Result Analysis (Conclusions):

OPTIMALITY ANALYSIS :

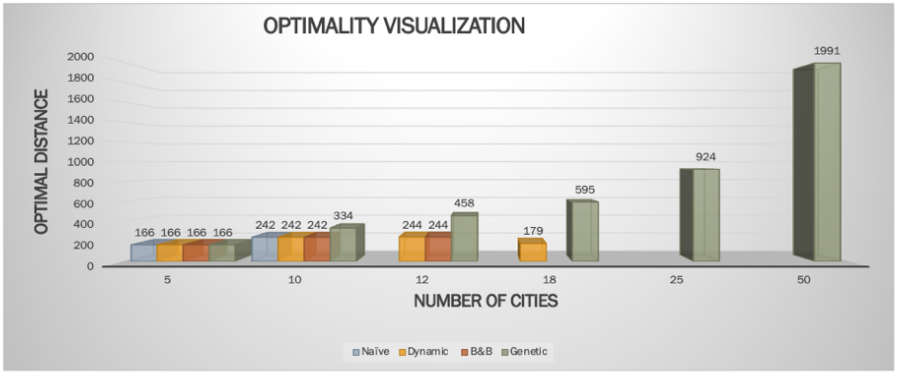


Fig-2

The first comparison is done for checking the optimality of the 4 algorithms implemented. The X axis represents the number of cities while the Y axis represents the optimal distance. As clearly seen above, the 3 methods naïve, dynamic and B&B (marked with blue, yellow, and orange respectively) always give the best solution possible. A similar trend is observed for genetic algorithm (marked with green) when we set the number of cities to 5. However, as we increase the number of cities to 10 and more, we can easily see that the bar for genetic algorithm starts increasing. This is justifiable as the genetic algorithm is a heuristic approach and it values speed over precision. The gap increases as we take a greater number of cities. Around 12 nodes, the naïve approach stops working as expected as it has a factorial complexity and will take a long time to compute the path. Similarly at around 18, even the branch and bound method stops working, leaving only dynamic and genetic as competitors. At the 25 mark it is clearly seen that only the genetic approach remains. Though the values may be quite distant to the optimal solution, it is the only approach out of the 4 that works for many cities (even 500+!!) and will give us a way better solution that a random guess, because the genetic algorithm can make an educated guess.

Result:

From our research on the optimality analysis, we can conclude that for smaller number of cities (in the range of 1-20) the 3 methods, namely- Naïve, B&B and Dynamic approaches perform the best, giving the most optimal solution possible for each case. However, as the threshold on the number of cities exceeds 20, genetic approach becomes the clear winner as the other algorithms completely stop working due to a large time complexity. This can be clearly seen from the graph.

TIME ANALYSIS:

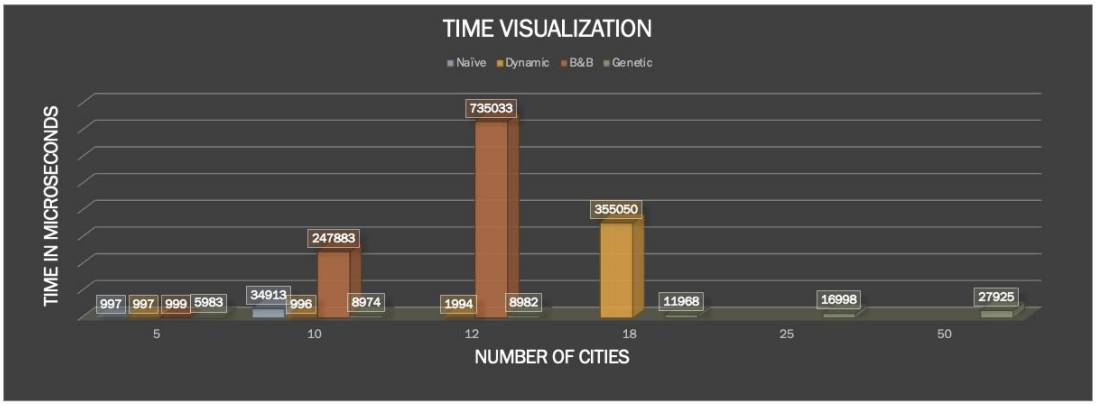


Fig-3

The time taken by an algorithm to solve a given task is of prime importance in applications where there is a time constraint. The X axis of the graph above represents the number of cities, and the Y axis represents time in microseconds. It is evident that, for smaller number of cities (ex- 5 cities) the time taken by the 3 non-heuristic algorithms are almost the same and whereas genetic performs poorly. As the number of cities increases above 10, the B&B algorithm has a great spike showing that it performs the worst out of the 4 algorithms. The sharp increase from 247883 microseconds for 10 cities to 735033 microseconds for 12 cities is proof to this statement. The implementation time of B&B is high because it performs some complex operations like node creation and pruning which is the reason for its mismatch with the time complexity of the algorithm. Dynamic performs well up to 18 cities after which it struggles due to its exponential nature. The genetic algorithm takes a lot of time on smaller datasets compared to the others but maintains a constant growth rate throughout. It performs way better than dynamic for 18 cities as shown above and has a slow growth rate for higher number of cities where the other algorithms fail to work.

Result:

Adding the optimality graph, where we determined that for small number of cities (1-12 range) the best algorithm to use where B&B, Naïve, and Dynamic, now from the new evidence found in the above graph for time analysis, we can conclude that the best algorithm for small data is the **Dynamic programming method** due to its relatively low computation-time growth. For larger datasets, genetic algorithm is the best approach comparatively as it not only tries to give optimal outputs for the cases where the other algorithms fail but also has a constant growth rate for computation time. The tradeoff between the optimality and computation taken is a field specific tradeoff and must be determined based on the relative importance of time over distance. For example, a tour system may compromise on speed of calculation to get a more optimal solution but for something like a satellite system cannot compromise on time as it has to work quickly for uninterrupted service.

Genetic algorithm specific analysis

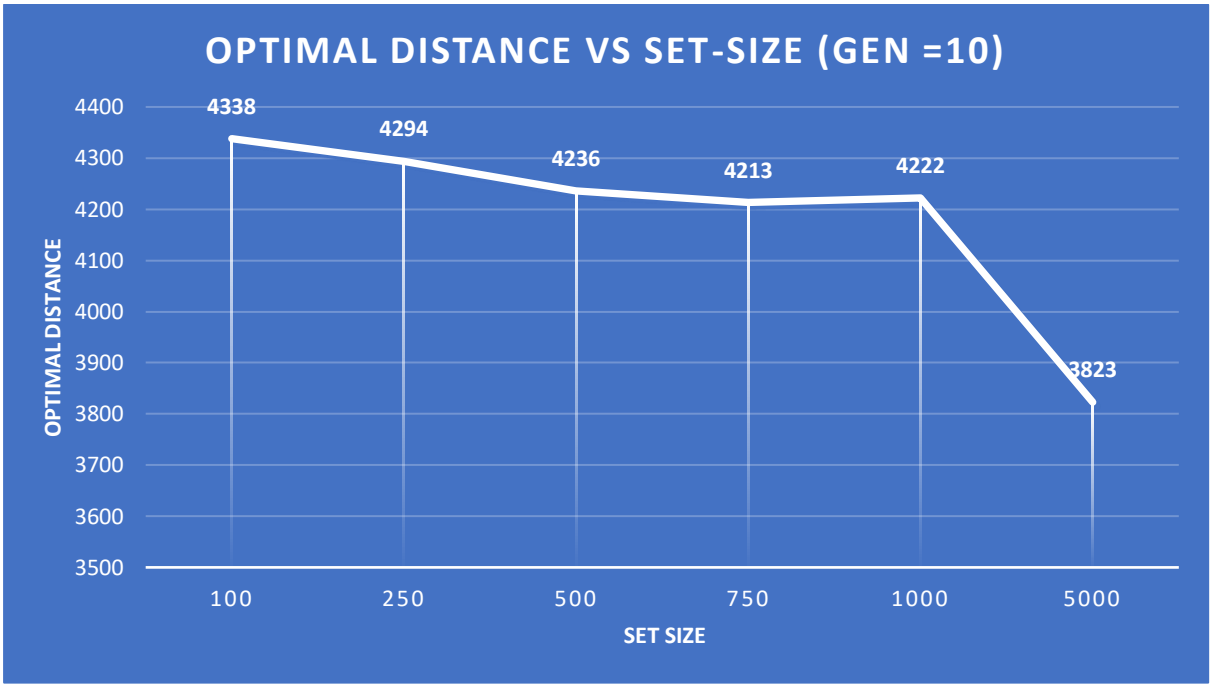


Fig-4

Now that we have established the relationship between time and optimality in general for the 4 algorithms, we now dive deeper into the genetic algorithm which has two hyperparameter which can be tuned to get a better solution – **Population/solution set size** and total number of **generations/iterations**. Firstly, we try to tune the population size by fixing the total number of generations to a constant (gen=10) and number of nodes as 100. The X axis represents the population size, and the Y axis represents the optimal distance. As seen from the graph above the path distance has a decreasing trend as the population size increases. This is intuitionally correct as a bigger population size means a greater number of solutions in each population. This increases the chance of finding an ideal parent for the mutation process which is the basis of the entire genetic algorithm. The sharp decrease from size=1000 to size=5000 can be observed above. The decrease however stops at a certain point where the optimality is the closest to the real optimal value for a given set of cities.

Result:

We can safely conclude that increasing the set size hyperparameter proves to be beneficial in regards with the optimality of the solution. This is intuitional as more solutions per iteration means more chances of finding an optimal parent solution to create a better child solution set. The recommended set size would be above 1000 for a good solution but depends on the task at hand as increasing set size would also mean increasing the computation time.

SET SIZE VS TIME ANALYSIS:

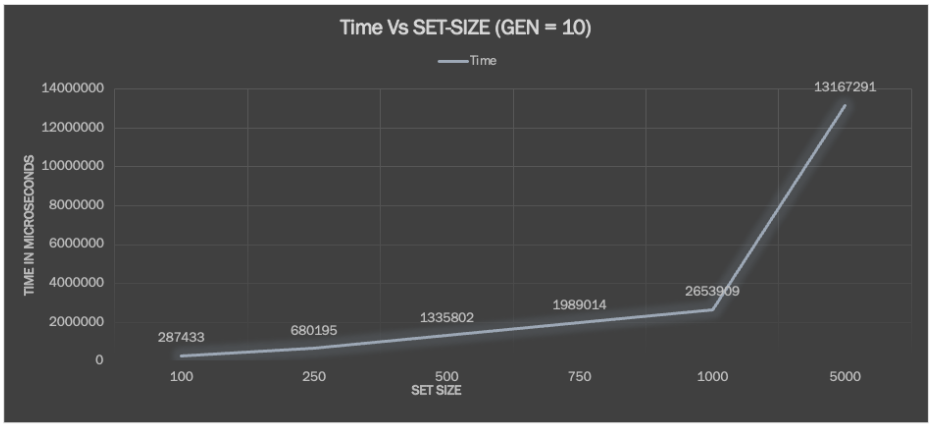
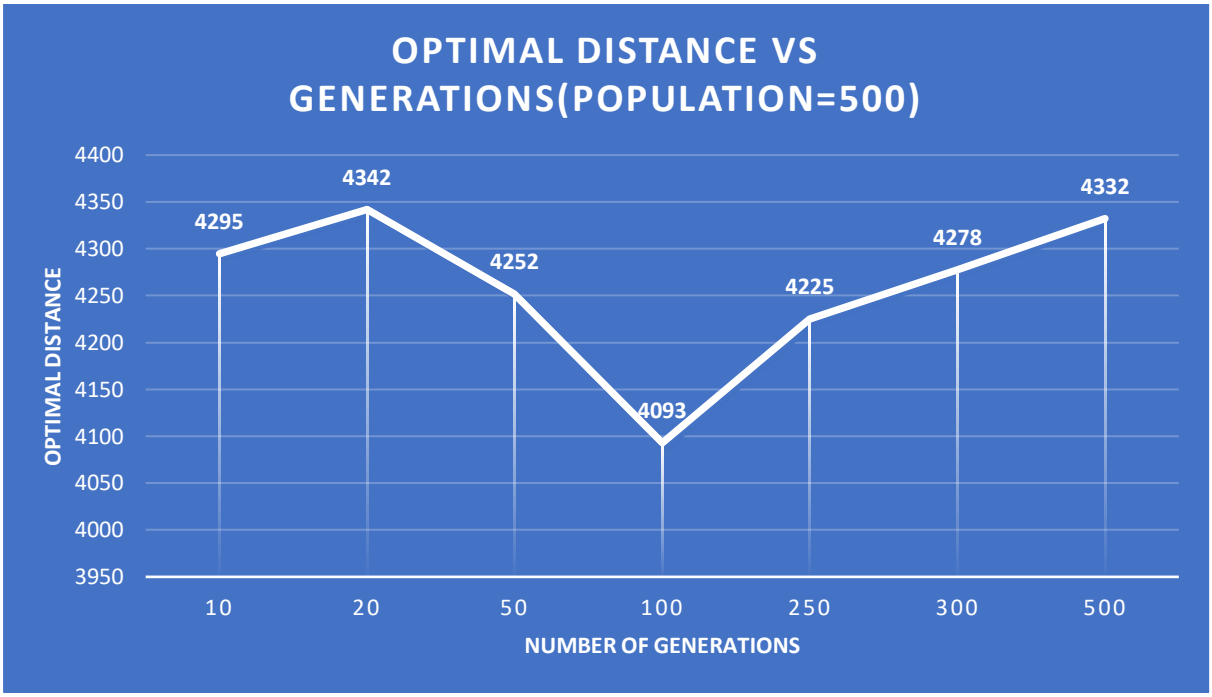


Fig-5

In the graph above, we map the relation between population size and computation time. As expected, the time increases as we increase the population size. Bigger population means more solutions to compute for the algorithm. The increase of computation time for an increase in 250 cities is around 5 lakh microseconds for 100 cities and gen size of 10.

Result:

The computation time is directly proportional to the population size as a greater number of solutions means more computation. The correct choice of set size is thus a field specific choice as in some cases speed is more important than optimality and in other cases it's the opposite.

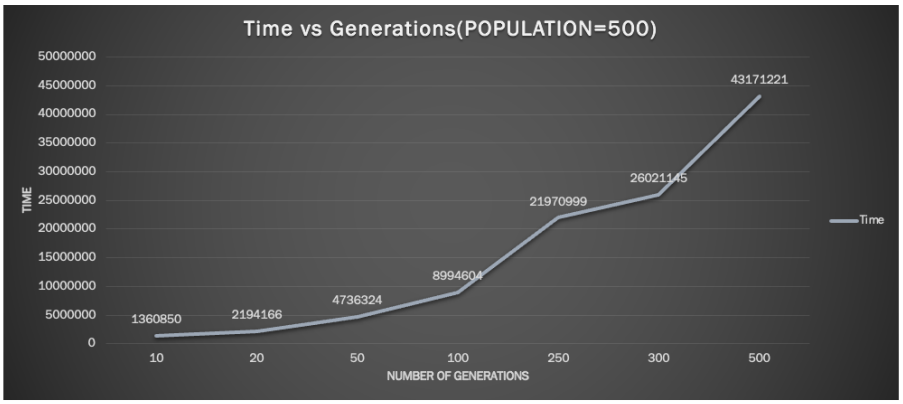


The second parameter that can be tuned is the total number of generations. By increasing the total number of generations/iterations we are making the algorithm repeat the same process of offspring creation multiple times. We are setting the population size to a constant (population=500) and the number of cities to 100. The X axis represents the number of generations and the Y axis represents the optimal distance. As you can notice above, there is a dip in the curve at gen=1000. At first this may seem illogical but there is a reason for this. The process of mutation is an educated guess where we interchange the nodes randomly for several times and check their fitness values. So, it is completely possible that sometimes the random process selects a higher value than the previous time we ran the code for a lower number of iterations. This is also the case for creating an initial population. This is also a random process. Thus, it is not only logical but also necessary that there are random dips and spikes in the graph. To confirm this idea, we expanded the graph above for higher generation values and we noticed the same trend. The graph has many dips and spikes. We can consider the least cost path as the optimal path after a few changes in the number of generations for practical purposes.

Result:

We thus safely conclude that due to the random nature of mutation and the random process of creating an initial population each time we run the program, it is reasonable for the us to observe dips and spikes in the graph and for an optimal path, we must take the least cost path among the several paths that are obtained. In the graph above the best distance would be 4093 units.

GENERATIONS VS TIME:



In this graph we compare the number of generations to the computation time where the X axis shows the number of generations, and the Y axis shows the computation time. We set the population size and number of cities to constant values (500 and 100 respectively). The computation time increases sharply as we increase the number of generations. This is reasonable as a greater number of iterations means that we are repeating the same process for a greater number of times.

Result:

We can conclude that the time complexity increases with an increase in generations. There is no correct value for this hyperparameter and depends on the use case.

Conclusion:

We put up the result in a tabular for easy of understanding:

For small datasets (1-20)	Best algorithm – Dynamic programming approach
For large datasets (20+)	Best algorithm - Genetic algorithm

Hyperparameters for genetic algorithm:

Population/set size	Higher population size gives better results but depends on the use case due to computation time.