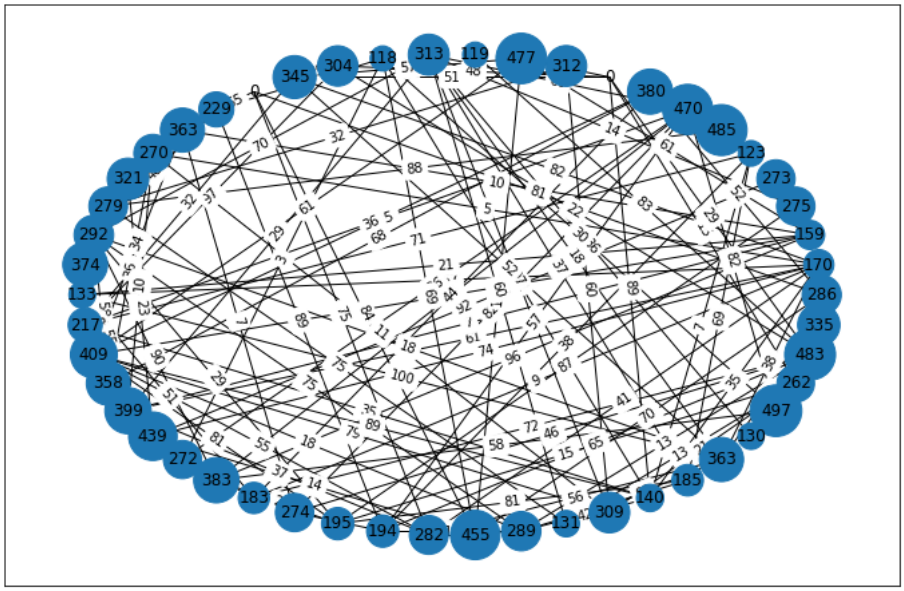

Figure 1: The network created by class `Network`.

Class `Truck` is the parent class for `FlexibleTruck` and `FixedTruck`. It has all the given attributes that both trucks share, such as the network, the fuel capacity, and waste capacity. Additionally, it contains several common methods including `refuel` and `dropoff_waste`. The difference is that `FixedTruck` requires another parameter for initialization, `algorithm`, to create the fixed path that the truck will follow, stored in the `path` attribute. Each time the truck runs, it will follow this path until it reaches the end. Meanwhile, `FlexibleTruck` stores a dictionary of the shortest path between all pairs in the `shortest_path` attribute. When it runs, it has a `visited` set and adds the location it has travelled to this set, continuing until `visited` contains all the farms. It chooses the next location by finding the closest node that we can travel from the current location that is not visited yet.

```
class FixedTruck(Truck):
  def __init__(self, ..., algorithm):
    ...
    self.path = algorithm(graph)

  def run_one_day(self):
    cur_path_idx = 0
    ...
    while cur_path_idx < len(self.path):
      ...
      #choosing next location
      next_loc = self.path[cur_path_idx]

class FlexibleTruck(Truck):
  def __init__(self, ...):
    ...
    self.shortest_path = dict(nx.all_pairs_dijkstra_path_length(self.graph.G))

  def run_one_day(self):
    visited = set([self.dropoff_node, self.fuel_node])

    while len(visited) < len(self.graph.G.nodes()):
      ...
      #greedy algorithm to go to closest node first
      potential_next_locs = [loc for loc in range(len(self.graph.G.nodes()))
                             if (loc not in visited and loc != self.fuel_node
                             and loc != self.dropoff_node and loc != cur_loc)]
      dist, next_loc = min([(self.shortest_path[cur_loc][loc], loc)
                          for loc in potential_next_locs], default = (0, 0))
```

Finally, there's the `Simulation` class which runs the simulation, plot and summarize the simulation.

Algorithm for `FixedTruck`

As discussed above, the `FixedTruck` uses 4 different heuristic function for travelling salesman problem to minimize distance, minimize distance to dropoff, minimize distance to fuel, and maximize waste collected. Fortunately, the `networkx` library already has the algorithm to the travelling salesman problem, so the algorithm only needs to update the `weight` parameter to the function. For minimizing distance, we don't need to use the `weight` since it's already the default.

```
def default_algo(graph):
    ...
    return nx.approximation.traveling_salesman_problem(graph.G)


def minimize_dist_to_fuel(graph):
    ...
    return nx.approximation.traveling_salesman_problem(graph.G, weight = "dist_to_fuel")


def minimize_dist_to_dropoff(graph):
    ...
    return nx.approximation.traveling_salesman_problem(graph.G, weight = "dist_to_dropoff")

def maximize_waste(graph):
    ...
    return nx.approximation.traveling_salesman_problem(graph.G, weight = "waste_neg")
```

Testing

In order to make sure the code is running correctly, I've put a few Error to make sure all our rules are met. In addition, I've used the `logging` library for debugging the `Truck` class to make sure it's working properly.

```
class Truck:
  ...

  def run_one_day(self, ...):
    while ...:
      if self.current_fuel <= 0:
          raise ValueError("Fuel is below 0. Truck is dead.")
      if self.current_waste > self.max_waste:
          raise ValueError("Waste overflow.")
    ...

    for node in self.graph.G.nodes():
        if self.graph.G.nodes[node]["waste"] > 0:
            raise ValueError(f"Waste at {node} not removed")
```

Simulation Results

Best algorithm for fixed-route truck

I first compare the different algorithms used for the fixed-route truck. For each algorithm, I run the simulation 1000 times, which is over 3 years of the truck collecting waste, and then measuring the range of distance travelled (or fuel used), its mean, standard deviation and 95% confidence interval.

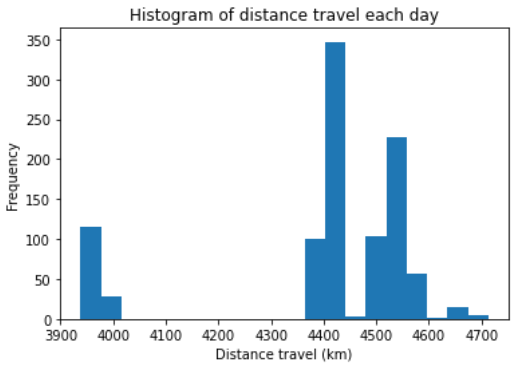

Figure 5: Histogram for fuel used in the default algorithm.

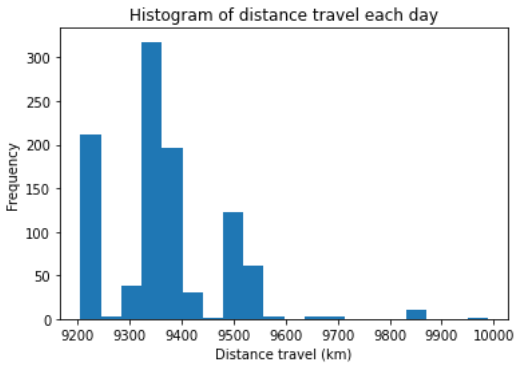

Figure 6: Histogram for fuel used in the algorithm which minimizes distance from the fuel station.

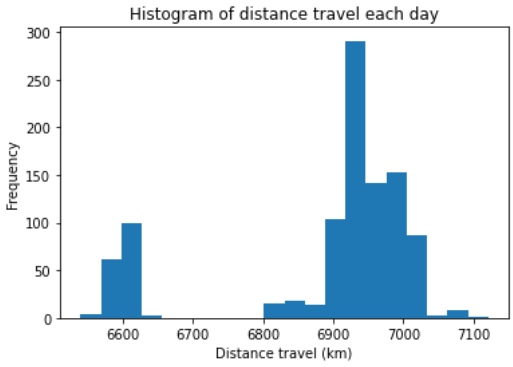

Figure 7: Histogram for fuel used in the algorithm which minimizes distance from the dropoff station.

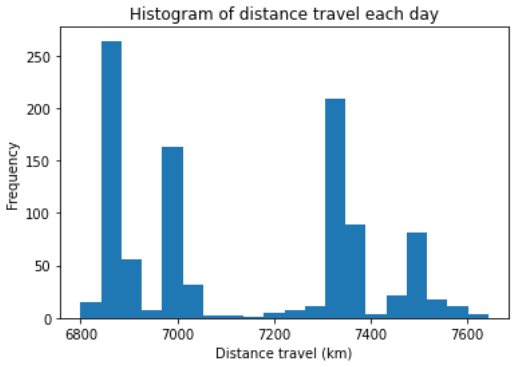

Figure 8: Histogram for fuel used in the algorithm which maximizes the waste from each farm.

	Minimize overall distance	Minimize distance from fuel station	Minimize distance from dropoff station	Maximize waste
Mean	4408.59	9367.95	6895.28	7134.59
Standard deviation	5.88	3.47	4.35	7.70
95% Confidence interval	[4397.04, 4420.713]	[9361.13, 9374.76]	[6886.74, 6903.81]	[7119.48, 7149.70]

From first glance based on the simulation results, we can see that the default algorithm is far superior to all other heuristics. This is because the mean is around 4408.59 km travelled, far lower than all the other heuristics (9367.95, 6895.28, 7134.59 respectively). The standard deviation for all 4 heuristics are quite low, and the 95% confidence interval do not overlap at all, making it very clear that we should use the default heuristic to minimize overall distance for the fixed-route path.

Additionally, even though the histogram is not normal, this is expected because the fuel used is dependent on the path taken, and since the path is the same, we can expect for there to be clusters as the truck would travel the same path and the fluctuation in waste for the farms are small enough that the path travels stay essentially the same.

Comparing fixed-route with flexible-route truck

Next, I compare the fixed versus flexible-route truck using the same parameters for 1000 times again, using the best heuristic for the fixed-route truck.

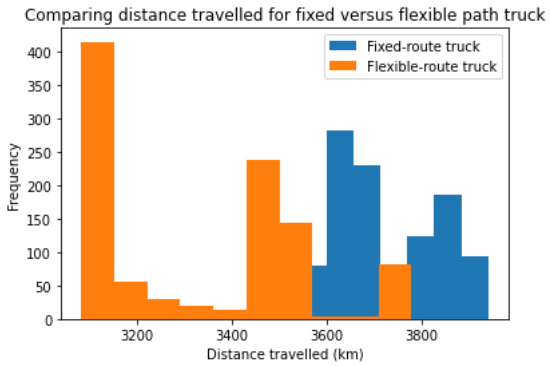

Figure 9: Histogram for fuel used in the fixed-route versus flexible-route truck

	Fixed-route truck	Flexible-route truck
Mean	3726.77	3330.34
Standard deviation	3.52	6.97
95% Confidence interval	[3719.85, 3733.68]	[3316.66, 3344.01]

Based on the histogram and the 95% confidence interval, it seems pretty clear that the flexible-route truck perform better than the fixed-route truck, as its upper bound for the 95% confidence interval is still smaller than the lower bound of the fixed-route truck. The flexible-route truck does have higher variation, which is expected since the route is not fixed and can change each

day. Both standard deviations are relatively small (fluctuation of 7 km for a 3,330-km trip is pretty reasonable), and we can expect that even if we increase the number of steps, the standard deviation would not change as much because there are natural fluctations in waste volumes and distance travelled.

Statistical analysis

To further prove that there's a clear difference between the two types of truck, I calculate the statistical significance and practical significance (i.e. effect size) of the two results.

```
#statistical significance
st.ttest_ind(fuels[0], fuels[1], equal_var = False)
>>> Ttest_indResult(statistic=50.76861487439598, pvalue=0.0)

#effect size
...
cohend(fuels[0], fuels[1])
>>> 2.270441479653127
```

We find that with a t-test for difference in means between two populations, the p-value calculated is 0.0, meaning the probability of this being due to chance is so small that it is completely negligible. This is expected due to the large difference in means and small standard deviation for both trucks. I also calculate the Cohen's D value, which turns out to be 2.2, meaning it is an extremely large effect. Thus, we can safely conclude that the flexible-route truck is superior to fixed-route truck.

Theoretical Analysis

While the flexible-route truck gives better results than the fixed-route truck, it also needs to be scalable. In the simulation, we only use 50 nodes and 5 edges on average, but in a real waste disposal system, the number can be a lot larger. Thus, I also want to calculate how both algorithm will scale in space and time as number of nodes (N)increase.

For the `FixedTruck`, the initialization method uses the Christofides algorithm to approximate the solution path to the travelling salesman problem. The algorithm has the runtime complexity of $O(N^2 \log N)$. For the space complexity, the algorithm guarantees the solutions will be within a factor of 3/2 of the optimal length. However, since we are saving the path in the `path` attribute, we can expect that the path would have the worst case scenario of $O(N^2)$ where for each node, we visit every other node associated with it. This bound can probably be tighter, but there doesn't seem to be a theoretical analysis on it yet.

For the `FlexibleTruck`, the initialization method uses the Dijkstra algorithm to find the shortest path from each node to every other node in the graph. Assuming the algorithm uses a min-heap to compare the distance from the source node to all other nodes, the overall time complexity for finding the shortest path for each node is $O(N + E \log N)$, where E is the total number of edges. E in turn depends on . In our simulation, we assume an average value for edges per node (5), thus E would be a constant times N, and the time complexity would be $O(N + N \log N) = O(N \log N)$. If E is a constant, then the runtime would be $O(N + \log N) = O(N)$, and if the graph is highly connected, would be approximately N^2, then complexity would be $O(N + N^2 \log N) = O(N^2 \log N)$. Note that this is only to find the shortest path for all nodes from 1 source, so we need to repeat this with every other node. So overall time complexity for the initialization method would be from $O(N^2)$ to $O(N^3 \log N)$, depending on the number of edges. For the space complexity, the `shortest_path` attribute needs to store a hash map of all nodes, and in each node the shortest path to all other node, thus it has a space complexity of $O(N^2)$.

For the `run_one_day` method, since the `FixedTruck` algorithm just cycle through the entire path with every other operation being an $O(1)$ operation, the runtime would be bounded by the path length, which is $O(N^2)$. It also requires $O(1)$ additional space. Meanwhile, the `FlexibleTruck` needs to go through all the list of nodes to filter the potential next location, then indexing it in the hash map to find the minimum value, thus for each node, it has $O(N)$ runtime, and since it does that for every node, the runtime is $O(N^2)$. It also needs to store the `visited` set and the `potential_next_locs` value, which is $O(N)$ in space.

	`FixedTruck`	`FlexibleTruck`
Time complexity for initialization	$O(N^2 \log N)$	$O(N^2)$ - $O(N^3 \log N)$
Space complexity for initialization	$O(N^2)$	$O(N^2)$
Time complexity for `run_one_day`	$O(N^2)$	$O(N^2)$
Space complexity for `run_one_day`	$O(1)$	$O(N)$

From the comparison, we can see the disadvantage to the `FlexibleTruck`, especially for the space complexity each time it runs. The time complexity might also be significantly higher, however, it depends on the density of the network (total number of edges). Since we can expect that in the real world, the road network is probably very densely connected, this is a significant disadvantage to the `FlexibleTruck` algorithm. However, if the number of edges is only a scaling factor of N or smaller, then it is

the superior algorithm as the maximum space complexity for both algorithms is `O(N^2)`. Note however, that this is due to the path having `O(N^2)` upper bound, which is probably not the tighest bound.

Conclusion

The flexible-route truck is superior in reducing the distance travel to the fixed-route truck algorithm, even when taking into account the best heuristic among 4 different comparison points. However, the algorithm takes more time and space complexity for initialization and execution, which is especially salient when the network is densely connected (and total number of edges is high). In contrast, if there are fewer edges in the network, the flexible-route is optimal and should be chosen. The tradeoffs can be further calibrated once we know the network topology and the physical constraints of computer hardware.

Acknowledgement

Thanks to Minh for brainstorming and coworking with me, and Hovik for keeping me accountable. I have attempted to referenced all codes I have used from external sources, but might miss a few.

References

Christofides, N. (1976). *Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem*. CARNEGIE-MELLON UNIV PITTSBURGH PA MANAGEMENT SCIENCES RESEARCH GROUP. https://apps.dtic.mil/sti/citations/ADA025602

kelkka. (2022, April 14). *Answer to "How to calculate cohen's d in Python?"* Stack Overflow. https://stackoverflow.com/a/71875070/13808976

Shortest Path Algorithms Tutorials & Notes | Algorithms. (n.d.). HackerEarth. Retrieved April 20, 2022, from https://www.hackerearth.com/practice/algorithms/graphs/shortest-path-algorithms/tutorial/

Stephanie. (2021, September 2). *Cohen's D: Definition, Examples, Formulas*. Statistics How To. https://www.statisticshowto.com/cohens-d/

traveling_salesman_problem—NetworkX 2.8 documentation. (n.d.). Retrieved April 20, 2022, from https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.approximation.traveling_salesma

HCs & LOs

- **cs166-Modeling**: I have clearly explained the rules, variables, assumptions and parameters of the simulation, as well as why they are appropriate given the goal of the simulation.
- **cs166-PythonImplementation**: I have used sophisticated Python methods, libraries and design patterns, including inheritance for the `Truck` parent class, `logging` which is the industry standard for debug, and wrap everything in object-oriented classes.
- **cs166-CodeReadability**: My code is well structured, well commented, with clear functions and variable names. It is also appropriately represented in the report to give a clear understanding even without the needs to look into the code.
- **cs166-EmpiricalAnalysis**: I have run each simulation 1000 times and compare the frequency and mean values. I have additionally carried out statistical analysis to make sure that my findings are significant both statistically and practically.
- **cs166-TheoreticalAnalysis**: I have used the appropriate empirical analysis for algorithm, namely comparing their space and runtime complexity scaling behavior. I have explained the analysis clearly and also state its limitations where there is no theretical findings yet.
- **cs166-Professionalism**: I have written the report in a clear, organized manner, with clear sections and headings, good explanation and presentation of the findings.
- **#significance:** I have used t-test to calculate statistical significance and Cohen's D for practical significance, explaining these results and why they are important to the result findings.
- **#descriptivestats:** I have used accurate measurement for the location and spread of the data using mean and standard deviation, the 95% confidence interval while keeping in mind the histogram accompanied and how these values can be accurately interpreted.
- **#variables:** I have clearly identified the independent and dependent variable in the system, as well as how I isolate them to be the only variable of interest by keeping everything else that might be confounding as a constant. I also accurately measure the impact on the dependent variable.