Name: Qingsong Tan User-name: grkp39

Algorithm A: Genetic Algorithm

Algorithm B: Ant Colony Optimization Algorithm

Description of enhancement of Algorithm A:

Description of enhancement of Algorithm B:

*A basic implementation of the ACO algorithm:*

1. *The code begins with a uniform setting of the parameters.*
2. *Create a pheromone\_matrix and initialize the pheromone. Then initialize an optimal path.*
3. *Randomly selecting a city as a starting point, thus achieving the effect of randomly placing ants. (i.e. function: probability\_of\_selection)*

*Main loop:*

1. *In the main loop: Then an iterative loop is started to build the path of each ant by a for ant in range(num\_ants) loop. Then continue to construct city-to-city routes and find the shortest paths and update them by for\_in range(num\_cities). Also according to the structure of the ACO algorithm of the lecture after each iteration the pheromone needs to be stored and evaporated, i.e. updated. Also enhance the pheromone according to the quality of the paths.*
2. *The final output prints the optimal solution.*

*An enhanced implementation of the ACO algorithm:*

*Heuristic function: the attractiveness function is introduced to directly calculate the attractiveness between cities, integrating the effects of pheromones and distance. More dynamic city selection: roulette selection based on calculated attractiveness is implemented with the select\_next\_city function, making city selection more randomized and efficient. Visit logging: the visited cities are tracked using the collection visited, ensuring that ants do not visit cities repeatedly. Pheromone update strategy: pheromones are added only when the best path is found, rather than for all paths. This helps to speed up convergence and avoid precocious local optima.Heuristic function: the attractiveness function is introduced to directly calculate the attractiveness between cities, integrating the effects of pheromones and distance. More dynamic city selection: roulette selection based on calculated attractiveness is implemented with the select\_next\_city function, making city selection more randomized and efficient. Visit logging: the visited cities are tracked using the collection visited, ensuring that ants do not visit cities repeatedly. Pheromone update strategy: pheromones are added only when the best path is found, rather than for all paths. This helps to speed up convergence and avoid precocious local optima.*

*A basic implementation of the GA algorithm:*

1. *According to the function generate\_initial\_population function, generate the initial population.*
2. *The tour\_length function is then used to calculate the fitness of a given travelling path used to evaluate the path.*
3. *select\_parents is a selection function that I used the roulette wheel selection method taught in the lecture to select parents from the current population for the next generation.*
4. *Then the crossover and mutate functions are used to construct the crossover and mutate processes respectively, i.e. used to randomly generate new individuals.*

*Crossover: using the order crossover method which The number of occurrences of the city is kept to exactly one, thus avoiding the generation of illegal solutions.*

*Mutate: This is a basic swap mutation. For a given tour, it iterates over each element and decides with a small probability (specified by the mutation\_rate parameter) whether to mutate or not.*

1. *When all the functions are completed is to start integrating them into the genetic\_algorithm functions to implement a basic genetic algorithm to solve the TSP problem.*

*An enhanced implementation of the GA algorithm:*

*Mainly improved code for selection and crossover mutation, in trying to improve using tournament selection, PMX crossover method and inversion mutation. Tournament selection is simpler and faster to implement than roulette selection, as it quickly determines the winner through a fixed-size "bidding match". PMX guarantees legitimacy and values the genetic structure of the parent through a mapping mechanism, suitable for scenarios with complex element positional relationships. Therefore, it can obtain better search performance and quality of the solution. Inverted mutation decides with a certain probability whether or not to make a mutation. Compared to the exchange mutation, the inversion mutation may have a greater impact on the performance of the solution because it is able to change the structure of the solution more fundamentally by changing the internal order of a sub-sequence.*

***DESCRIPTION OF ALGORITHM ONLY IF THE ALGORITHM IS NOT COVERED IN LECTURES***

Description of *non-standard* Algorithm A:

*Describe any non-standard algorithms you have implemented that* ***have not been covered in lectures*** *(otherwise these boxes should be blank) You need to convince me that your implementation is indeed that of the named algorithm and you need to* ***provide a full reference to the source for your algorithm****. You should* ***include a pseudocode description****. You can vary the sizes of these boxes but do not change the font (Calabri), font size (11), the paragraph properties (single space) or the header and footer and everything should fit onto one side of A4. (You can delete these instructions.)*

***Remember: You need my express permission to implement a non-standard algorithm!***

Description of *non-standard* Algorithm B: