The Traveling Sales Problem using genetic algorithm

The problem is to minimize the distance travelled by a salesperson as they visit all the cities, visiting each city exactly once.

In [9]:
```
import random
import math
import re
from prettytable import PrettyTable

#This class to define the stucture of node with city id and its 2D coordinates (x,y)
class Node:
    def __init__(self, id, x, y):
        self.id = int(id)
        self.x = float(x)
        self.y = float(y)

#Input city data and its co-ordinates (x, y) - This is various benchmark data
#file_name = "burma14.tsp"
#file_name = "eil51.tsp"
#file_name = "berlin52.tsp"
file_name = "eil76.tsp"
#file_name = "lin105.tsp"
#file_name = "bier127.tsp"
#file_name = "gr137.tsp"
#file_name = "rat195.tsp"
#file_name = "lin318.tsp"
#file_name = "rat575.tsp"


#Creating data set based on input city files
dataset = []
with open(file_name, "r") as f:
    for line in f:
        new_line = re.split(r'\s+', line.strip())
        if new_line[0].isdigit():
            id, x, y = new_line[0], float(new_line[1]), float(new_line[2])
            dataset.append(Node(id=id, x=x, y=y))

N = len(dataset) # Number of cities
print("Number of Cities:" + str(N))

#Creating distance matrix using Euclidean distance formula
def distance_matrix(node_list):
    matrix = [[0 for _ in range(N)] for _ in range(N)]

    for i in range(0, len(matrix)-1):
        for j in range(0, len(matrix[0])-1):
            matrix[node_list[i].id][node_list[j].id] = math.sqrt(
                pow((node_list[i].x - node_list[j].x), 2) + pow((node_list[i].y - node_l
            )
    return matrix

matrix = distance_matrix(dataset)

# The class is for Chromosome for maintaing Node list. This will be used in crossover, m
class Chromosome:
    def __init__(self, node_list):
        self.chromosome = node_list
```

```
        chr_representation = []
        for i in range(0, len(node_list)):
            chr_representation.append(self.chromosome[i].id)
        self.chr_representation = chr_representation

        distance = 0
        for j in range(1, len(self.chr_representation) - 1):  # get distances from the m
            distance += matrix[self.chr_representation[j]-1][self.chr_representation[j +
        self.cost = distance

        self.fitness_value = 1 / self.cost


# create a random chromosome and start and end points should be same
def create_random_list(n_list):

    start = n_list[0]
    temp = n_list[1:]
    temp = random.sample(temp, len(temp))
    temp.insert(0, start)
    temp.append(start)
    return temp


# initialize the population
def initialization(data, pop_size):
    initial_population = []
    for i in range(0, pop_size):
        temp = create_random_list(data)
        new_ch = Chromosome(temp)
        initial_population.append(new_ch)
    return initial_population


# Select parent chromosomes to create child chromosomes using tournament selection
def selection(population):
    ticket_1, ticket_2, ticket_3, ticket_4 = random.sample(range(0, 99), 4)

    # create candidate chromosomes based on ticket numbers
    candidate_1 = population[ticket_1]
    candidate_2 = population[ticket_2]
    candidate_3 = population[ticket_3]
    candidate_4 = population[ticket_4]

    # select the winner according to their costs
    if candidate_1.fitness_value > candidate_2.fitness_value:
        winner = candidate_1
    else:
        winner = candidate_2

    if candidate_3.fitness_value > winner.fitness_value:
        winner = candidate_3

    if candidate_4.fitness_value > winner.fitness_value:
        winner = candidate_4

    return winner


# Two points crossover
def crossover(p_1, p_2):
    point_1, point_2 = random.sample(range(1, len(p_1.chromosome)-1), 2)
    begin = min(point_1, point_2)
    end = max(point_1, point_2)
```

```
    child_1_1 = p_1.chromosome[:begin]
    child_1_2 = p_1.chromosome[end:]
    child_1 = child_1_1 + child_1_2
    child_2 = p_2.chromosome[begin:end+1]

    child_1_remain = [item for item in p_2.chromosome[1:-1] if item not in child_1]
    child_2_remain = [item for item in p_1.chromosome[1:-1] if item not in child_2]

    child_1 = child_1_1 + child_1_remain + child_1_2
    child_2 += child_2_remain

    child_2.insert(0, p_2.chromosome[0])
    child_2.append(p_2.chromosome[0])

    return child_1, child_2


# Mutation operation
def mutation(chromosome):  # swap two nodes of the chromosome
    mutation_index_1, mutation_index_2 = random.sample(range(1, 10), 2)
    chromosome[mutation_index_1], chromosome[mutation_index_2] = chromosome[mutation_ind
    return chromosome


# Find the best chromosome of the generation based on the cost
def find_best(generation):
    best = generation[0]
    for n in range(1, len(generation)):
        if generation[n].cost < best.cost:
            best = generation[n]
    return best

# Use elitism, crossover, mutation operators to create a new generation based on a previ

def create_new_generation(previous_generation, crossover_probability, mutation_rate):
    new_generation = [find_best(previous_generation)]  # This is for elitism. Keep the b

    # Use two chromosomes and create two chromosomes. So, iteration size will be half of
    for a in range(0, int(len(previous_generation)/2)):
        parent_1 = selection(previous_generation)
        parent_2 = selection(previous_generation)

        if random.random() < crossover_probability:
            child_1, child_2 = crossover(parent_1, parent_2)  # This will create node li
            child_1 = Chromosome(child_1)
            child_2 = Chromosome(child_2)
        else:
            child_1 = parent_1
            child_2 = parent_2

        if random.random() < mutation_rate:
            mutated = mutation(child_1.chromosome)
            child_1 = Chromosome(mutated)

        new_generation.append(child_1)
        new_generation.append(child_2)

    return new_generation


def genetic_algorithm(num_of_generations, pop_size, cross_prob, mutation_rate, data_list

    new_gen = initialization(data_list, pop_size)

    costs_for_plot = []
    for iteration in range(0, num_of_generations):
```

```
        new_gen = create_new_generation(new_gen, cross_prob, mutation_rate)
        costs_for_plot.append(find_best(new_gen).cost)

    return new_gen, costs_for_plot
```

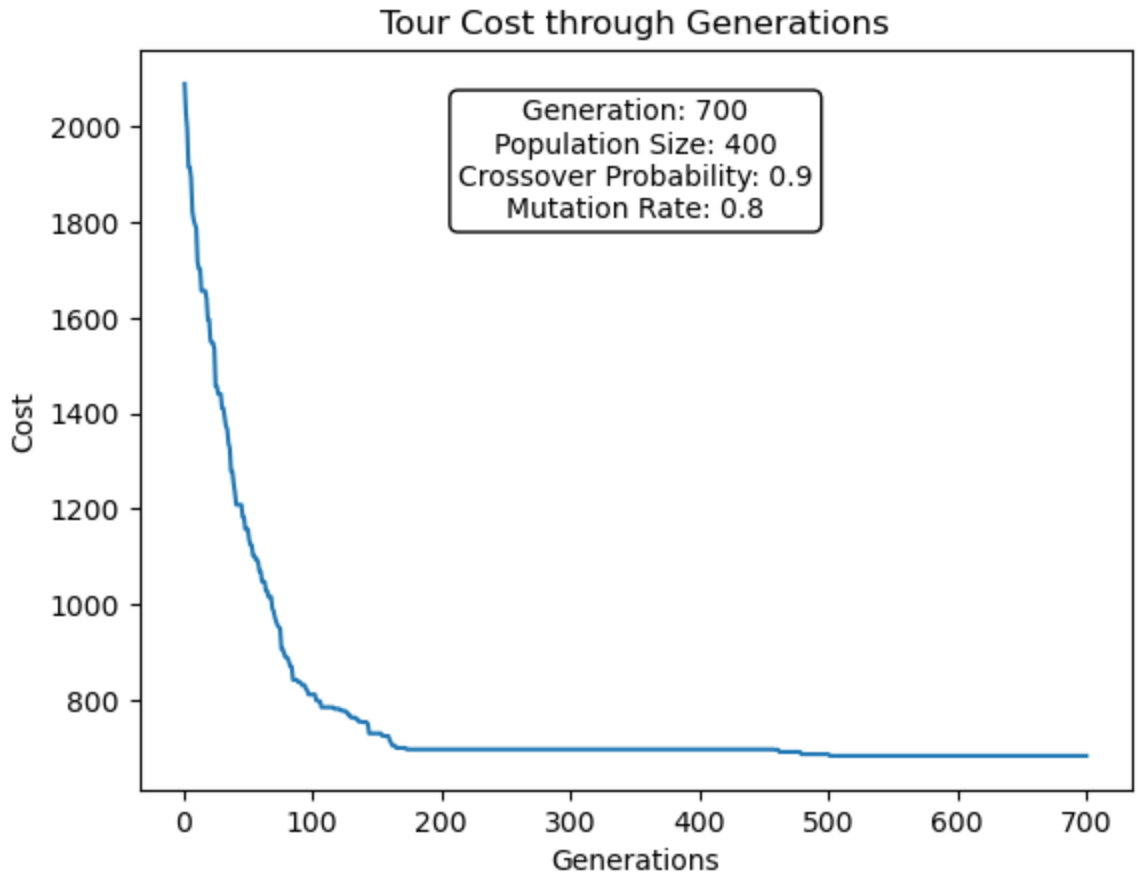
```
Number of Cities:76
```

In [10]:
```
import numpy as np
import matplotlib.pyplot as plt

def draw_path(solution):
    x_list = []
    y_list = []

    for m in range(0, len(solution.chromosome)):
        x_list.append(solution.chromosome[m].x)
        y_list.append(solution.chromosome[m].y)

    fig, ax = plt.subplots()
    plt.scatter(x_list, y_list)  # alpha=0.5

    ax.plot(x_list, y_list, '--', lw=2, color='black', ms=10)
    ax.set_xlim(0, 2000)
    ax.set_ylim(0, 1300)

    plt.show()

def draw_cost_generation(y_list, generation, pop_size, cross_prob, mutation_rate):
    x_list = np.arange(1, len(y_list) + 1)  # create a numpy list from 1 to the numbers

    plt.plot(x_list, y_list)

    plt.title("Tour Cost through Generations")
    plt.xlabel("Generations")
    plt.ylabel("Cost")

    # Add annotation with parameter values
    parameter_label = f'Generation: {generation}\nPopulation Size: {pop_size}\nCrossover
    plt.annotate(parameter_label, xy=(0.5, 0.85), xycoords='axes fraction', fontsize=10,
                 bbox=dict(boxstyle="round,pad=0.3", edgecolor="black", facecolor="white

    plt.show()
```

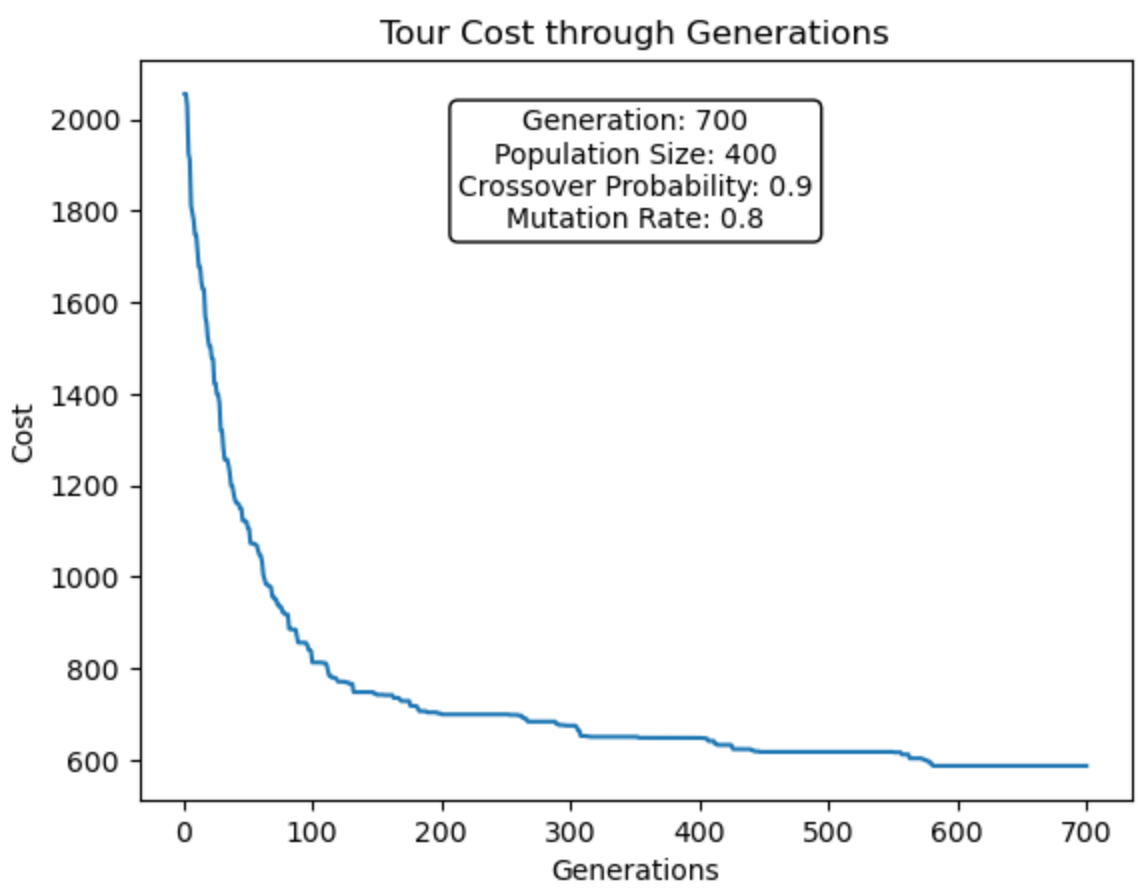
In [11]:
```
import random

# Set the number of runs
num_runs = 30

# Genetic Algorithm Parameter Combinations
parameter_combinations = [
    (700, 400, 0.9, 0.8),
    # Add more combinations as needed
]

# Loop through each parameter combination
for params in parameter_combinations:
    # Extract parameters
    numbers_of_generations, population_size, crossover_probability, mutation_rate = para

    # Loop through multiple runs with different random seeds
    for run in range(num_runs):
        # Set a different random seed for each run
        random_seed = run + 1
        random.seed(random_seed)
```

```
        # Run genetic algorithm
        last_generation, y_axis = genetic_algorithm(
            num_of_generations=numbers_of_generations,
            pop_size=population_size,
            cross_prob=crossover_probability,
            mutation_rate=mutation_rate,
            data_list=dataset
        )

        # Find the best solution
        best_solution = find_best(last_generation)

        # Display results for each run
        best_cost_last_generation = last_generation[0].cost
        best_path_last_generation = last_generation[0].chr_representation
        print(f"Run {run + 1} - Minimum tour length: {best_cost_last_generation:.2f}")
        print(f"Run {run + 1} - Best path: {best_path_last_generation}")

        # Draw cost vs generation plot for each run
        draw_cost_generation(y_axis, numbers_of_generations, population_size, crossover_
```

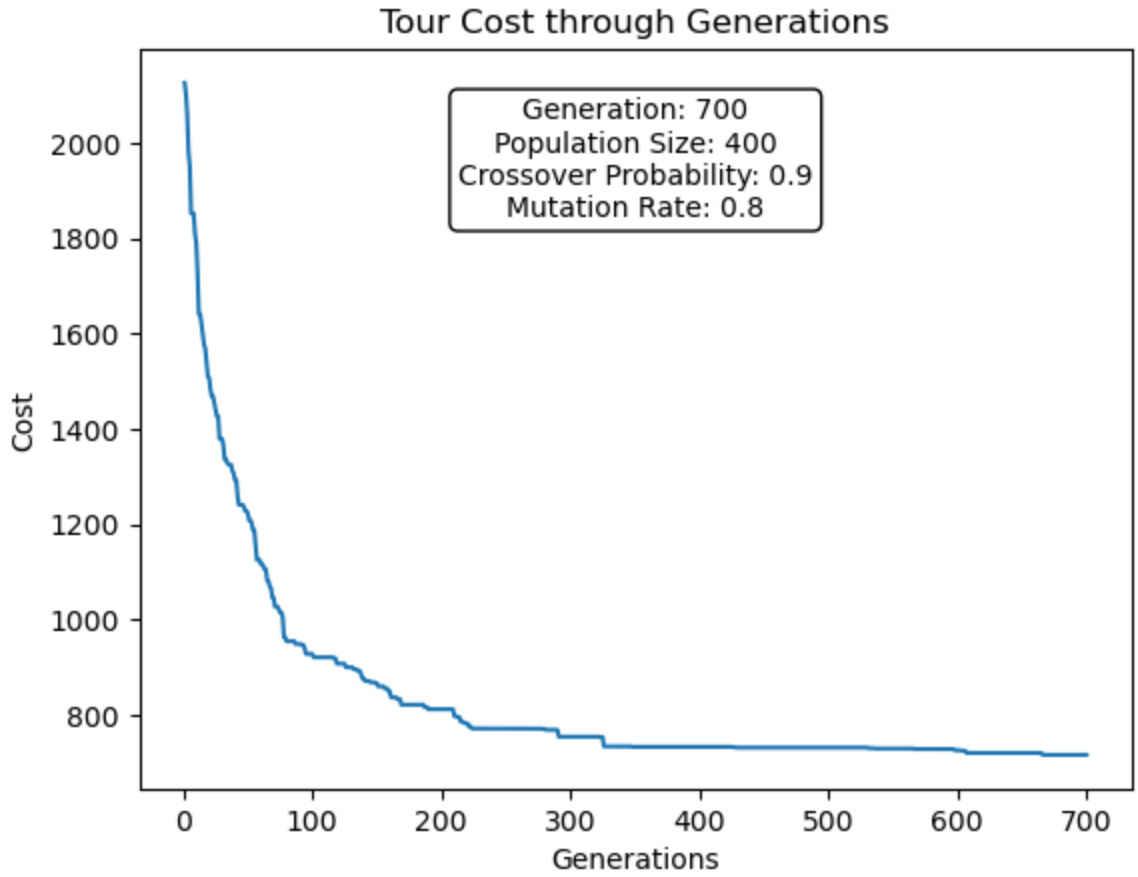
```
Run 1 - Minimum tour length: 683.27
Run 1 - Best path: [1, 56, 26, 19, 51, 45, 33, 10, 40, 13, 73, 59, 32, 11, 39, 66, 12, 6
7, 15, 60, 9, 8, 54, 36, 20, 14, 47, 35, 68, 27, 76, 52, 7, 3, 34, 74, 63, 75, 29, 22, 4
8, 37, 21, 61, 70, 72, 71, 38, 6, 16, 58, 55, 28, 53, 5, 46, 49, 30, 31, 69, 18, 41, 4,
17, 50, 25, 24, 42, 43, 44, 57, 64, 2, 23, 65, 62, 1]
```




```
Run 2 - Minimum tour length: 586.58
Run 2 - Best path: [1, 25, 50, 24, 57, 42, 65, 43, 44, 64, 17, 4, 45, 33, 51, 19, 56, 2
6, 10, 32, 11, 59, 73, 40, 13, 41, 18, 27, 68, 8, 47, 35, 5, 76, 69, 3, 31, 46, 16, 58,
14, 55, 20, 36, 54, 39, 66, 67, 12, 60, 15, 9, 53, 28, 30, 6, 38, 21, 71, 61, 72, 37, 7
0, 48, 49, 75, 22, 62, 29, 23, 63, 2, 74, 34, 7, 52, 1]
```

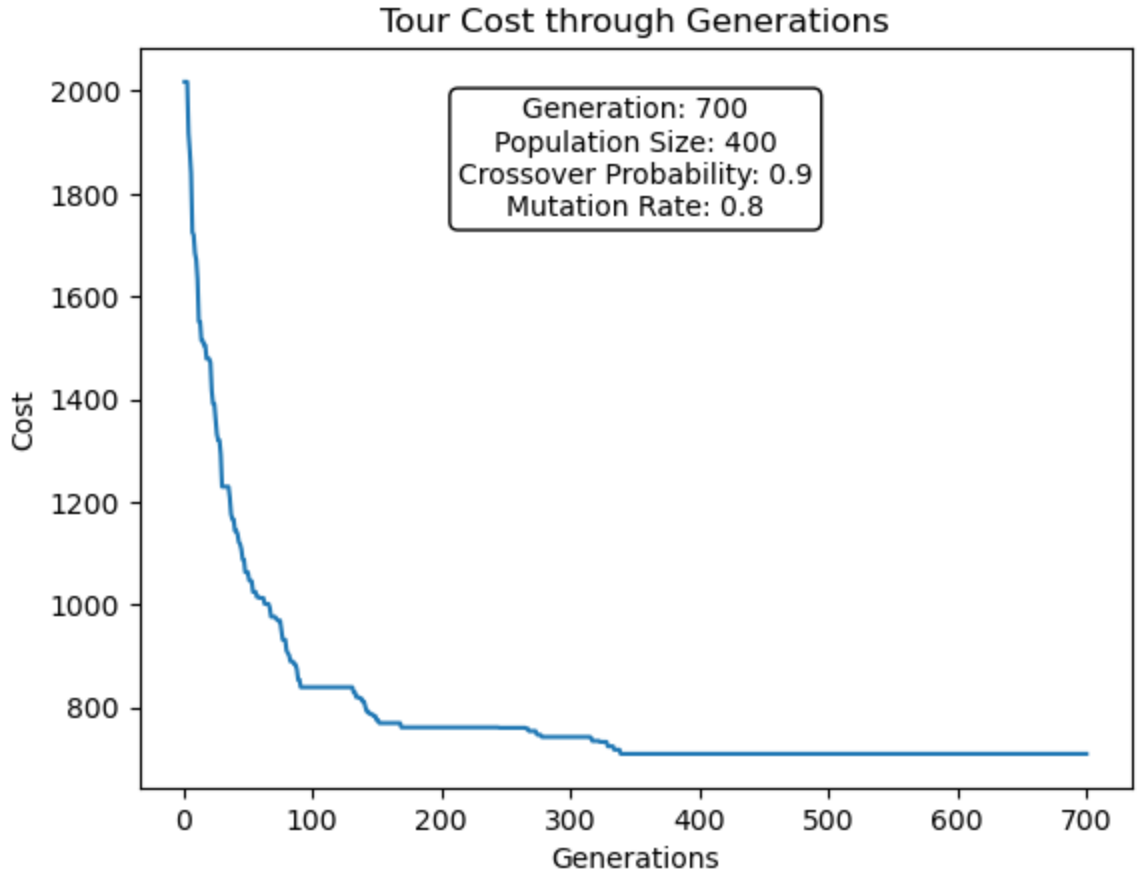



```
Run 3 - Minimum tour length: 716.43
Run 3 - Best path: [1, 65, 43, 42, 44, 2, 64, 24, 57, 25, 50, 17, 34, 74, 63, 23, 62, 3
7, 58, 55, 15, 60, 54, 11, 32, 73, 18, 52, 41, 4, 45, 56, 26, 19, 51, 33, 10, 40, 13, 2
7, 35, 28, 6, 72, 70, 61, 71, 21, 38, 16, 30, 46, 31, 49, 48, 22, 29, 75, 3, 7, 69, 76,
5, 53, 9, 36, 14, 20, 47, 68, 8, 12, 66, 67, 39, 59, 1]
```

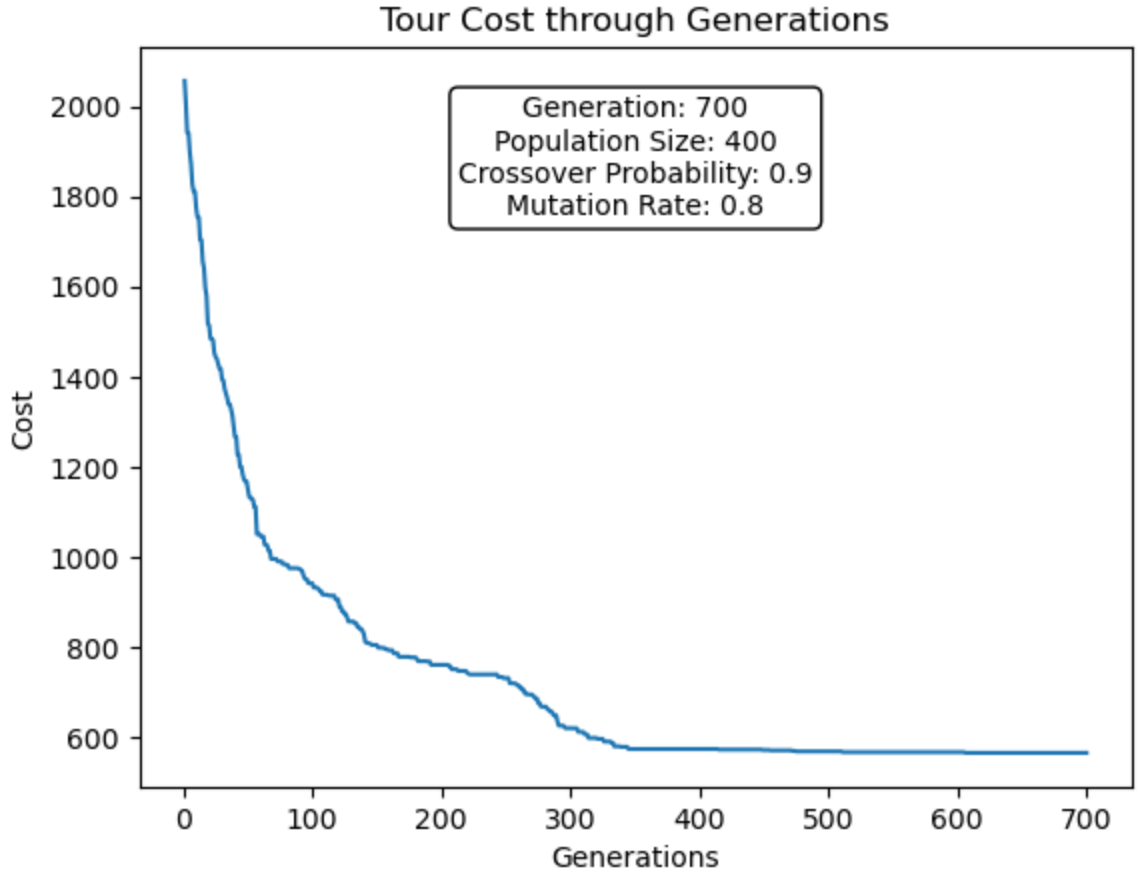



```
Run 4 - Minimum tour length: 709.48
Run 4 - Best path: [1, 57, 24, 50, 25, 19, 51, 26, 56, 33, 41, 4, 45, 10, 40, 32, 73, 5
9, 11, 39, 66, 67, 60, 15, 54, 12, 9, 8, 27, 13, 18, 52, 17, 34, 74, 2, 64, 42, 44, 65,
```

43, 23, 29, 63, 3, 22, 62, 75, 31, 76, 7, 69, 5, 46, 6, 30, 14, 20, 36, 68, 35, 47, 53, 28, 55, 38, 61, 37, 72, 21, 58, 16, 49, 48, 70, 71, 1]

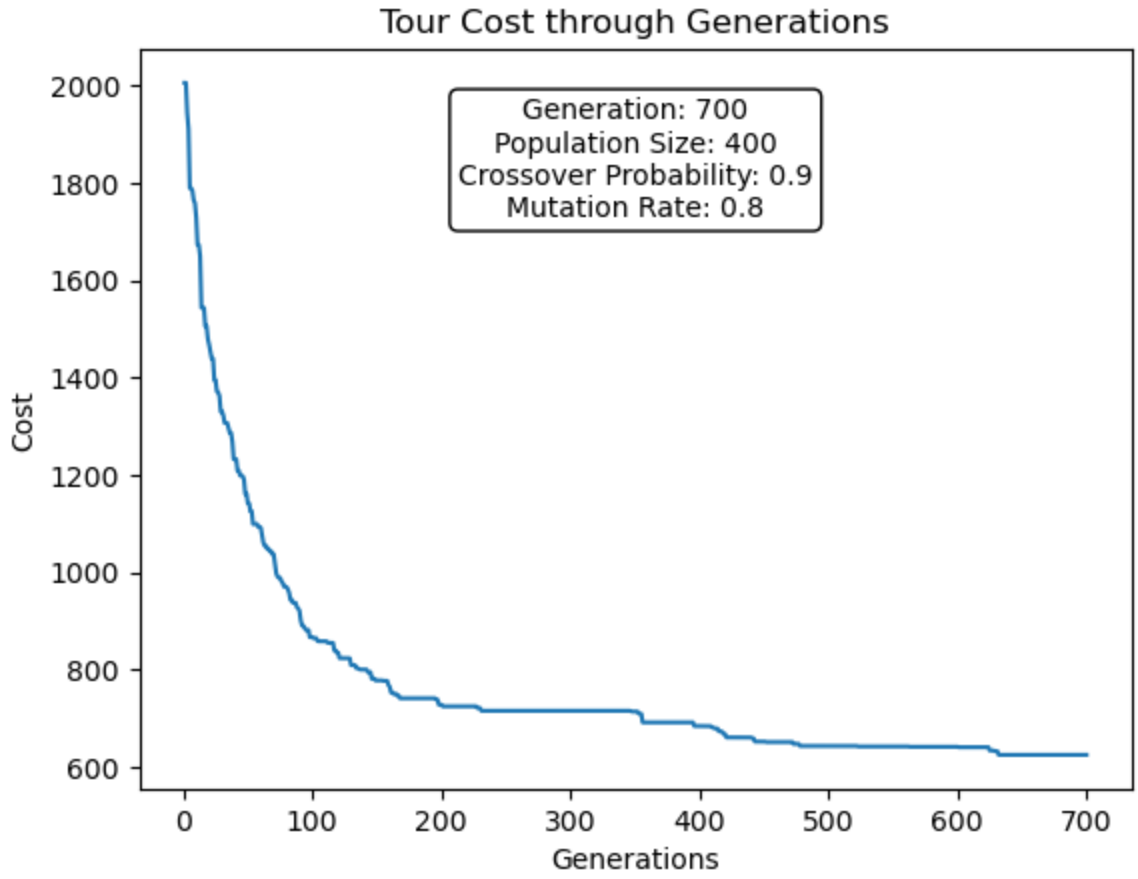


Run 5 - Minimum tour length: 565.47
Run 5 - Best path: [1, 32, 11, 39, 66, 67, 12, 60, 15, 54, 36, 8, 27, 47, 9, 20, 55, 14, 28, 53, 35, 68, 5, 76, 7, 69, 3, 31, 49, 30, 46, 58, 16, 6, 38, 21, 71, 61, 72, 70, 37, 48, 22, 62, 75, 29, 23, 63, 74, 34, 52, 17, 64, 2, 44, 43, 65, 42, 57, 24, 50, 25, 19, 56, 26, 51, 33, 45, 4, 18, 13, 41, 10, 40, 73, 59, 1]

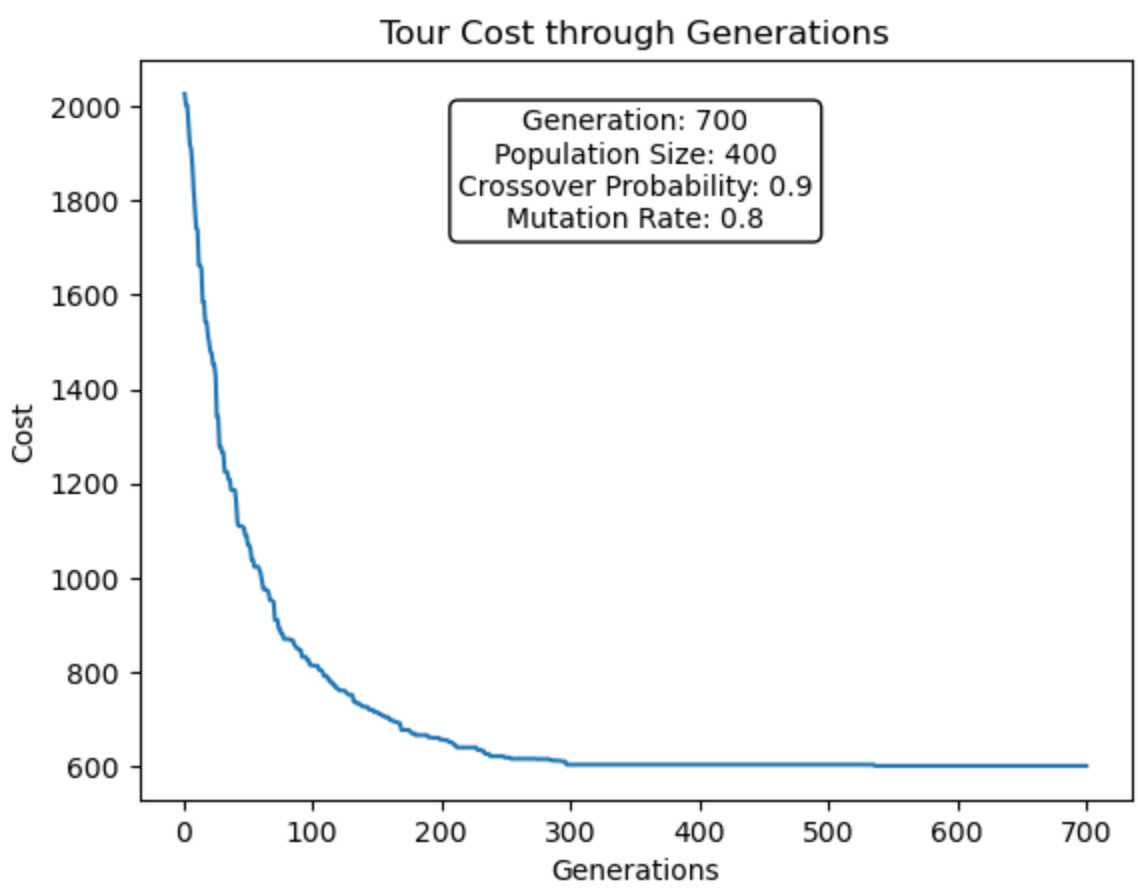


Run 6 - Minimum tour length: 625.62

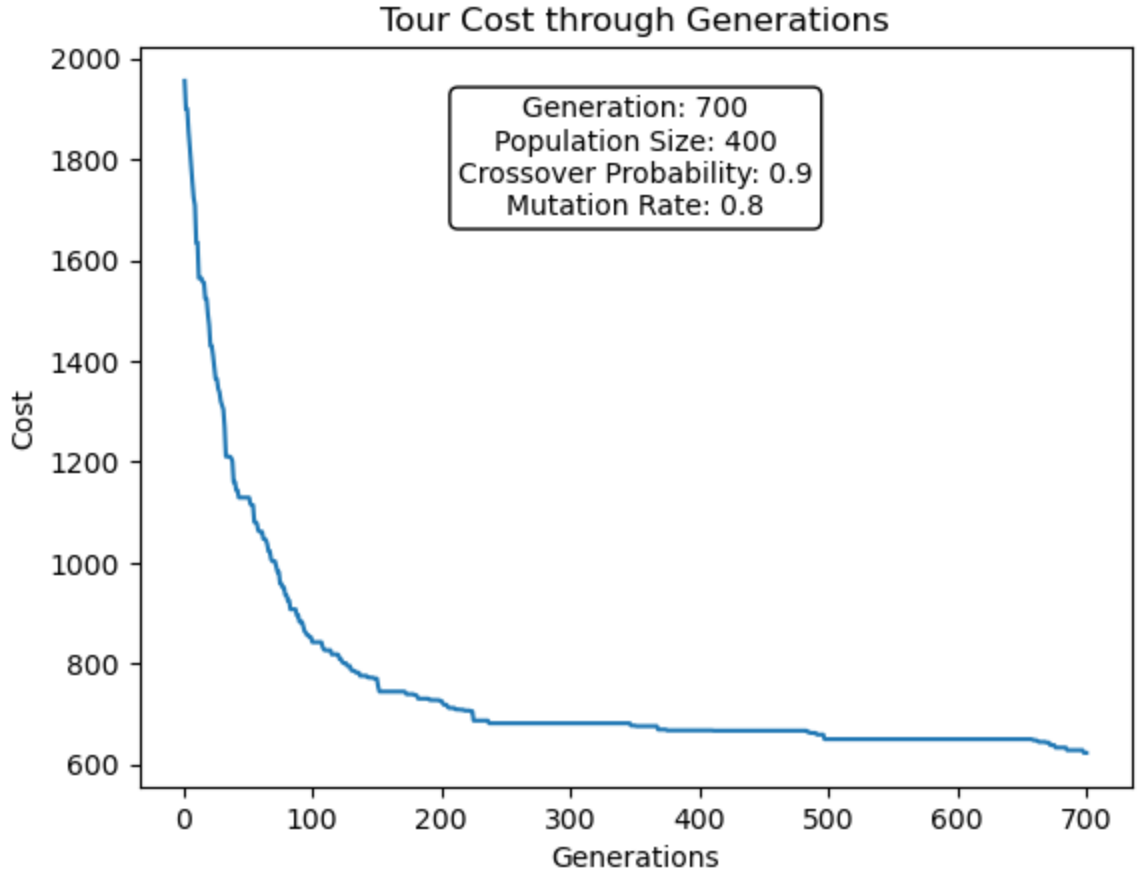
Run 6 - Best path: [1, 55, 20, 15, 60, 67, 12, 66, 39, 11, 32, 59, 73, 40, 10, 4, 64, 3
4, 69, 3, 31, 75, 22, 48, 49, 38, 6, 30, 46, 5, 68, 27, 76, 7, 17, 52, 18, 13, 41, 45, 1
9, 56, 26, 33, 51, 25, 50, 57, 24, 42, 44, 65, 43, 23, 2, 74, 63, 29, 62, 70, 37, 72, 6
1, 71, 21, 16, 58, 14, 28, 53, 47, 35, 8, 54, 36, 9, 1]



Run 7 - Minimum tour length: 602.11
Run 7 - Best path: [1, 32, 40, 10, 33, 45, 4, 18, 41, 13, 73, 59, 11, 39, 66, 67, 12, 5
4, 60, 15, 20, 55, 14, 28, 53, 47, 35, 9, 36, 8, 27, 68, 5, 46, 16, 58, 21, 71, 61, 72,
37, 38, 6, 30, 49, 48, 75, 29, 63, 74, 64, 24, 57, 2, 34, 52, 7, 69, 76, 3, 31, 22, 70,
62, 23, 65, 43, 42, 44, 17, 50, 25, 19, 56, 26, 51, 1]

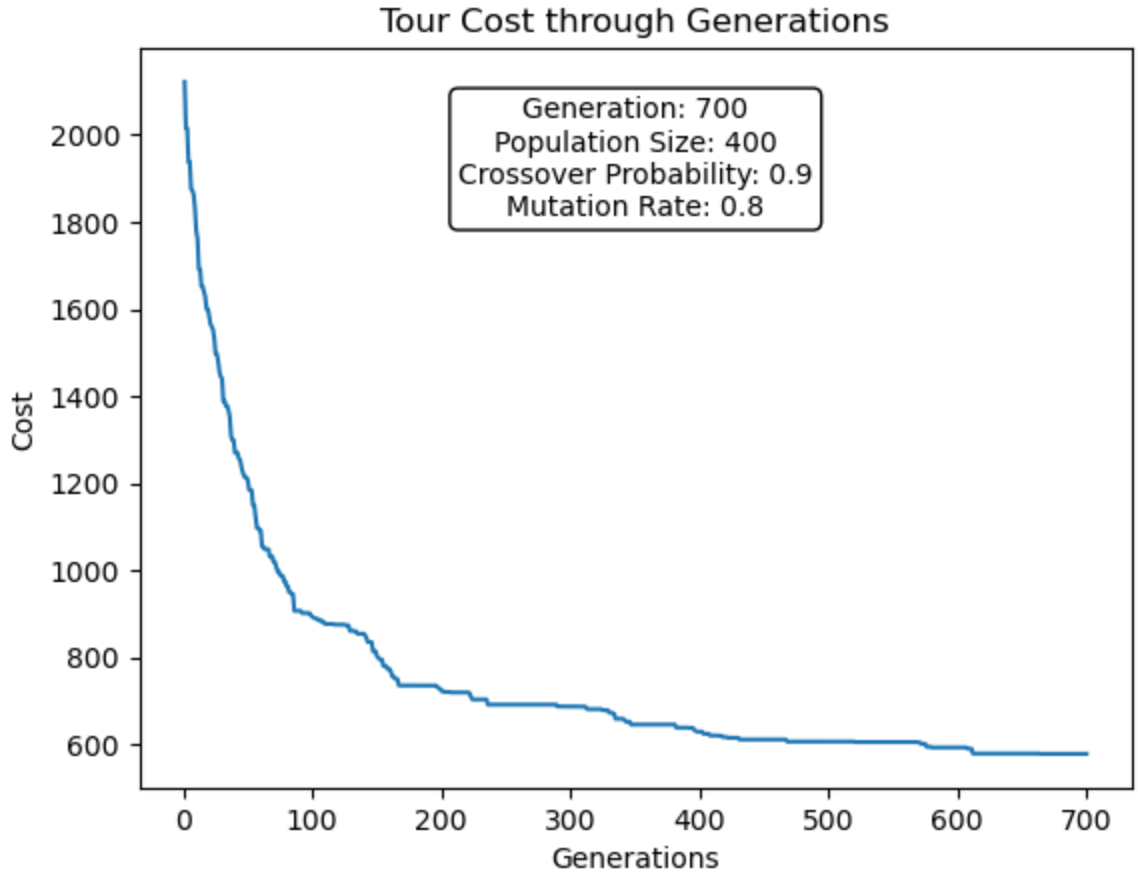


Run 8 - Minimum tour length: 622.89
Run 8 - Best path: [1, 20, 55, 14, 28, 47, 35, 53, 5, 3, 31, 49, 48, 37, 70, 72, 61, 71, 21, 38, 58, 16, 6, 30, 46, 76, 27, 18, 7, 69, 68, 8, 9, 36, 15, 60, 54, 12, 67, 66, 39, 11, 32, 59, 73, 13, 41, 33, 10, 40, 26, 56, 51, 19, 25, 4, 45, 52, 34, 2, 64, 17, 50, 24, 57, 42, 44, 43, 65, 23, 63, 74, 75, 22, 29, 62, 1]

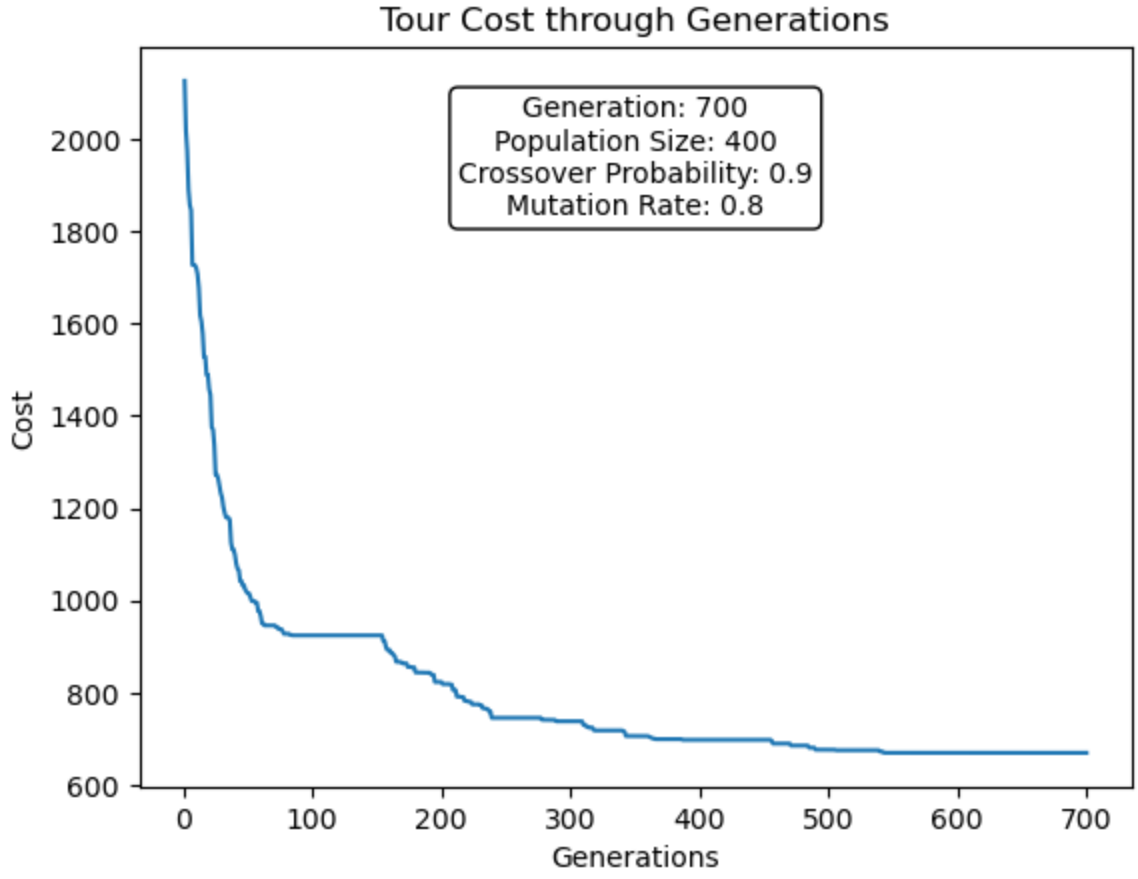


Run 9 - Minimum tour length: 577.62
Run 9 - Best path: [1, 32, 11, 66, 39, 12, 67, 60, 15, 20, 9, 36, 54, 8, 5, 68, 27, 13, 59, 73, 40, 10, 41, 18, 52, 7, 69, 76, 46, 28, 35, 47, 53, 55, 14, 58, 16, 38, 21, 71, 6

1, 72, 37, 70, 48, 6, 30, 49, 22, 75, 31, 3, 74, 63, 29, 62, 23, 65, 43, 42, 44, 2, 34, 64, 24, 57, 25, 50, 17, 4, 45, 33, 51, 19, 26, 56, 1]

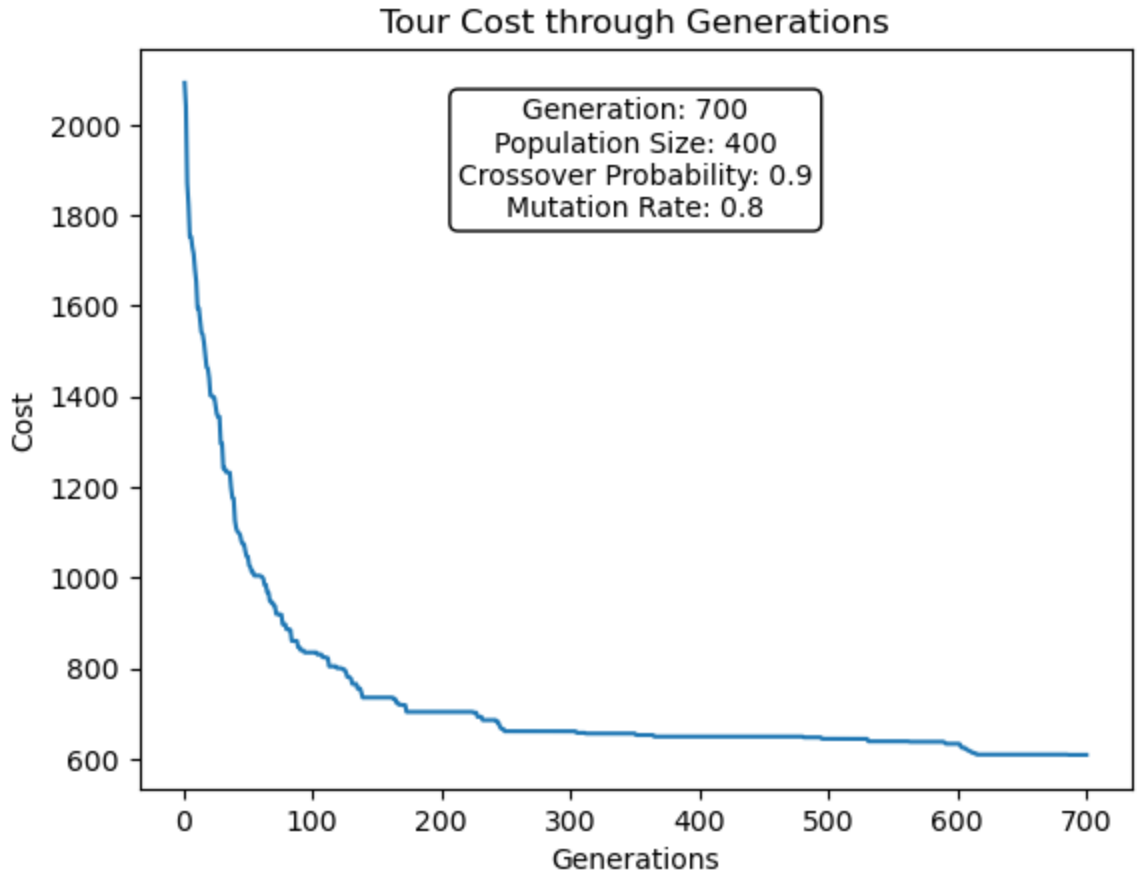


Run 10 - Minimum tour length: 669.75
Run 10 - Best path: [1, 20, 55, 14, 28, 53, 35, 47, 9, 8, 36, 54, 15, 60, 12, 39, 66, 67, 32, 11, 73, 59, 13, 41, 18, 27, 68, 76, 69, 52, 4, 45, 34, 7, 5, 58, 16, 6, 48, 63, 74, 44, 23, 65, 43, 42, 57, 24, 19, 51, 26, 56, 40, 10, 33, 25, 50, 17, 64, 2, 62, 29, 75, 3, 31, 46, 30, 49, 22, 70, 37, 38, 21, 71, 61, 72, 1]

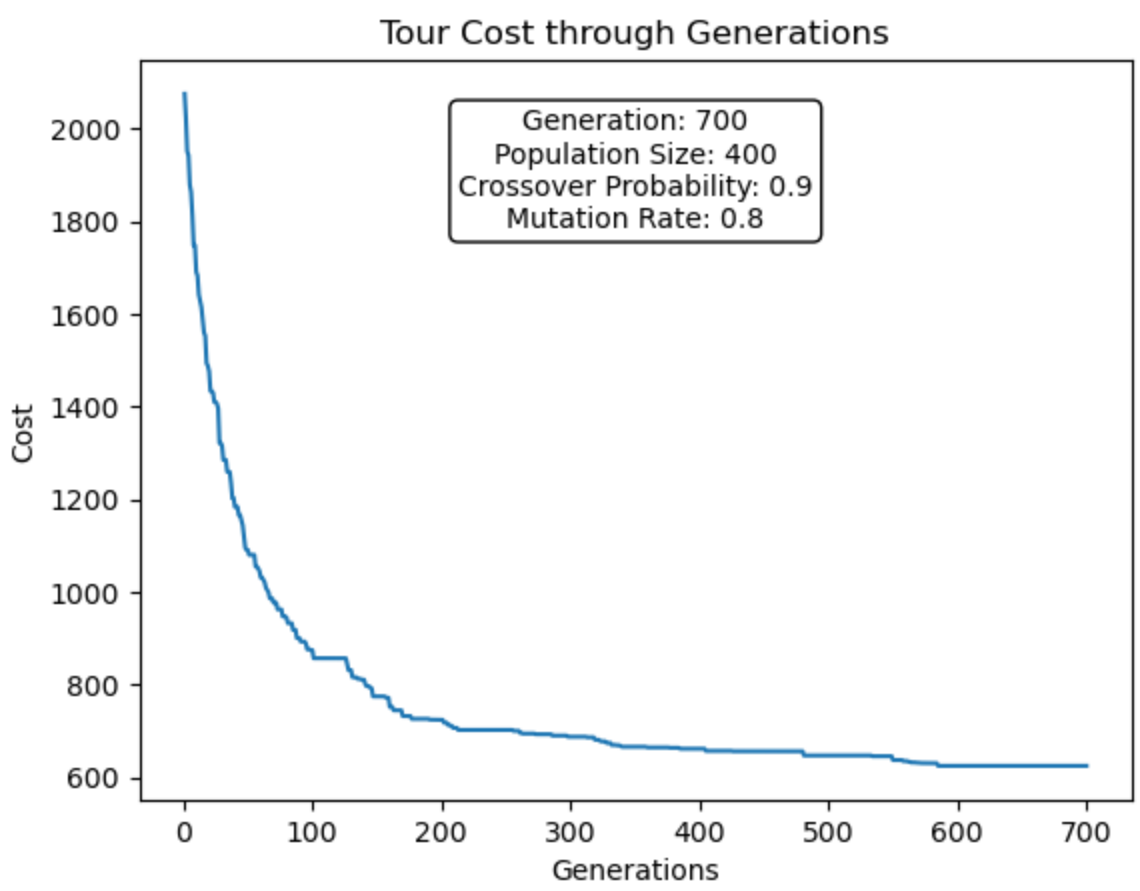


Run 11 - Minimum tour length: 608.30

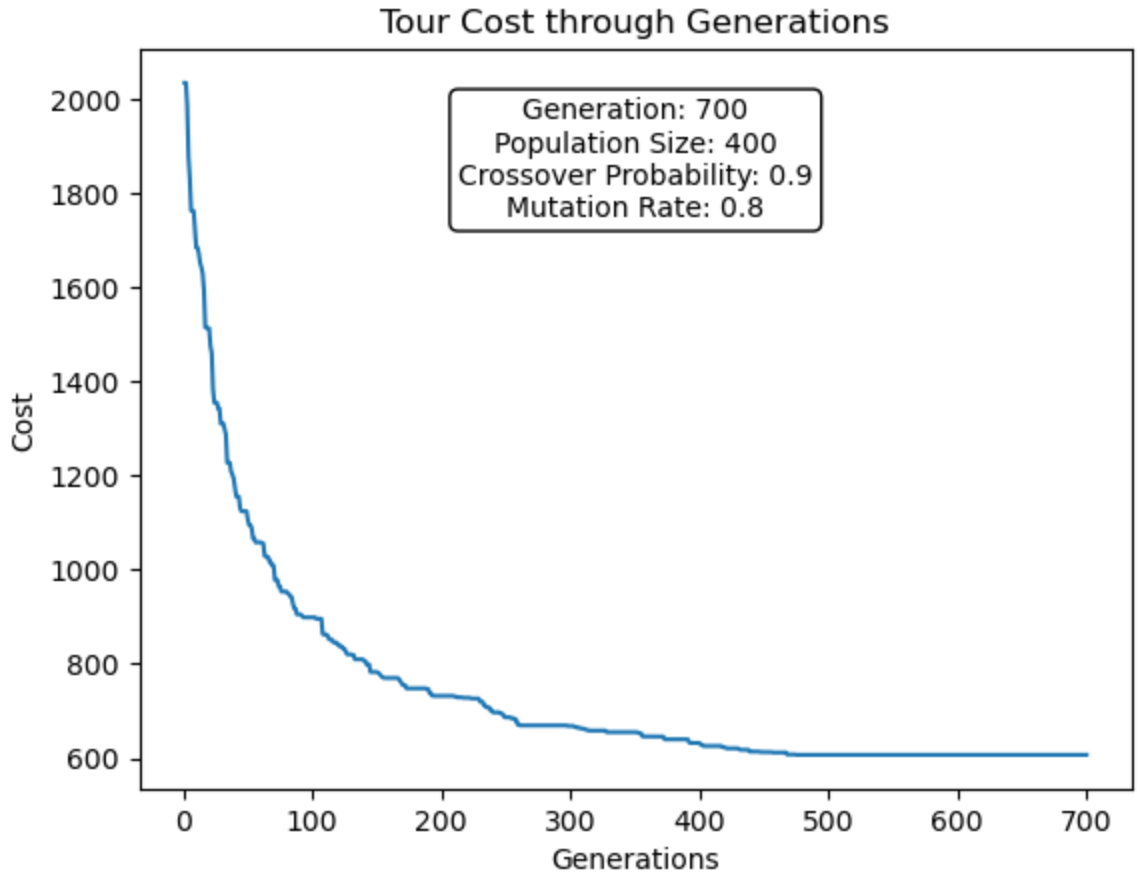
```
Run 11 - Best path: [1, 26, 56, 19, 51, 33, 10, 41, 45, 4, 25, 50, 17, 2, 74, 63, 75, 2
2, 62, 29, 23, 65, 43, 44, 42, 57, 24, 64, 34, 3, 7, 52, 18, 69, 76, 5, 35, 28, 46, 31,
49, 48, 37, 70, 72, 61, 71, 21, 38, 6, 30, 16, 58, 14, 55, 20, 36, 8, 54, 15, 9, 53, 47,
68, 27, 13, 73, 59, 66, 67, 60, 12, 39, 11, 40, 32, 1]
```




```
Run 12 - Minimum tour length: 625.28
Run 12 - Best path: [1, 32, 11, 59, 40, 73, 13, 41, 45, 33, 10, 26, 56, 51, 19, 25, 50,
64, 34, 74, 63, 29, 75, 2, 24, 57, 44, 42, 43, 65, 23, 62, 22, 70, 37, 72, 61, 71, 21, 3
8, 48, 49, 30, 46, 76, 69, 7, 17, 4, 52, 18, 27, 8, 35, 47, 9, 20, 60, 67, 66, 39, 12, 1
5, 54, 36, 53, 28, 14, 55, 58, 16, 6, 31, 3, 5, 68, 1]
```

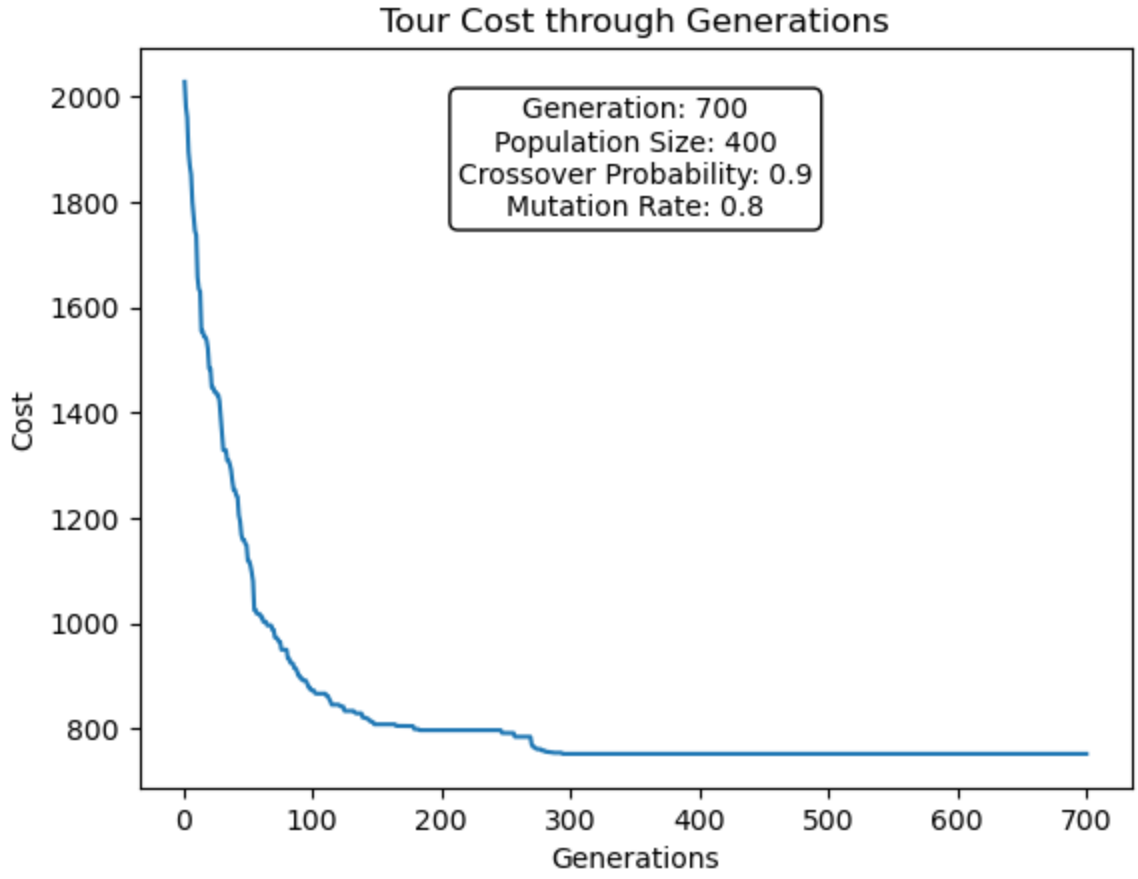


Run 13 - Minimum tour length: 606.83
Run 13 - Best path: [1, 32, 40, 10, 41, 13, 73, 59, 11, 39, 66, 67, 12, 54, 60, 15, 36, 8, 27, 68, 35, 47, 9, 20, 55, 28, 53, 14, 58, 16, 30, 46, 49, 6, 48, 37, 72, 61, 71, 21, 38, 70, 62, 65, 43, 23, 29, 34, 64, 52, 18, 7, 69, 76, 5, 31, 22, 75, 3, 63, 74, 2, 44, 42, 57, 24, 25, 50, 17, 4, 45, 33, 26, 51, 19, 56, 1]

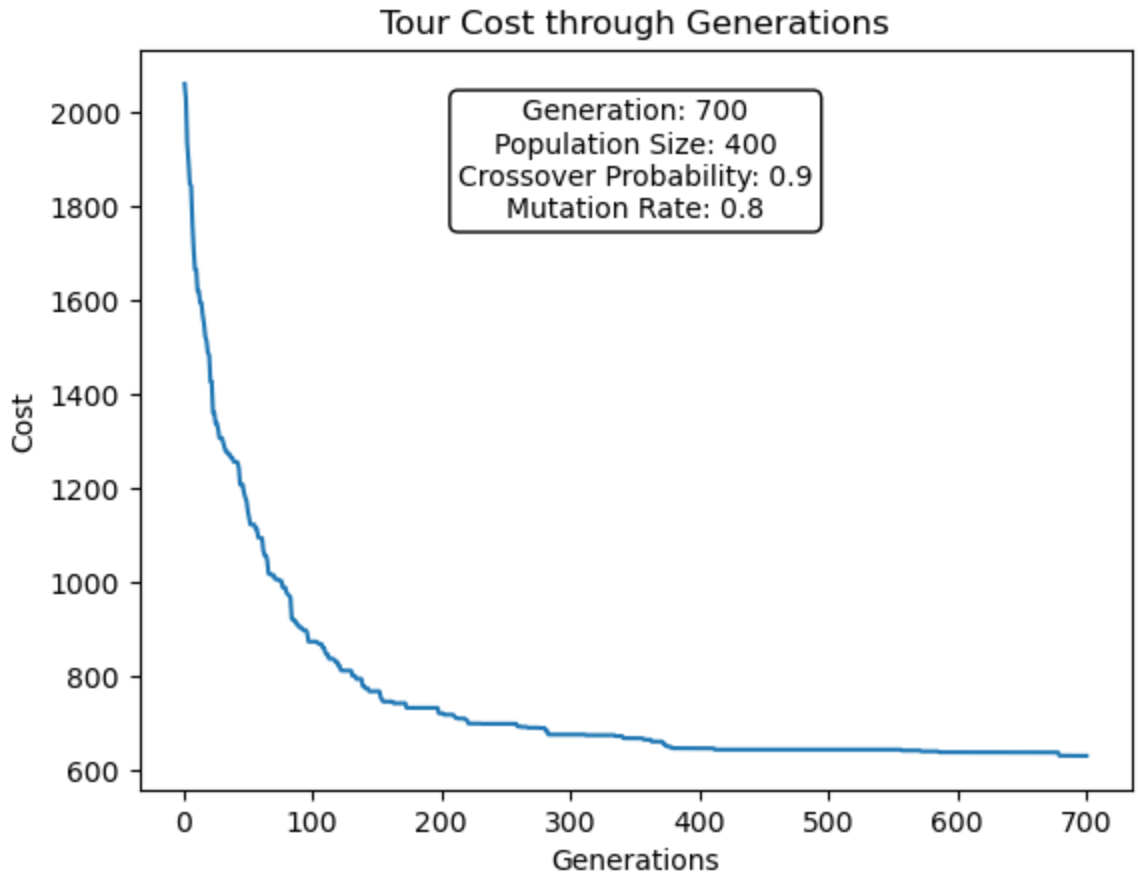


Run 14 - Minimum tour length: 752.20
Run 14 - Best path: [1, 65, 43, 42, 44, 2, 63, 74, 64, 34, 7, 52, 17, 4, 25, 50, 57, 24, 23, 62, 29, 75, 6, 38, 61, 37, 22, 48, 49, 46, 28, 55, 20, 47, 53, 9, 36, 15, 54, 12, 6

0, 67, 66, 39, 13, 68, 27, 76, 69, 3, 31, 70, 72, 21, 71, 16, 30, 58, 14, 8, 35, 5, 18, 33, 45, 19, 51, 41, 10, 11, 59, 73, 32, 40, 56, 26, 1]

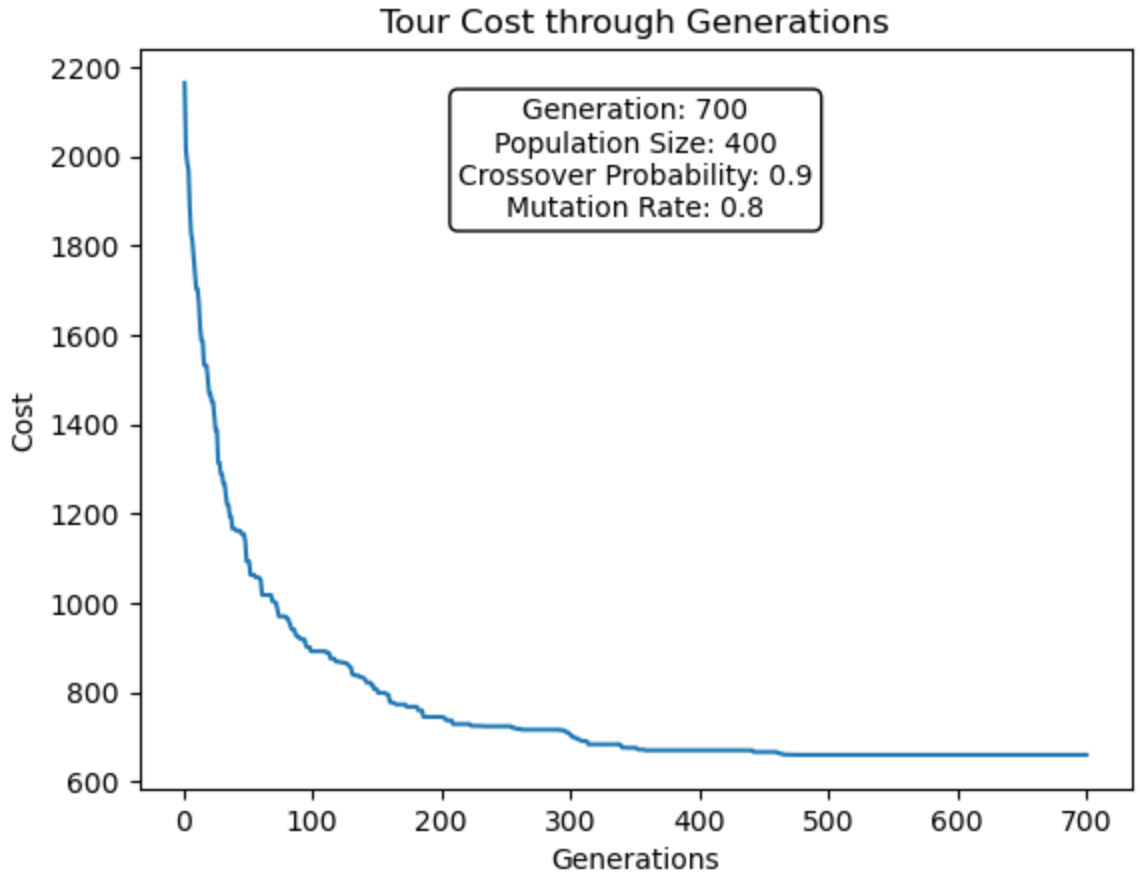


Run 15 - Minimum tour length: 631.86
Run 15 - Best path: [1, 32, 59, 11, 39, 66, 67, 60, 15, 20, 55, 14, 47, 5, 46, 30, 49, 22, 75, 31, 3, 7, 69, 76, 68, 35, 53, 28, 6, 58, 16, 21, 71, 61, 72, 38, 48, 37, 70, 62, 29, 23, 63, 34, 64, 50, 25, 24, 57, 42, 65, 43, 44, 2, 74, 17, 52, 45, 51, 19, 56, 26, 10, 33, 4, 18, 13, 41, 40, 73, 27, 8, 9, 36, 54, 12, 1]

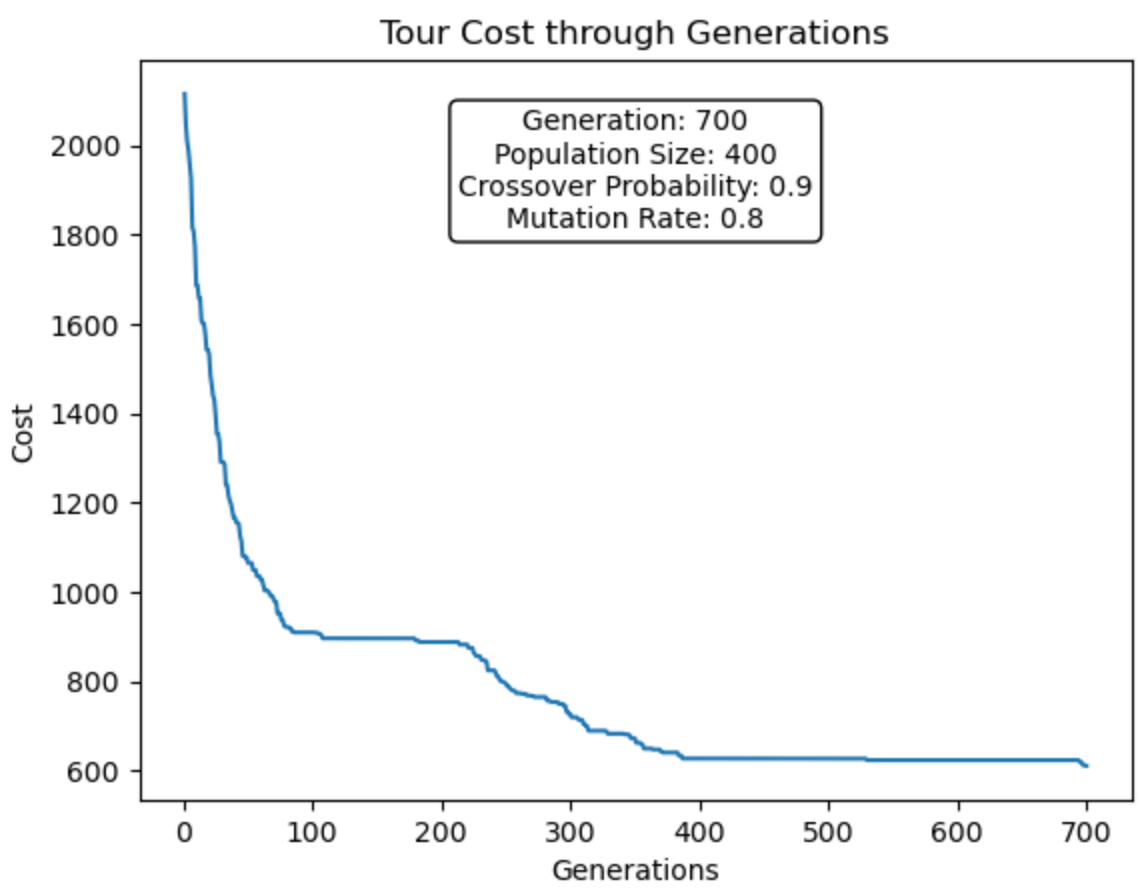


Run 16 - Minimum tour length: 660.45

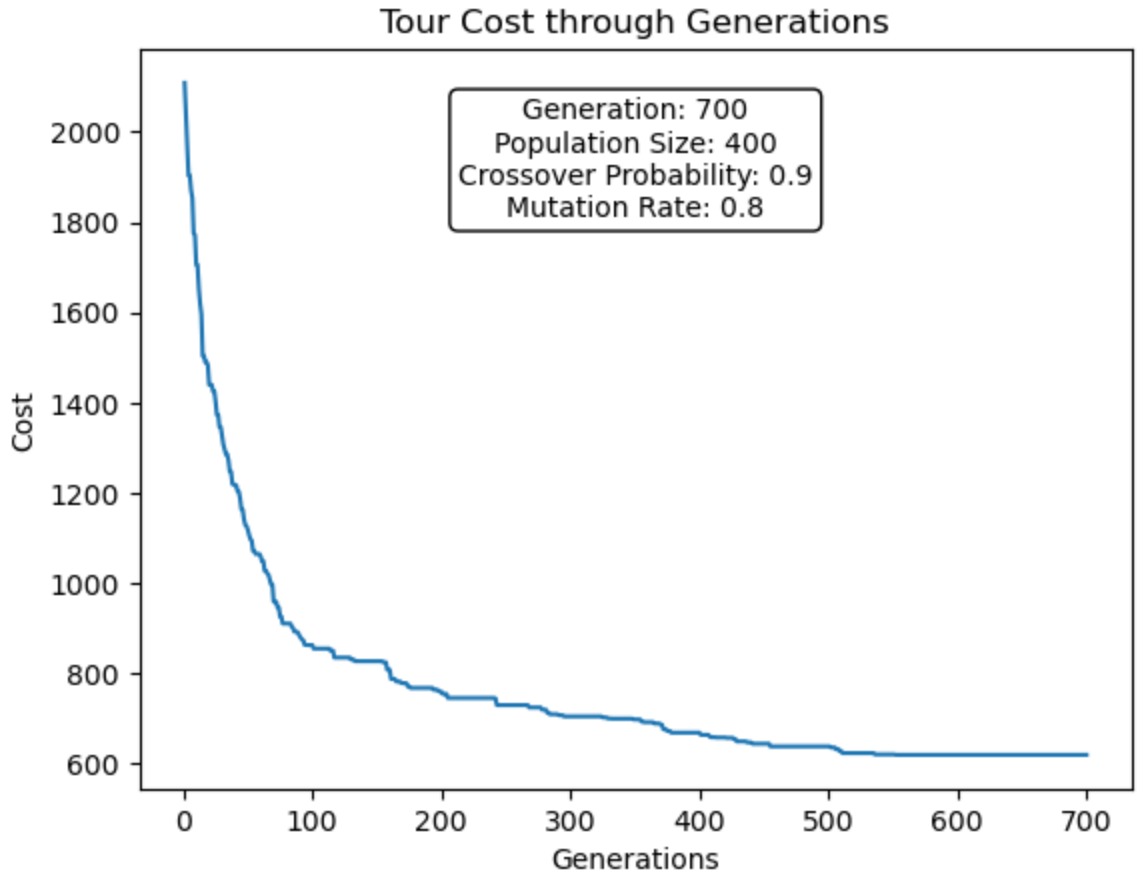
Run 16 - Best path: [1, 27, 68, 76, 69, 5, 35, 47, 53, 46, 30, 28, 14, 55, 58, 16, 6, 4
8, 62, 22, 70, 37, 38, 71, 72, 61, 21, 49, 31, 3, 75, 29, 63, 74, 23, 43, 44, 2, 7, 52,
4, 45, 41, 18, 13, 59, 11, 8, 9, 20, 36, 54, 15, 60, 12, 39, 67, 66, 32, 73, 10, 40, 26,
56, 19, 33, 51, 25, 50, 17, 34, 64, 24, 57, 42, 65, 1]



Run 17 - Minimum tour length: 610.85
Run 17 - Best path: [1, 14, 55, 58, 16, 49, 30, 6, 21, 38, 71, 61, 72, 70, 37, 48, 22, 6
2, 29, 75, 31, 3, 69, 76, 5, 46, 28, 53, 68, 35, 47, 8, 9, 36, 20, 60, 15, 54, 12, 67, 6
6, 39, 59, 11, 73, 40, 10, 32, 56, 26, 33, 45, 4, 18, 27, 13, 41, 51, 19, 50, 17, 52, 7,
34, 64, 2, 74, 63, 23, 65, 43, 44, 42, 24, 57, 25, 1]

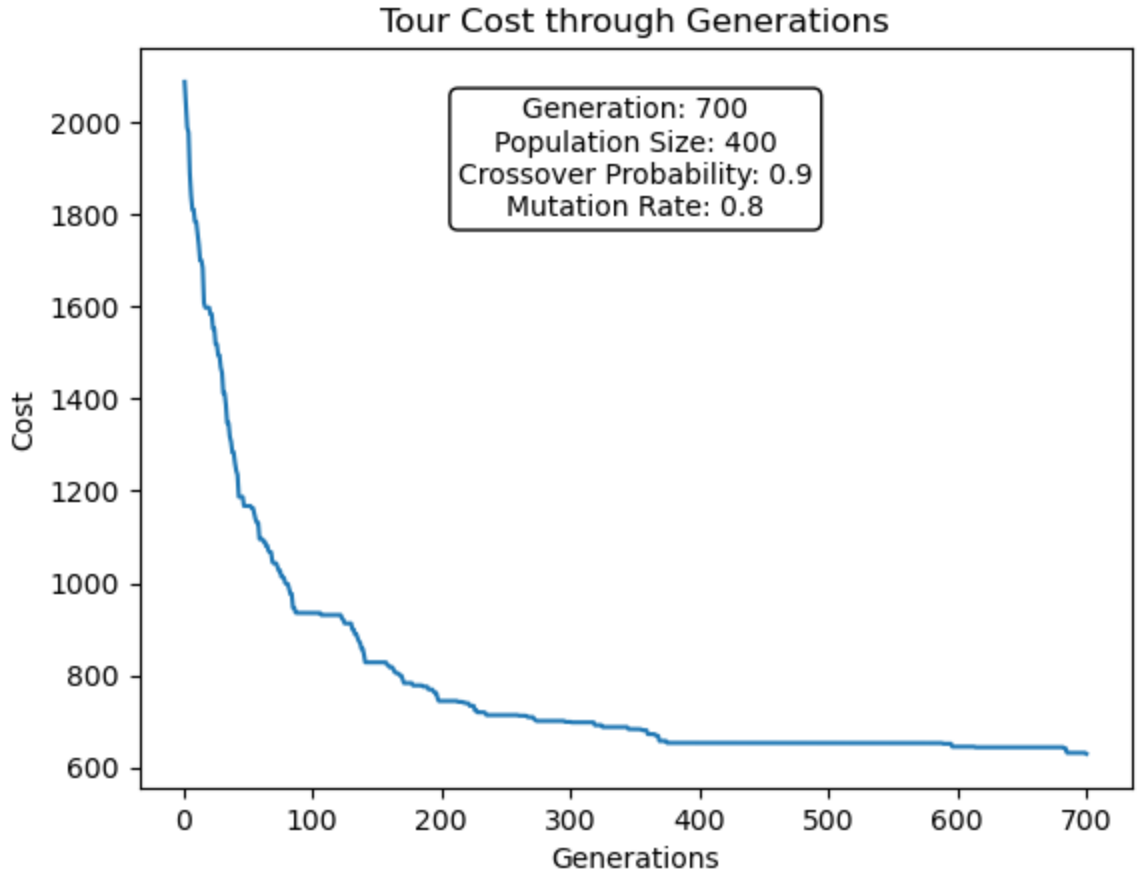



```
Run 18 - Minimum tour length: 619.62
Run 18 - Best path: [1, 69, 3, 31, 49, 30, 46, 28, 53, 35, 5, 76, 68, 47, 9, 20, 55, 14,
58, 16, 6, 72, 61, 71, 21, 38, 70, 37, 48, 22, 62, 29, 75, 74, 2, 43, 65, 23, 63, 7, 52,
18, 41, 13, 59, 32, 10, 33, 45, 4, 17, 64, 34, 44, 42, 57, 24, 50, 25, 19, 51, 56, 26, 4
0, 73, 11, 39, 12, 15, 60, 67, 66, 54, 36, 8, 27, 1]
```

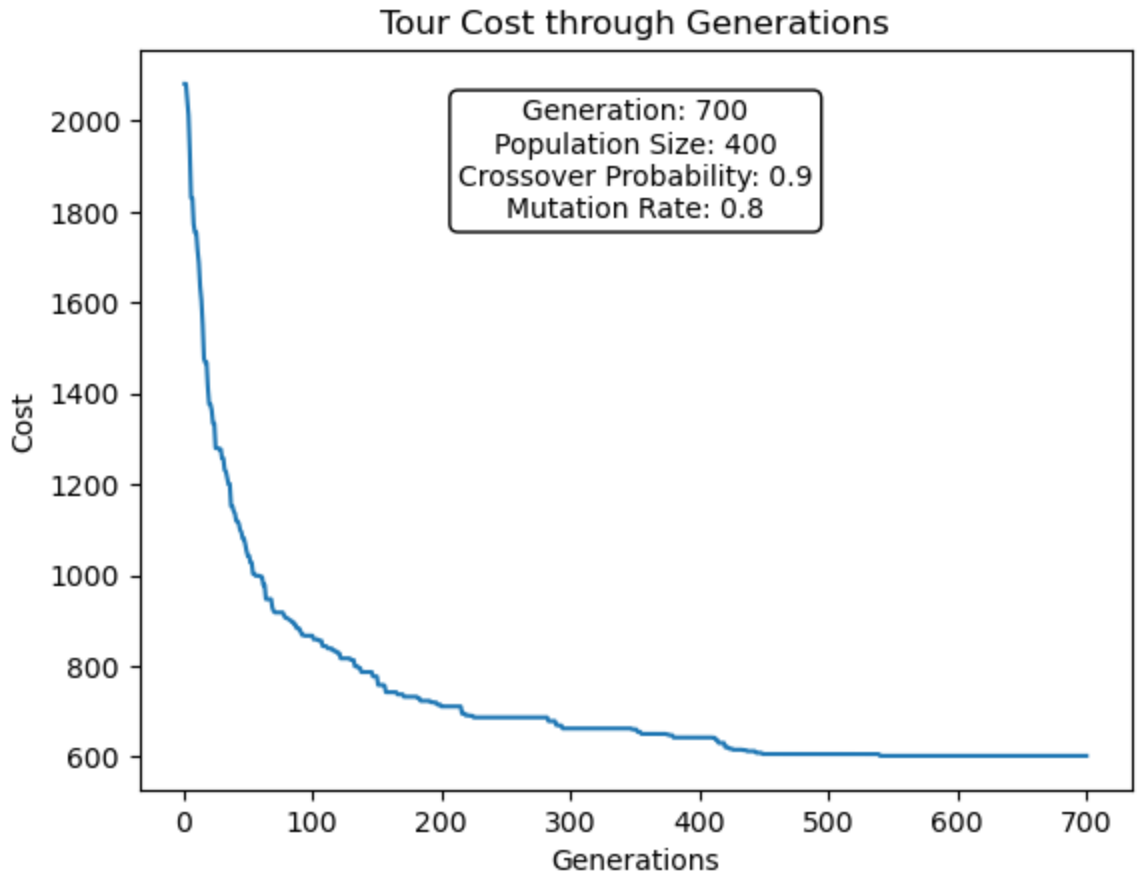



```
Run 19 - Minimum tour length: 631.73
Run 19 - Best path: [1, 43, 44, 42, 57, 24, 2, 74, 63, 29, 23, 65, 62, 22, 6, 38, 37, 7
0, 72, 61, 71, 21, 58, 16, 30, 46, 49, 48, 75, 3, 31, 28, 53, 47, 9, 20, 55, 14, 35, 5,
```

76, 68, 8, 36, 54, 15, 60, 67, 12, 66, 39, 59, 27, 18, 41, 13, 40, 32, 11, 73, 10, 26, 5
6, 19, 51, 33, 45, 4, 52, 50, 25, 17, 64, 34, 7, 69, 1]

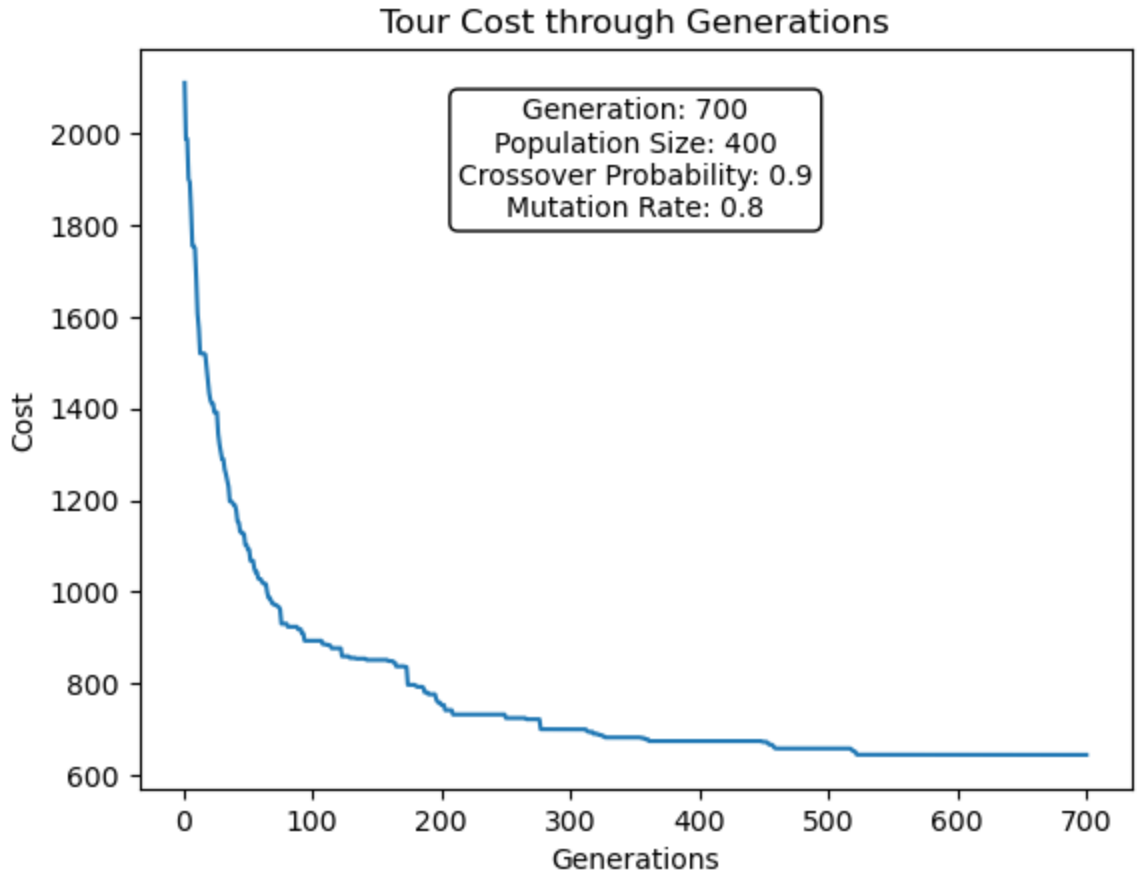


Run 20 - Minimum tour length: 601.95
Run 20 - Best path: [1, 60, 15, 54, 12, 67, 66, 39, 11, 59, 73, 32, 10, 40, 27, 68, 35,
53, 5, 47, 9, 36, 8, 20, 55, 14, 28, 46, 30, 31, 3, 75, 29, 63, 23, 44, 2, 74, 34, 24, 5
0, 25, 19, 56, 26, 51, 33, 45, 4, 41, 13, 18, 76, 69, 7, 52, 17, 64, 57, 42, 43, 65, 62,
70, 22, 48, 37, 72, 61, 71, 21, 38, 6, 49, 16, 58, 1]

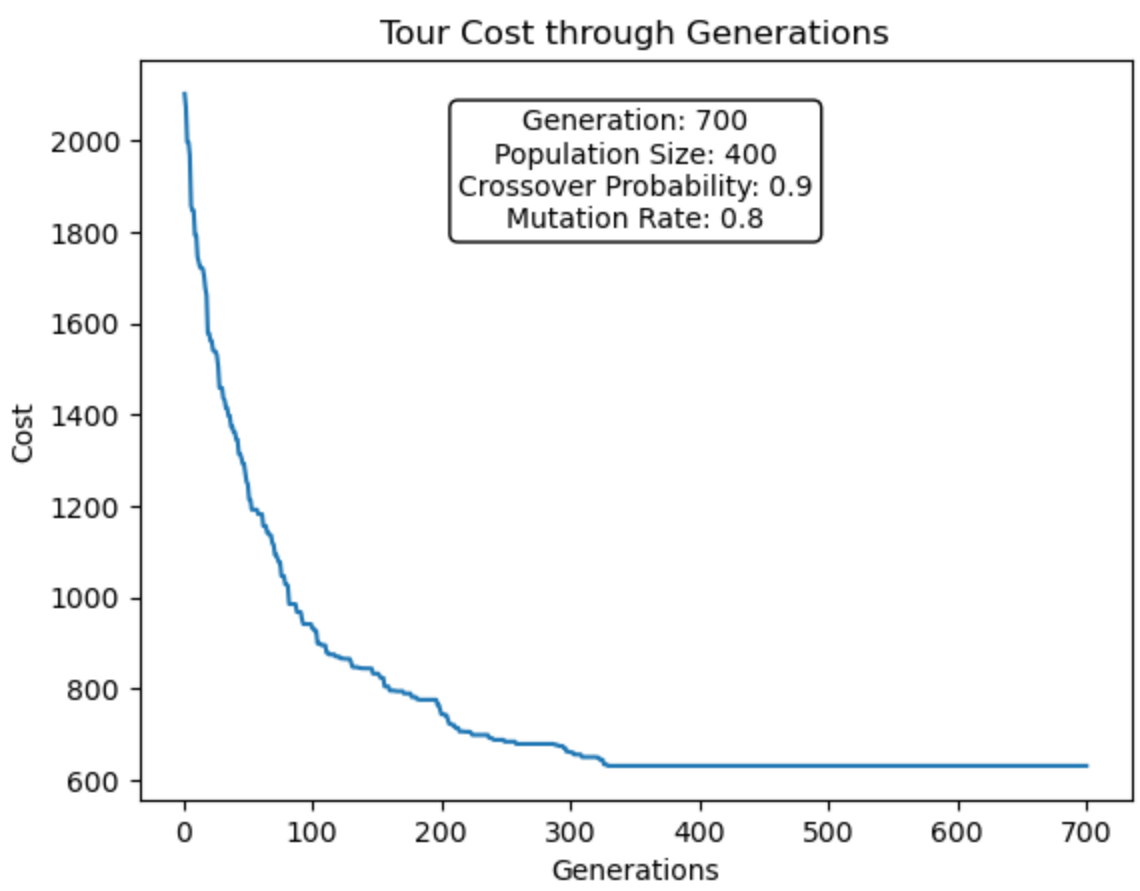


Run 21 - Minimum tour length: 645.08

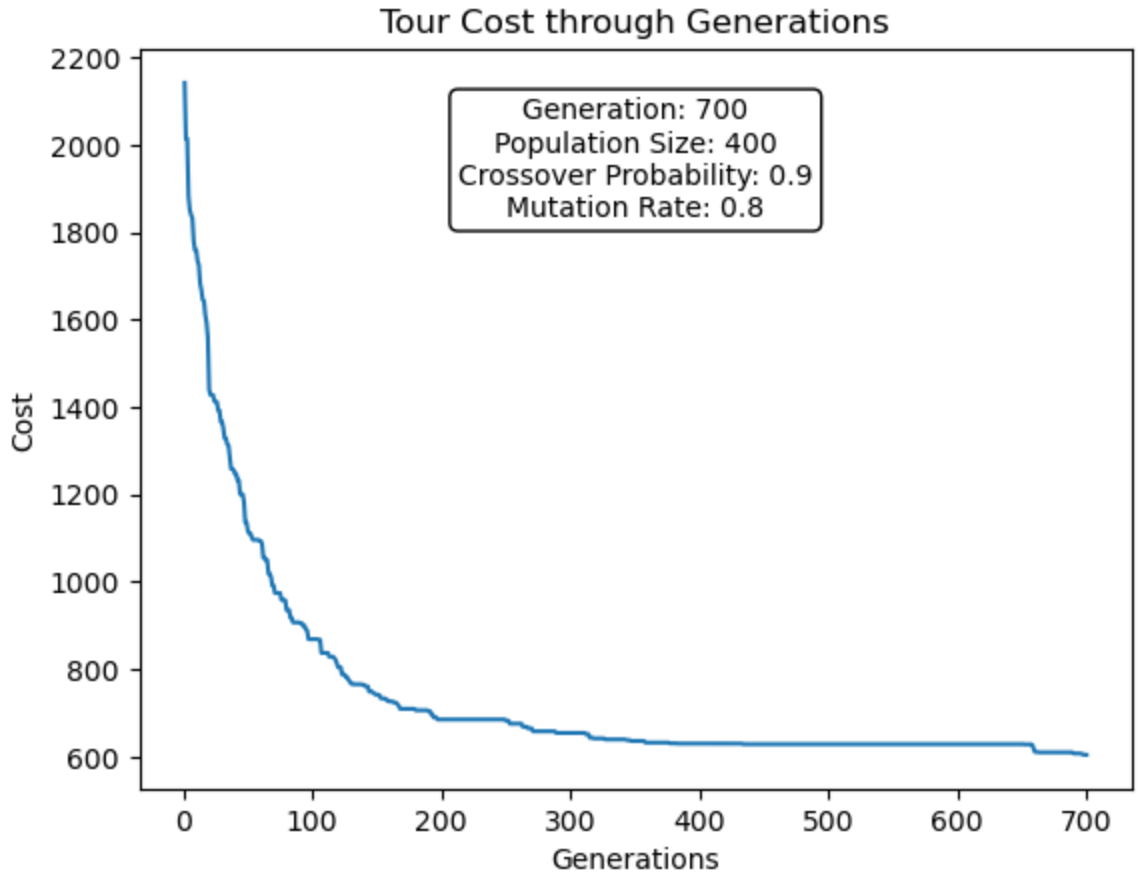
Run 21 - Best path: [1, 2, 64, 34, 74, 63, 3, 7, 69, 76, 5, 68, 47, 35, 9, 20, 15, 54, 3
6, 27, 59, 11, 32, 66, 39, 12, 67, 60, 8, 73, 40, 10, 41, 13, 18, 52, 4, 45, 33, 51, 26,
56, 19, 25, 50, 17, 57, 24, 42, 44, 43, 65, 23, 29, 75, 48, 37, 70, 62, 22, 6, 49, 31, 4
6, 30, 28, 53, 55, 14, 16, 58, 38, 61, 72, 71, 21, 1]



Run 22 - Minimum tour length: 631.70
Run 22 - Best path: [1, 32, 56, 26, 51, 19, 25, 45, 33, 41, 13, 18, 52, 34, 64, 50, 17,
4, 10, 40, 73, 59, 11, 39, 66, 67, 12, 60, 15, 54, 36, 8, 35, 14, 55, 20, 47, 9, 27, 76,
69, 7, 3, 29, 75, 31, 22, 49, 30, 46, 5, 68, 53, 28, 16, 58, 21, 71, 61, 70, 37, 72, 38,
6, 48, 62, 23, 63, 74, 2, 24, 57, 42, 43, 44, 65, 1]

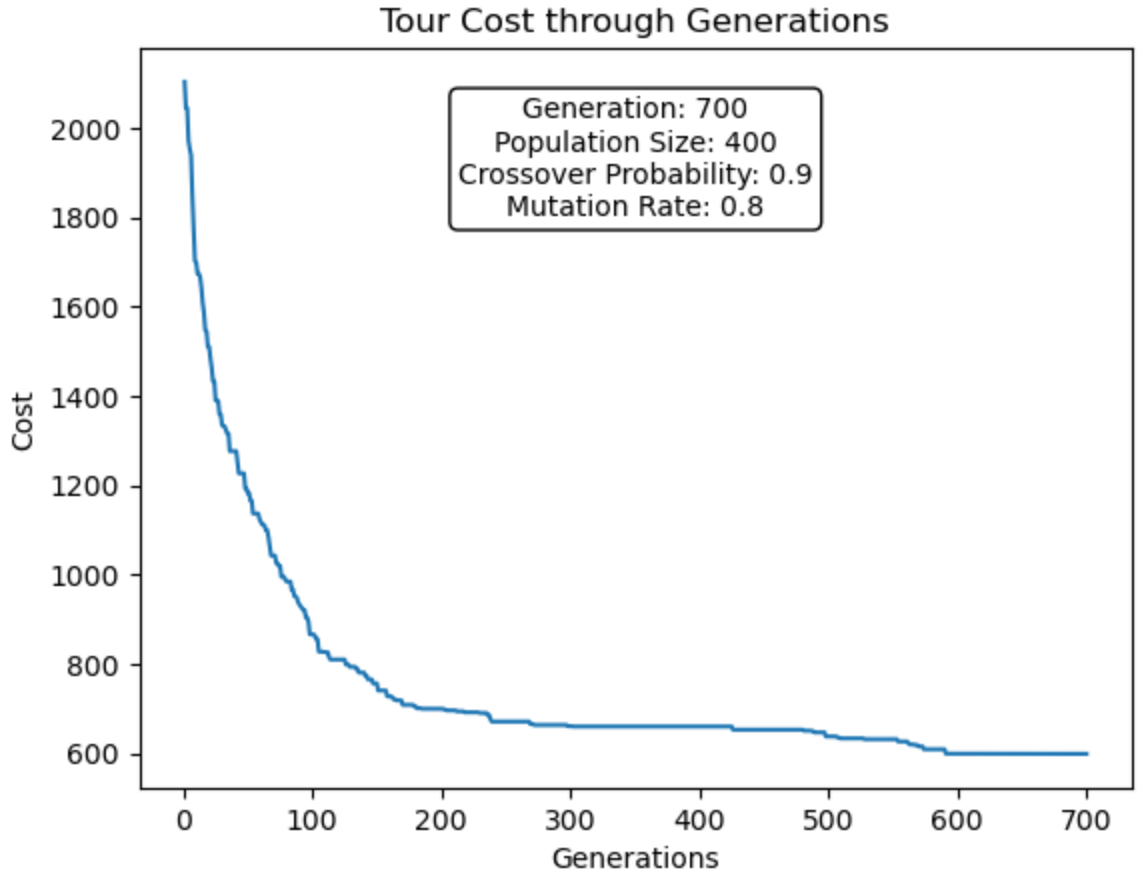



```
Run 23 - Minimum tour length: 604.64
Run 23 - Best path: [1, 19, 51, 25, 50, 24, 57, 2, 44, 42, 43, 65, 23, 62, 70, 22, 37, 7
2, 61, 71, 21, 38, 6, 16, 58, 14, 55, 53, 28, 46, 30, 48, 49, 31, 5, 69, 76, 68, 35, 47,
9, 36, 8, 12, 54, 15, 20, 60, 67, 66, 39, 59, 32, 11, 73, 40, 10, 33, 41, 13, 27, 18, 1
7, 64, 34, 74, 63, 29, 75, 3, 7, 52, 4, 45, 26, 56, 1]
```

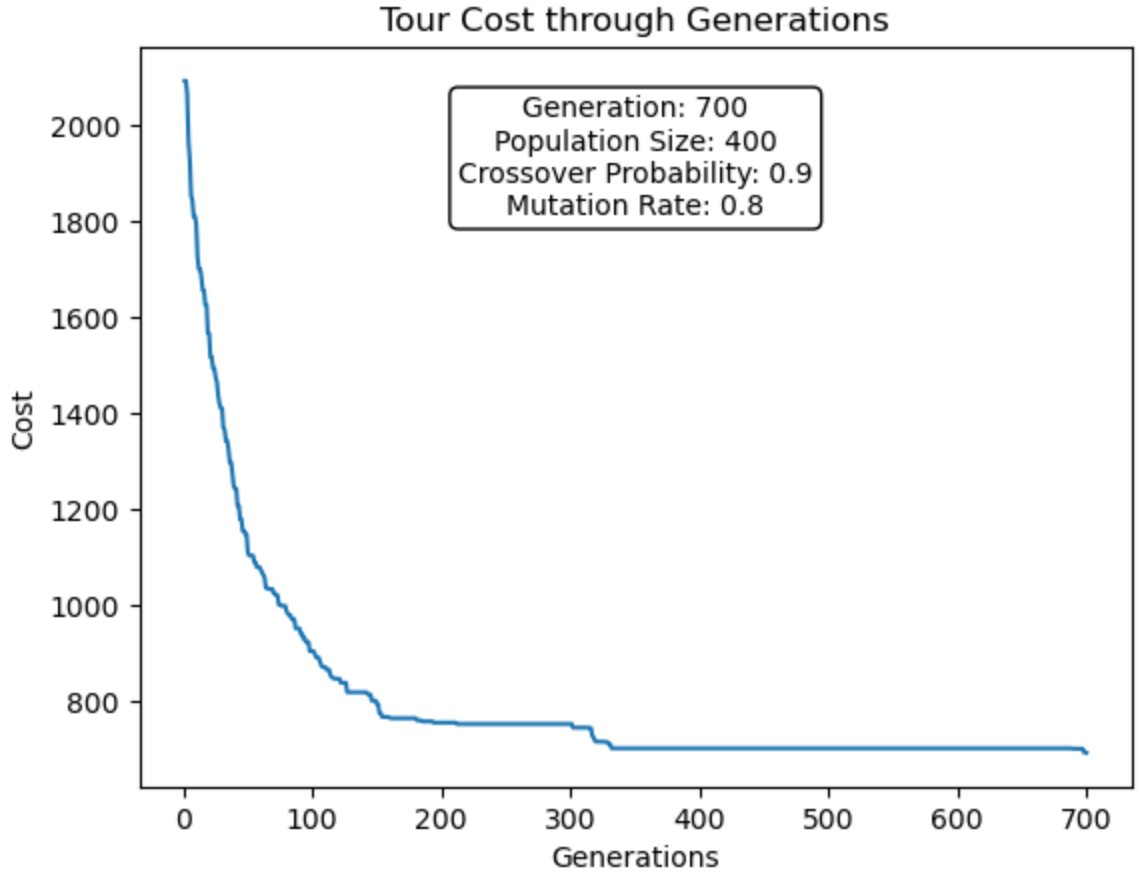



```
Run 24 - Minimum tour length: 599.11
Run 24 - Best path: [1, 55, 14, 58, 16, 38, 21, 71, 61, 72, 37, 70, 62, 29, 63, 23, 2, 3
4, 24, 57, 42, 65, 43, 44, 74, 64, 17, 4, 45, 41, 33, 51, 26, 19, 56, 32, 10, 40, 73, 5
```

9, 11, 39, 66, 12, 67, 60, 15, 54, 36, 20, 9, 8, 27, 76, 69, 7, 3, 31, 75, 22, 48, 6, 4
9, 30, 46, 28, 53, 47, 35, 5, 68, 13, 18, 52, 50, 25, 1]

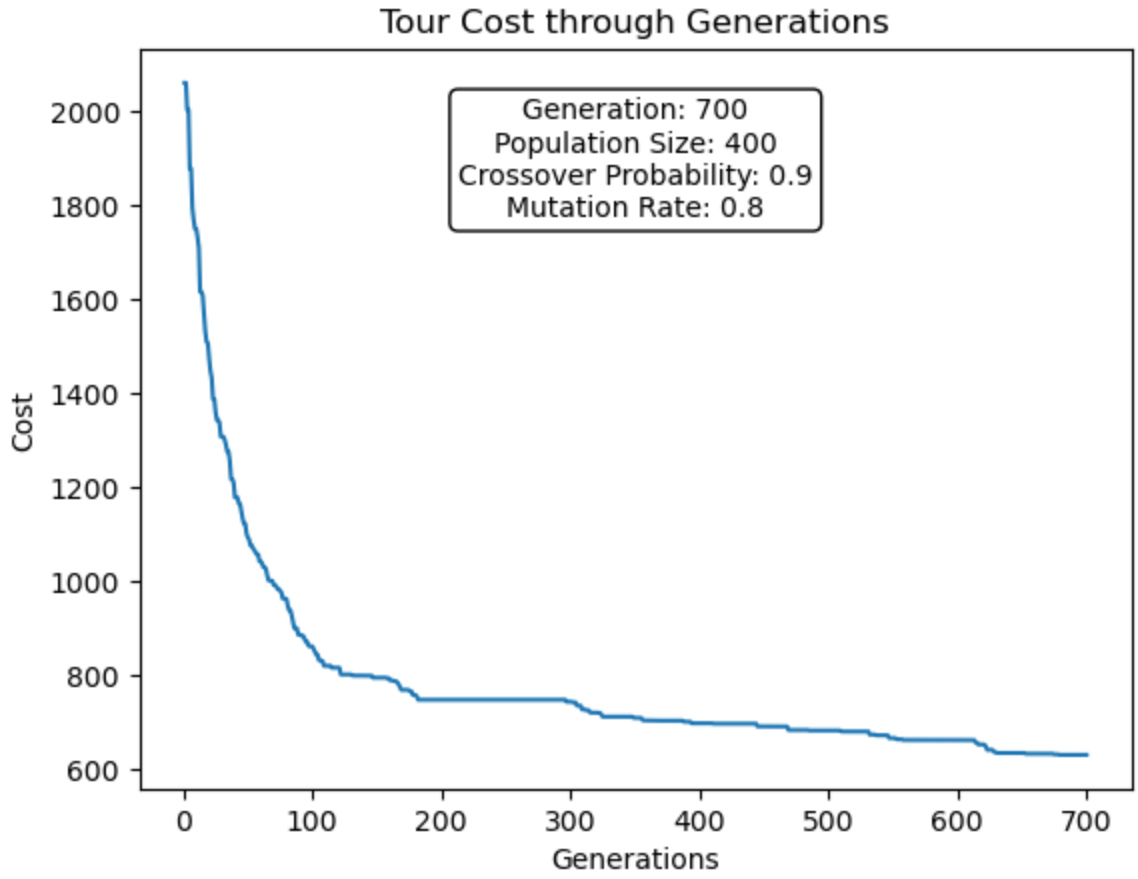


Run 25 - Minimum tour length: 693.70
Run 25 - Best path: [1, 3, 29, 75, 49, 30, 28, 46, 31, 69, 5, 35, 68, 8, 36, 54, 9, 20,
55, 15, 60, 12, 67, 66, 11, 32, 39, 59, 13, 27, 76, 47, 53, 14, 58, 16, 6, 48, 38, 71, 2
1, 61, 72, 37, 22, 70, 62, 23, 65, 42, 2, 34, 74, 63, 64, 24, 44, 43, 57, 25, 50, 17, 4,
45, 10, 33, 18, 7, 52, 41, 73, 40, 19, 51, 26, 56, 1]

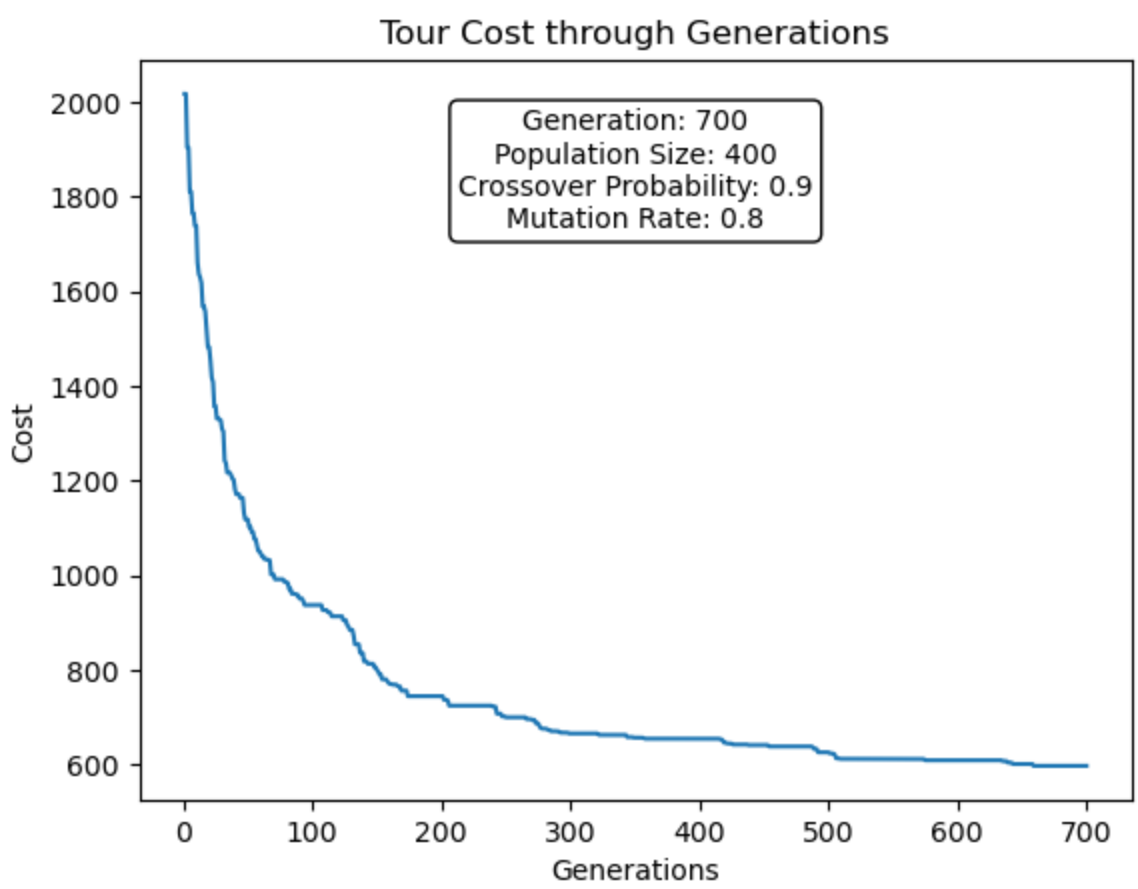


Run 26 - Minimum tour length: 631.93

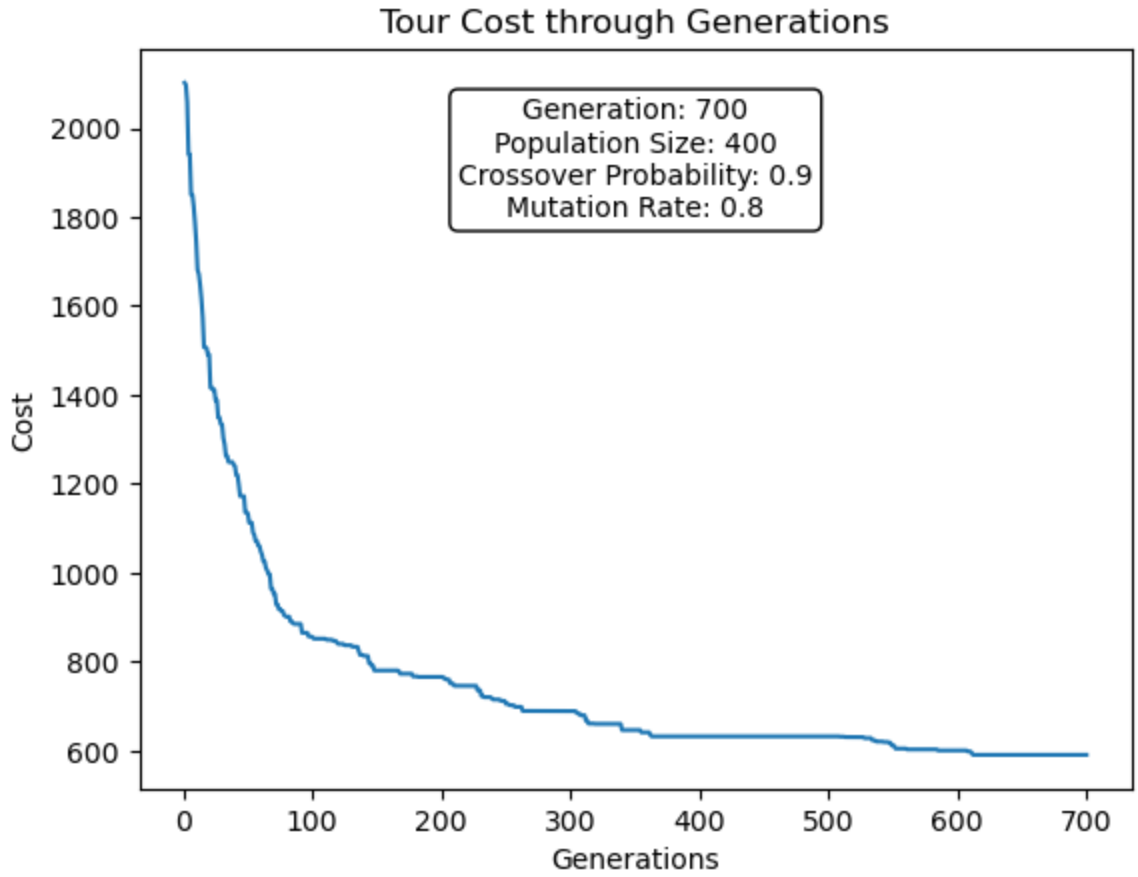
Run 26 - Best path: [1, 69, 7, 52, 34, 17, 64, 24, 50, 4, 45, 33, 10, 51, 26, 56, 19, 25, 57, 65, 42, 44, 43, 74, 2, 23, 62, 29, 63, 3, 31, 6, 38, 21, 71, 61, 72, 37, 70, 48, 22, 75, 49, 30, 16, 58, 14, 55, 20, 8, 36, 9, 54, 15, 60, 12, 67, 66, 39, 59, 40, 13, 27, 68, 5, 46, 28, 53, 47, 35, 76, 18, 41, 73, 11, 32, 1]



Run 27 - Minimum tour length: 597.81
Run 27 - Best path: [1, 70, 22, 62, 23, 63, 29, 75, 3, 31, 69, 76, 68, 5, 46, 35, 47, 9, 54, 12, 67, 60, 15, 36, 20, 55, 14, 58, 16, 21, 71, 61, 72, 37, 48, 38, 6, 49, 30, 28, 53, 8, 27, 13, 73, 59, 11, 66, 39, 32, 40, 10, 41, 18, 4, 45, 33, 51, 26, 56, 19, 25, 50, 57, 24, 64, 17, 52, 7, 34, 74, 2, 44, 42, 43, 65, 1]

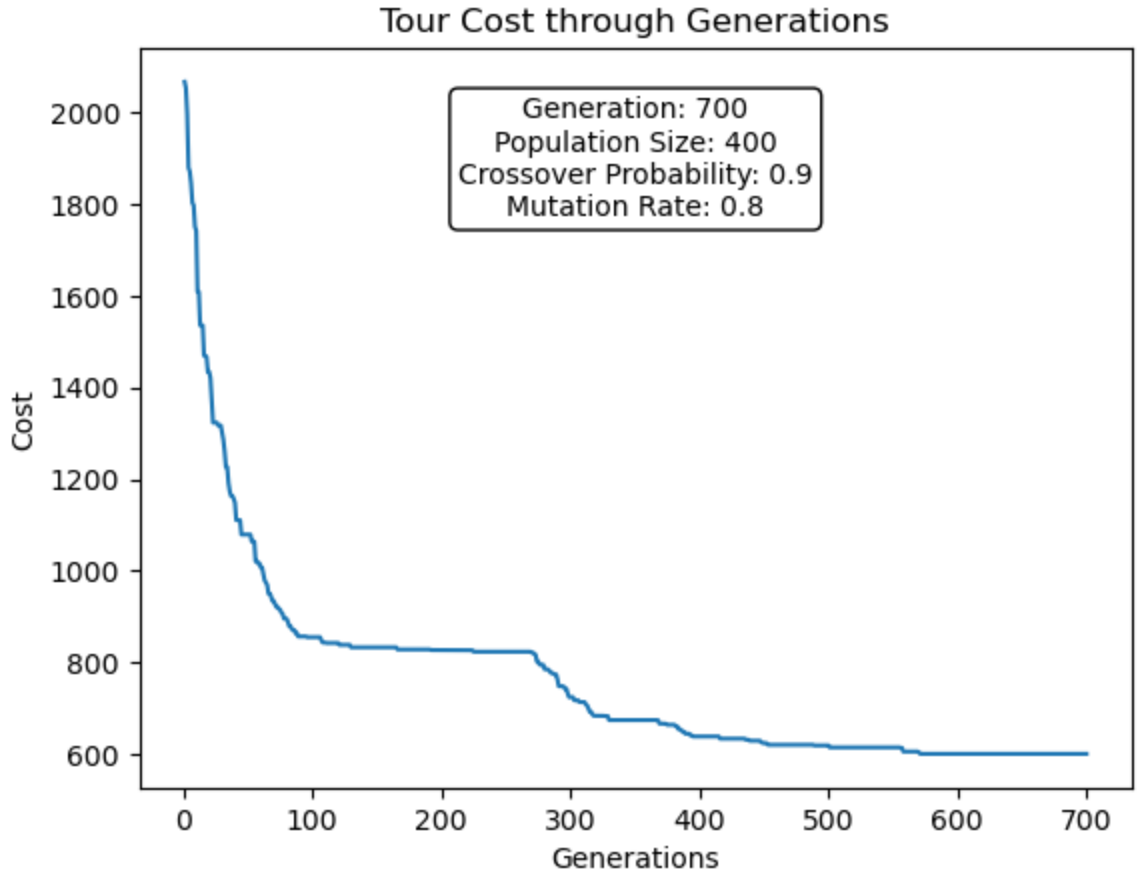



```
Run 28 - Minimum tour length: 590.50
Run 28 - Best path: [1, 32, 11, 39, 12, 66, 67, 60, 15, 54, 8, 36, 20, 55, 14, 58, 16, 3
8, 21, 71, 72, 61, 37, 70, 75, 22, 48, 6, 49, 31, 30, 46, 28, 53, 9, 47, 35, 68, 5, 76,
69, 7, 74, 2, 44, 42, 43, 65, 23, 62, 29, 63, 3, 34, 64, 24, 57, 50, 25, 51, 19, 56, 26,
33, 45, 4, 17, 52, 18, 41, 10, 40, 13, 27, 59, 73, 1]
```

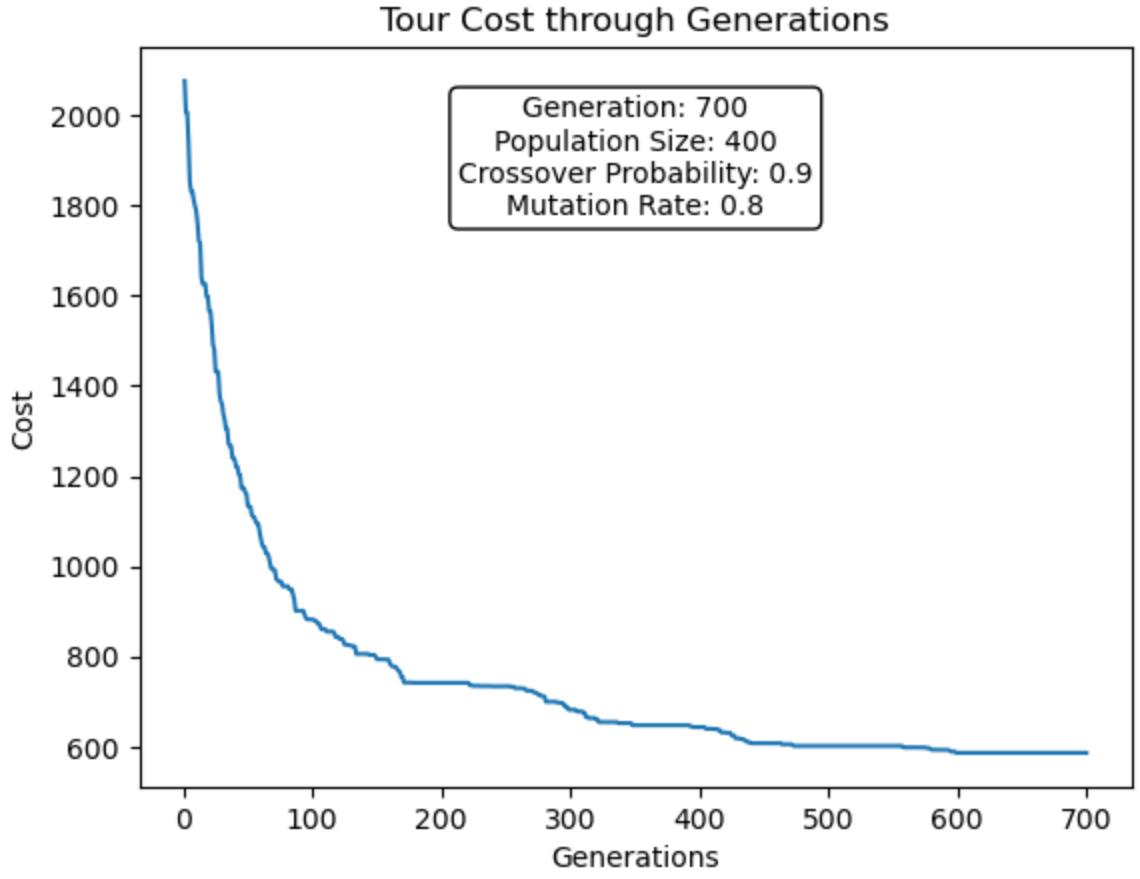



```
Run 29 - Minimum tour length: 601.33
Run 29 - Best path: [1, 25, 4, 45, 33, 10, 40, 73, 41, 13, 27, 68, 18, 52, 64, 17, 50, 2
4, 57, 42, 65, 43, 44, 2, 34, 74, 63, 23, 62, 22, 49, 6, 30, 58, 16, 38, 21, 71, 61, 72,
```

70, 37, 48, 75, 29, 3, 31, 5, 76, 7, 69, 46, 28, 14, 53, 35, 47, 9, 55, 20, 8, 36, 54, 1
5, 60, 12, 67, 66, 39, 11, 59, 32, 26, 56, 19, 51, 1]

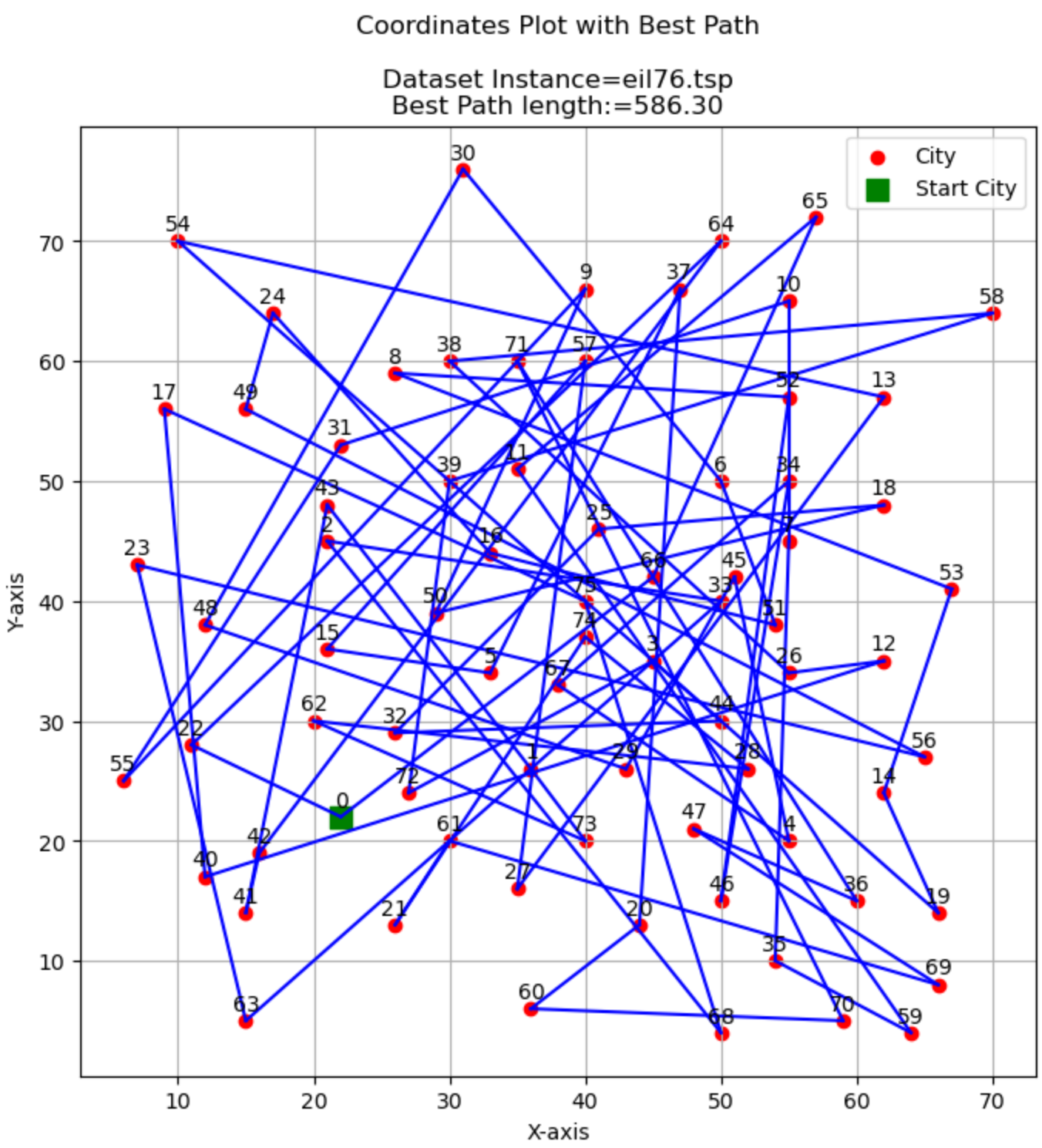


Run 30 - Minimum tour length: 586.30
Run 30 - Best path: [1, 56, 26, 19, 51, 33, 45, 17, 64, 24, 57, 25, 50, 4, 41, 13, 18, 5
2, 76, 69, 31, 3, 63, 29, 75, 22, 49, 6, 30, 46, 28, 58, 16, 38, 21, 71, 61, 72, 70, 37,
48, 62, 23, 65, 43, 42, 44, 2, 74, 34, 7, 5, 68, 27, 47, 35, 53, 9, 14, 55, 20, 36, 8, 5
4, 15, 60, 12, 67, 66, 39, 11, 59, 73, 10, 40, 32, 1]



In [12]:

```
import matplotlib.pyplot as plt
import re

def read_coordinates_from_file(file_name):
    coordinates = []
    with open(file_name, 'r') as file:
        for line in file:
            new_line = re.split(r'\s+', line.strip())
            if new_line[0].isdigit():
                id, x, y = new_line[0], float(new_line[1]), float(new_line[2])
                coordinates.append((x, y))
    return coordinates

# Read coordinates from the file
coordinates = read_coordinates_from_file(file_name)

# Extract x and y coordinates from the list of coordinates
x_coords, y_coords = zip(*coordinates)

# Plot the coordinates
plt.figure(figsize=(8, 8))
plt.scatter(x_coords, y_coords, c='red', marker='o', label='City')

# Annotate each point with its index
for i, (x, y) in enumerate(coordinates):
    plt.annotate(str(i), (x, y), textcoords="offset points", xytext=(0, 5), ha='center')

# Connect the points in the order of the best path indices
for start, end in best_path_indices:
    x_start, y_start = coordinates[start]
    x_end, y_end = coordinates[end]
    plt.plot([x_start, x_end], [y_start, y_end], linestyle='-', color='blue')

# Connect back to the starting point
x_start, y_start = coordinates[best_path_indices[-1][1]]
x_end, y_end = coordinates[best_path_indices[0][0]]
plt.plot([x_start, x_end], [y_start, y_end], linestyle='-', color='blue')

# Mark the starting point with a different marker or color
start_index = best_path_indices[0][0]
x_start, y_start = coordinates[start_index]
plt.scatter(x_start, y_start, c='green', marker='s', s=100, label='Start City')

# Show the legend
plt.legend()

# Show the plot
plt.title(f'Coordinates Plot with Best Path\n\nDataset Instance={file_name}\nBest Path l

plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.grid(True)
plt.show()
```


Coordinates Plot with Best Path
Dataset Instance=eil76.tsp
Best Path length:=586.30
City
Start City
X-axis
Y-axis

In [ ]: