Assignment 3

Question 1

(1) Tabu Search Setup

A tabu search algorithm was developed to solve the quadratic assignment problem (QAP). The program first parses the flow and distance data, then initializes the following: a candidates list via a priority queue of pairs of costs and department layouts, and a tabu list via a map of department layouts and tabu tenures. The stopping criterion for the program is 100,000 iterations, and for each iteration, it performs the tabu search. It randomly selects five candidate solutions from the entire neighbourhood and evaluates each layout to determine associated costs. The costs are computed by multiplying the appropriate flows and distances for each department based on their positions in the layout. These costs and layouts are then pushed to the priority queue, which automatically sorts by lowest cost from the top. Next, the tabu list is updated, where all candidates in the tabu list have their tenures reduced by one, and if a tenure reaches zero, that candidate is removed. Then, any candidate solutions that appear in the tabu list are discarded, since tabu search skips previously visited solutions in the list to escape local optima. Now, the candidate at the top of the priority queue is the next best solution, so add this to the tabu list with a tabu tenure of five. Lastly, the program checks if this solution is better than the current best solution, and if so, replace the best layout and best cost solutions with this one. The costs and candidate list are reset, and the loop repeats until the termination criteria is met.

(2) Tabu Search Results

The program was run various times with 100,000 iterations, a candidate list of size 5, and a tabu tenure of 5. The initial solution was random every time, the neighbourhood was the entire neighbourhood, and no aspiration criteria was used. A sample result is shown below:

Layout					**Cost**
17	4	20	7	6	2,792
19	15	8	5	3	
2	11	16	1	13	
14	18	12	10	9	

Paolo Torres

The obtained results varied from as low as 2,570 to as high as 3,172, depicted in the next section on the tabu search modifications. Overall, the results appear to be reasonable and relatively close to the optimal solution of 2,570. The optimal solution was achieved once, shown in the next section when the neighbourhood search strategy was modified. This problem is NP-hard, and as such, does not have a polynomial time solution. Since there are 20 locations and the problem is encoded as a permutation setup, the total amount of solutions is 20 factorial. This amounts to a solution space of an order of magnitude of 10^{18}. Thus, with such a large solution space, one could only achieve the optimal solution with a different candidate selection strategy or modifying the tabu setup parameters such as the termination criteria or tabu list size.

(3) Tabu Search Modifications

The initial starting point was randomized and the tabu search was executed ten times:

Run	1	2	3	4	5	6	7	8	9	10
Cost	2,794	2,908	2,878	2,858	2,850	2,830	2,880	2,870	2,850	2,836

The tabu list size was changed twice: once to be smaller and once to be larger:

Smaller	3,090	3,076	3,146	3,062	3,172	3,074	3,026	3,046	3,148	3,012
Larger	2,676	2,668	2,688	2,710	2,670	2,684	2,599	2,680	2,676	2,706

The tabu list size was changed to be a dynamic one, chosen randomly between a range:

Cost	2,654	2,678	2,688	2,610	2,680	2,680	2,670	2,610	2,708	2,654

Two aspiration criteria were added, as stated in the question, for two experiments:

Best Solution So Far	2,684	2,680
Best Solution in Neighbourhood	3,102	3,000

Only a section of the neighbourhood was used rather than the whole neighbourhood:

Cost	2,588	2,666	2,570	2,576	2,618	2,636	2,584	2,592	2,646	2,602

Lastly, a frequency based tabu list was added to encourage the search to diversify:

Cost	3,070	3,052	3,050	3,066	3,078	3,106	3,094	3,056	3,050	3,090

Analyzing all the modifications above, the trend for my tabu search appears to be that all intensification techniques improve the solution, whereas diversification techniques worsen it. In particular, increasing the size of the tabu list, using a dynamic tabu list size, and only using a section of the neighbourhood as opposed to the whole neighbourhood, all resulted in improved cost values. As shown, the initial cost range of roughly $[2{,}700, 2{,}900]$ improved to around $[2{,}570, 2{,}700]$. Conversely, decreasing the size of the tabu list and utilizing a frequency based tabu list led to increased cost values. The cost from the initial range of $[2{,}700, 2{,}900]$ worsened to roughly $[2{,}900, 3{,}200]$. The outlying result is the aspiration criteria for choosing the best solution in the neighbourhood, as this is an intensification technique, however, it significantly increased the cost.

As mentioned, the behaviour of my tabu search appears to be that intensifying improves it while diversifying worsens it. Using the domain and scope of the problem, this is attributed to the fact that the problem has such a large solution space. Even with $100{,}000$ iterations, it is only able to reach the most optimal solution of $2{,}570$ one time, however, it does get close many times, with the best solutions being in the $[2{,}570, 2{,}700]$ range. To further emphasize, the solution space is 20 factorial, which results in a massive search space, and with only a limited number of iterations, it would take many attempts before eventually attaining the most optimal solution. As such, intensification seems to help my tabu search because it allows my program to exploit small portions of the search space, which is necessary for a problem of this size. It is almost as if the program needs to first get lucky to attain a favourable arrangement, and then focus in on that, only making minor modifications to reach the optimal solution. The solution space is too large for diversification relative to the number of iterations, so it is desirable to initially reach a suitable arrangement, and then home in on that specific setup.

Assignment 3

Question 2

(a) Representation

The solutions utilize a two-dimensional matrix data structure, where the rows correspond to each population, and the columns represent ISE, t_r, t_s, and M_p, respectively. To ensure maximum precision of the final solutions, the output values carry over all of the decimal places as returned by the program. Significantly large solutions, as well as solutions that result in zero actual counts, are removed from the population, since they are clearly non-optimal. The optimal input values are represented as an array of three elements, holding the optimal K_p, T_i, and T_d values with all decimal places carried over. The optimal fitness value is a single number, which gets updated whenever the current generation's optimal value is better. Finally, the optimal output values are represented as an array of four elements, holding the optimal ISE, t_r, t_s, and M_p values with all decimal places carried over.

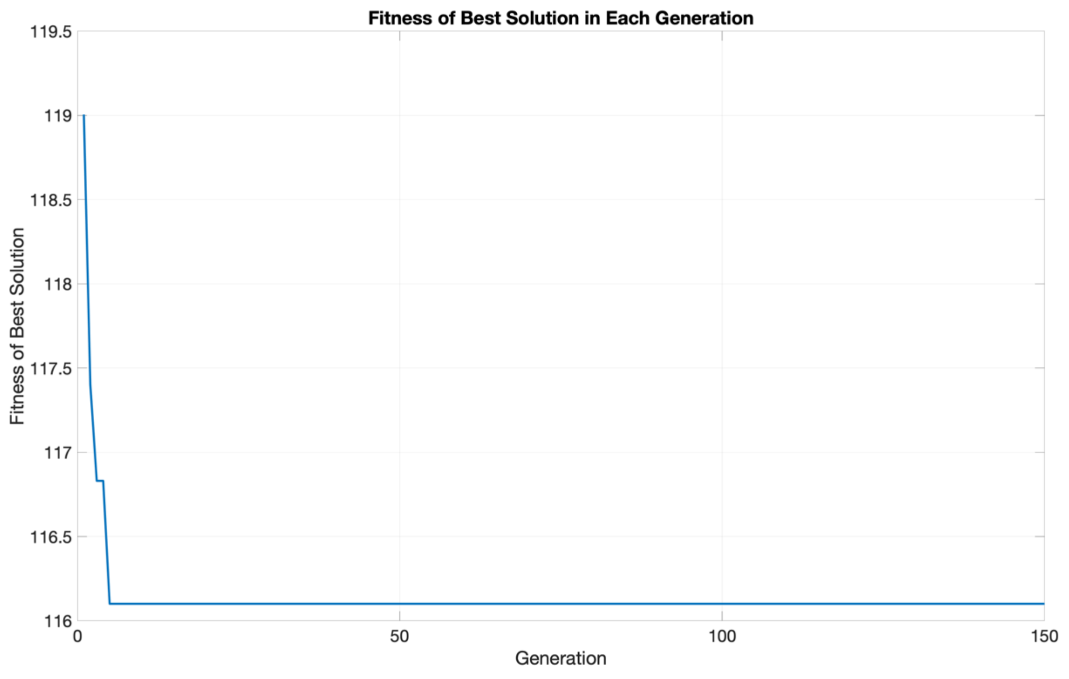
(b) Fitness Function Formulation

The fitness function used to evaluate the solutions was an additive one, where the four outputs, namely the integral squared error ISE, the rise time t_r, the settling time t_s, and the maximum overshoot magnitude M_p, were added together. This was done for every set of input parameters, namely K_p, T_i, and T_d, for the size of the population. For each computation, a condition was added to check if this fitness value was better than the current best fitness value, and if so, record the set of inputs, its fitness value, and set of outputs that correspond to it. In addition, the total fitness sum and fitness average for each generation was computed to determine the associated probabilities, expected counts, and actual counts. This was done for the number of required generations of 150, where the fitness function would evaluate the new solutions introduced for subsequent generations. This allows the program to determine the most optimal set of inputs, fitness value, and set of outputs it can get by the end of the last generation.

(c) Genetic Algorithm Implementation

The fitness function above was used to calculate the fitness values for each set of inputs for every generation. Some solutions returned significantly bad results, such as having high integral squared errors and maximum overshoot magnitudes, so these were removed as they would not be optimal solutions. Similarly, any other entries that correspond to an actual count of zero were removed, as they are not optimal enough compared to the other solutions. Any entries with an actual count of greater than one were copied that amount of times, since these are more promising solutions, so that later on, different crossover and mutation techniques can be applied. The following crossover operator was used: compute positive and negative deltas for each input parameter. If the deltas result in going out of range, clamp them to their respective limits. The deltas used are 1 for K_p, 0.1 for T_i, and 0.01 for T_d. Calculate the outputs that would result from applying the positive and negative delta values for each parameter and apply the delta that would more optimize the solution. This strategy allows crossover to tend towards changes that minimize the output, which is the desired result. The mutation operator used is similar, except rather than changing all the parameters, it randomly selects one of the three parameters to change. Just like crossover, it simulates applying the two delta values to the selected parameter and chooses the one that more optimizes the solution. This is done for every generation until the final optimal solution is achieved.

(d) Fitness of Best Solutions Across Generations



Analyzing the plot of best fitness values for each generation, the first generation starts at a fitness value of approximately 119. At generation two, the value drops to about 117.4, depicting an improvement from generation one, since the goal is to minimize the output parameters. The value decreases again to about 116.83 at generation three, and stays there at generation four, further improving in value. At generation five, it drops to 116.1, and stays there until the end of the algorithm at generation 150. Overall, the algorithm performed significantly well, as it converged to this optimal fitness value in only five generations. The final optimal input values, fitness value, and output values are as follows:

$$K_p = 5.2537, T_i = 6.1768, \text{ and } T_d = 2.3013$$

$$Fitness = 116.1$$

$$ISE = 82.8010, t_r = 0.9827, t_s = 16.8464, \text{ and } M_p = 15.4705$$

(e) Modifying Number of Generations

Experiments were done with two different values for the number of generations:

Number of Generations	10	150	200
Fitness	117.34	116.10	115.95

The results show that as the number of generations increases, the optimal fitness value gets better. This makes sense because having more generations allows the program to generate more solutions that could be more optimal. More crossover and mutation instances occur, resulting in evaluating more solutions with the fitness function to find better final values.

(f) Modifying Population Size

Experiments were done with two other population sizes:

Population Size	10	50	80
Fitness	123.71	116.10	115.51

The results show that as the population size increases, the optimal fitness value gets better. Similar to increasing the number of generations, this allows the program to evaluate more solutions and thus, have the chance at arriving at more optimal solutions. The trade-off for increasing the population size is an increase in the time and space usage, so although increasing the population size appears to be an obvious positive, one must consider the program's time and space constraints and requirements.

Assignment 3

(g) Modifying Crossover Probability

Experiments were done with two different values for the crossover probabilities:

Crossover Probability	0.4	0.6	0.8
Fitness	115.75	116.10	116.51

The results show that as the crossover probability increases, the optimal fitness value becomes worse. This implies that focusing on exploring and making big jumps to different areas of the search space, hinders its ability to find the optimal solution. This could be because of how crossover is implemented, where all parameters get modified, which could lead it to stray away from more optimal solutions it was close to.

(h) Modifying Mutation Probability

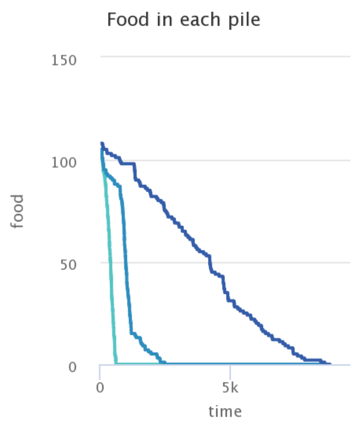
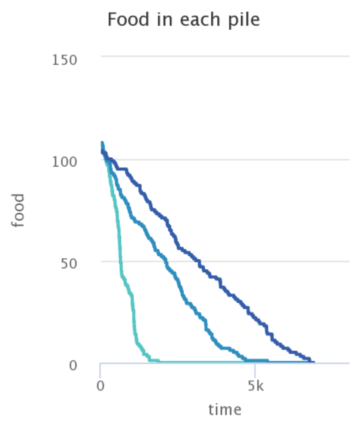
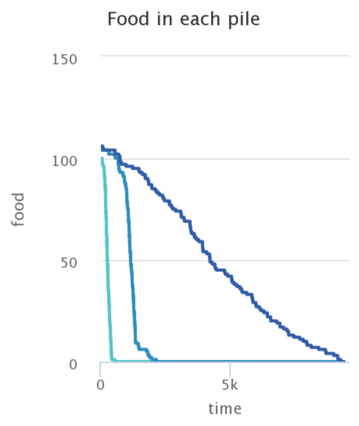
Experiments were done with two different values for the mutation probabilities:

Mutation Probability	0.1	0.25	0.5
Fitness	116.60	116.10	115.79

The results show that as the mutation probability increases, the optimal fitness value gets better. This implies that focusing on exploiting and staying near the parent by only creating random small diversions, helps its ability to find the optimal solution. This is most likely due to how the mutation operator works, where it only modifies one of the parameters, and in a way that more optimizes the solution, keeping it close to what it was based on.

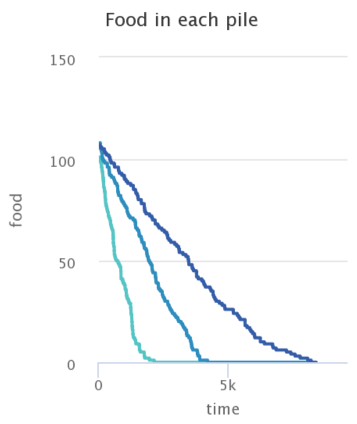
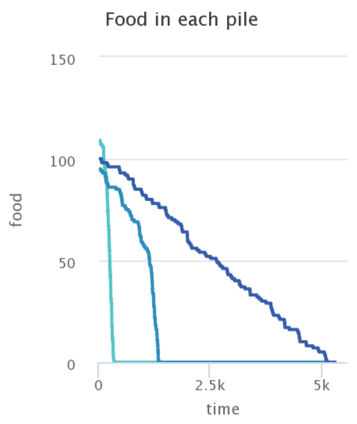
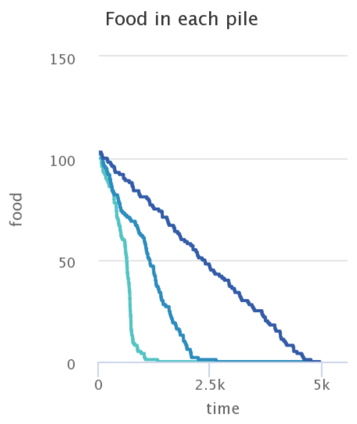
Assignment 3

Question 3

Results	Parameters		
	Population	Diffusion Rate	Evaporation Rate
	30	40	10
	30	40	20
	30	80	10

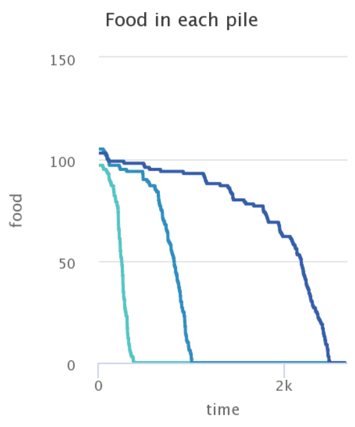
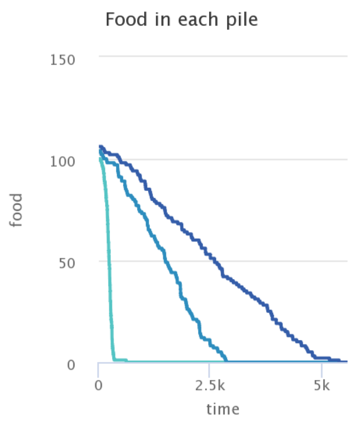
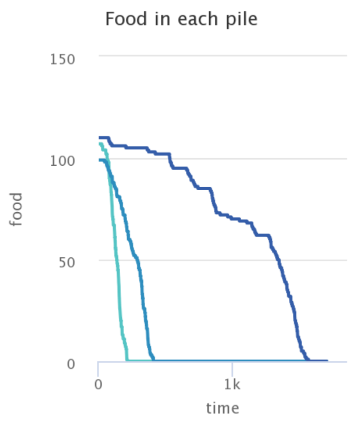
Paolo Torres

Assignment 3

	30	80	20
	50	40	10
	50	40	20

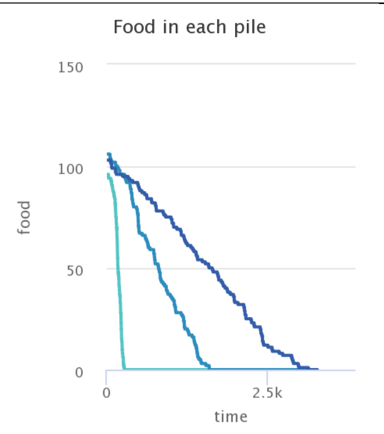
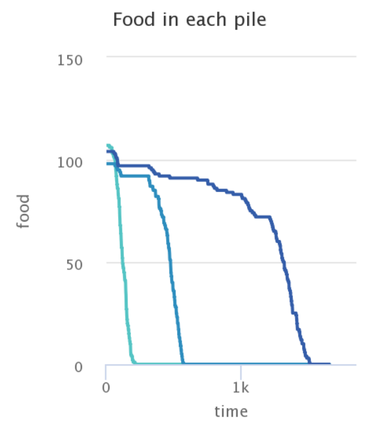
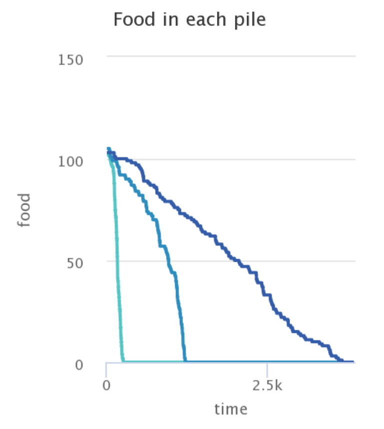
Paolo Torres

Assignment 3

	50	80	10
	50	80	20
	100	40	10

Paolo Torres

Assignment 3

	100	40	20
	100	80	10
	100	80	20

Comparing the first two experiments, where only the evaporation rate changes, the increase in the evaporation rate caused the first two food forages to take much longer due to the pheromone decay parameter decreasing more quickly, thus giving the ants less trails to follow. Comparing the first and third experiments, where only the diffusion rate changes, there appears to be no significant differences, most likely due to having a lower population, where a diffusion

Paolo Torres

rate increase would have less impact. Comparing the third and fourth experiments, where again only the evaporation rate changes, a similar effect occurs, where the first two forages take longer to complete due to having lesser trails to follow.

The trend described above applies to both the 50 and 100 population experiments as well. The biggest difference between the population differing experiments is the time they take to complete. For the 30 population experiments, the final forages complete in roughly 7k to 8k in time for the 20 evaporation rate cases, and around 9k to 10k in time for the 10 evaporation rate cases. For the 50 population experiments, the 20 evaporation rate cases, as well as the 40-diffusion rate and 10 evaporation rate case, finish at about 5k in time, whereas the 80-diffusion rate and 10 evaporation rate case finish at about half the time at 2.5k. Lastly, for the 100 population experiments, the 80-diffusion rate and 20 evaporation rate case finishes at around 4k time, 40-diffusion rate and 20 evaporation rate finishes at 3k in time, and finally the two 10 evaporation rate cases finishing at around 2k in time.

Overall, it appears the amount of population has the largest impact on the completion time for the ants. The higher the population, the lower amount of time it takes to finish. This makes sense because with more search agents available, it allows the colony to cover more of the search space and hence, have a higher chance at finding food. In addition, having more ants results in more pheromone trails, thereby attracting more of the colony towards the general area of the food. This also explains why there appears to be a faster exponential decay with higher populations, as more pheromone trails results in more exploitation at those desirable locations in the search space.

Although the trend appears to be, the higher the population the better, in practicality, there are limitations. Namely, the size of the population would be constrained by specific space limitations such as memory of the program or computer. Moreover, the results show that the evaporation rate has a considerable effect on the completion time. In particular, when keeping population and diffusion rate consistent, an increase in the evaporation rate increases the overall time to finish. This makes sense because as the evaporation rate increases, the pheromone trails disappear at a faster rate, thereby giving the ants less chances at finding desirable areas. Conversely, the diffusion rate appears to have much less of an impact on the completion time, signifying that the presence of pheromone trails is more important than its movement.