CS 486 – Assignment #2
Justin Franchetto
20375706

Section I: Written questions

a) Using depth-first search, we would expand the states in the order: *s, h, k, c, a, b, d, m, e, n, g*

b) Manhattan Distance heuristic, h:

i. To get from one state to another, it will always require us to make at least one move, with or without barriers in place. However, trying to move with barriers in place will cause us to take either the same number of moves (if there is no barrier in the way) or more moves (if there is a barrier in the way) than the Manhattan Distance, which means our heuristic always underestimates the true cost and is thus admissible.

ii. Using greedy best-first search with the Manhattan Distance heuristic, we would expand the states in the following order: *s, h, k, c, a, b, d, m, g*

iii. Using A* search with the Manhattan Distance heuristic, we would expand the states in the following order: *s, h, k, f, p, q, r, t, g*. This is assuming that we use the length of the current path as the tiebreak and that we do not expand the same node twice.

Section II: Programming questions

A) Informed Search (A):*

a) We represent the problem using the following:

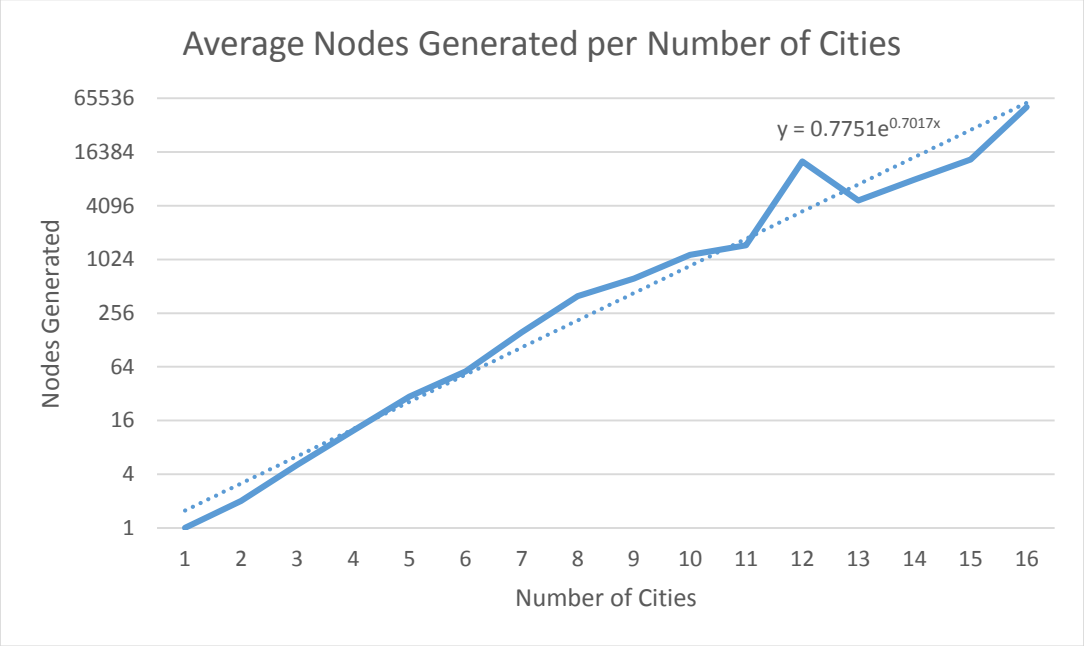
i. **States:** A state is the path we've visited so far, and the cost of that path. This could be represented by a (Cost, Path) tuple.
ii. **Initial State:** The initial state is the tuple (0, [A]), as we start at A and have yet to visit any cities.
iii. **Goal State:** The goal state is a tuple which has a path containing all cities, starting and ending with A, and the cost of that path. There are several goal states, though not all are optimal.
iv. **Operators:** The operation for A* is adding a new city to the path stored in the tuple, resulting in a transition to another state. The cost of this is the Euclidean cost between the new city and the last visited city.

b) The heuristic for this problem is to compute the cost of the Minimum Spanning Tree (using Prim's algorithm) of all unvisited cities, and then add the cost to get back to where we started. To find a solution to the TSP, we will need to visit every city in some order and then return home. The MST naturally gives us a way to visit every city in the way that costs the least, and we can be clever by adding the starting node to the list of unvisited cities so that Prim's automatically finds the minimum cost to get back to the start as well. Since we are always finding the minimum cost, it means we will never overestimate the true cost to traverse all of the cities, and therefore this heuristic is admissible.

c) The average number of nodes generate for each number of cities is as follows:

Cities	1	2	3	4	5	6	7	8
Nodes	1	2	5.1	12.4	29.7	56.8	155.9	397.5

Cities	9	10	11	12	13	14	15	16
Nodes	623.4	1147.6	1476.7	12804.6	4680.9	8052.1	13536.7	52133.7



The trendline gives an estimate of about 70 billion ($0.7751*e^{0.7017(36)}$) nodes generated for an instance with 36 cities. Given that it takes about 15 seconds to process 175,000 nodes, it should take around 72 days to process the 70 billion nodes needed to complete the *Problem36* instance.

B) Local Search (Simulated Annealing):

Note: For this question, an iteration refers to one attempted move, regardless of if it was accepted by the probability function or not.

a. There are a few possible local search operators for simulated annealing. All of them preserve the tour because they do not add or remove any cities, and thus result in the creation of a new tour.

i. **2/3-Random:** 2/3 Random cities on the tour are chosen and their order is swapped.
ii. **2/3-Adjacent:** 2/3 adjacent cities on the tour are chosen and swapped.
iii. **2-Optimal:** Two cities (c_i, c_j) in the tour are chosen at random, and their immediate neighbors are found (c_{i+1}, c_{j+1}).The cities are moved such that c_j and c_{i+1} switch places, effectively taking away two edges and creating two new edges.

After experimenting with different operators, it appeared that 2-Random performed best and resulted in optimal or near-optimal solutions, and is thus used in my implementation of Simulated Annealing. However, given sufficient time, the 2-Adjacent may outperform the random swap as it makes smaller changes resulting in a more steady approach to the goal state.

b. Three different annealing schedules were examined, and are outlined below:

i. **Linear:** The temperature is reduced by a fixed amount (0.05) after every iteration
ii. **Fitzpatrick:** The temperature is reduced by multiplying by a fixed cooling rate

iii. **Fitzpatrick + Iteration Constraint:** The temperature is reduced by multiplying by a fixed cooling rate, but is required to run for a set number of iterations (5) before the temperature is changed.

The three annealing schedules were testing by running three different problem instances, all of size 16. The final solution costs and runtimes are as follows:

(*Starting Temperature = 15000, Cooling Rate = 0.999985, Ending Temperature = 1)*

Instance(Optimal cost)	Linear(0.05)	Fitzpatrick	Fitzpatrick + Iteration
16: #1 (404.02)	(440.04, 9.4s)	(404.02, 20.3s)	(404.02, 98.0s)
16: #6 (346.51)	(406.24, 9.5s)	(346.51, 19.6s)	(346.52, 98.2s)
16: #10 (330.51)	(392.61, 9.4s)	(330.51, 19.8s)	(330.51, 99.1s)

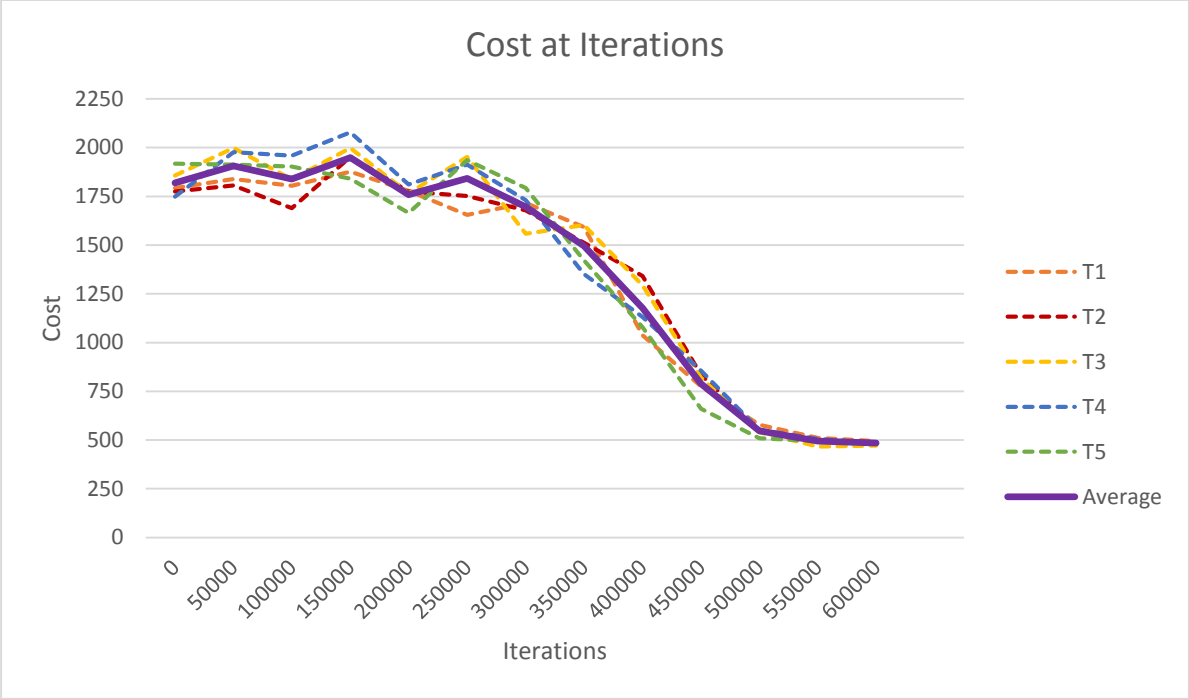
The solutions found by the linear schedule are too far from the optimal solution, and the introduction of an iteration constraint causes the algorithm to run for longer than needed (though, this schedule may see improved solutions for problems with more cities). The Fitzpatrick schedule will be chosen for the Travelling Salesman Problem because it provides accuracy in reasonable amounts of time.

c. In order to more accurately reflect how the cost of the solution to *Problem36* changes over the course of the Simulated Annealing algorithm, five trials were run on the 36 city instance with the following results:

(*Starting Temperature = 15000, Cooling Rate = 0.999985, Ending Temperature = 1, Fitzpatrick Schedule)*

Iterations	0	50000	100000	150000	200000	250000	300000
T1 Cost	1793.244	1839.331	1804.288	1875.968	1778.89	1655.857	1715.761
T2 Cost	1775.274	1805.845	1689.949	1952.137	1774.313	1751.203	1678.373
T3 Cost	1856.821	1998.299	1837.093	1997.462	1765.173	1952.032	1558.491
T4 Cost	1748.956	1976.073	1957.991	2078.095	1811.211	1913.275	1730.76
T5 Cost	1917.811	1913.428	1902.348	1840.776	1664.822	1935.159	1793.496
Average	1818.421	1906.595	1838.334	1948.888	1758.882	1841.505	1695.376

Iterations	350000	400000	450000	500000	550000	600000	**Best**
T1 Cost	1592.316	1038.474	775.3224	578.8696	510.5559	494.221	490.4574
T2 Cost	1511.775	1342.347	834.247	551.5243	496.2134	488.9769	482.7297
T3 Cost	1602.532	1293.841	819.1003	540.088	467.1115	471.7481	465.7625
T4 Cost	1352.14	1130.848	855.8655	552.2554	506.1912	483.1828	483.1828
T5 Cost	1422.354	1078.88	660.2157	508.5305	494.6654	488.699	480.8829
Average	1496.223	1176.878	788.9502	546.2536	494.9475	485.3656	480.6031



From the plot above, we can see how the cost of the solution changes throughout the course of the algorithm. As the number of iterations increases, the cost of the solution rapidly decreases before leveling out near the optimal cost, which is 464. Using the Fitzpatrick temperature schedule, the average final solution cost was 480.60 (approximately 3.5% error) and the lowest cost was 465.76, running with an average time of 32.44 seconds.

d. Simulated annealing is a complete algorithm, because we start with an initial random solution (regardless of how bad the solution is, it is valid) and all of the operators preserve the tour, meaning that we will always finish with a solution.

e. Simulated annealing is optimal, but only if a sufficiently good temperature function is used. Otherwise, the final solution can be near-optimal for some instances, or very poor for others. If given sufficient time to cool, the algorithm will settle on the optimal tour (global minimum); if it is cooled too quickly it will not have made enough moves to get away from the initial random solution, resulting is a poor solution (local minimum).