Discrete Optimisation (COMS4050A) Assignment 1

Name: Luke Renton

Student ID: 2540440

Due date: 5 September 2025

Question 1: 2-opt heuristic STSP

Code Implementation

```
"""
Luke Renton (2540440)
Question 1: 2-Opt heuristic for Symmetric Traveling Salesman Problem (STSP)
"""

# Distance matrix from the problem
M = [
    [0,   1.5, 3,   13, 3.5,  4.5, 1.5],
    [1.5, 0,   1.5, 1.3, 13,  13,  2.3],
    [3,   1.5, 0,   1.5, 3,   13,  3],
    [13,  1.3, 1.5, 0,   1.5, 13,  20],
    [3.5, 13,  3,   1.5, 0,   1.5, 3.3],
    [4.5, 13,  13,  13,  1.5, 0,   1.5],
    [1.5, 2.3, 3,   20,  3.3, 1.5, 0]
]

# Initial tour (given): x = (2, 7, 1, 4, 6, 5, 3)
initial_tour = [1, 6, 0, 3, 5, 4, 2]  # 0-based indexing

# Function to calculate total distance of a tour
def tour_cost(tour, dist_matrix):
    return sum(dist_matrix[tour[i]][tour[(i + 1) % len(tour)]] for i in
range(len(tour)))

# 2-Opt implementation
def two_opt(tour, dist_matrix):
    n = len(tour)
    improving_solutions = []
    count = 0

    D = tour_cost(tour, dist_matrix)
    improved = True

    while improved:
        improved = False
        for i in range(n - 2):
            for j in range(i + 2, n if i > 0 else n - 1):
                new_tour = tour[:i+1] + tour[i+1:j+1][::-1] + tour[j+1:]
```

```
                new_cost = tour_cost(new_tour, dist_matrix)
                if new_cost < D:
                    tour = new_tour
                    D = new_cost
                    improving_solutions.append(([city + 1 for city in tour], D))
# 1-based for output
                    count += 1
                    improved = True
                    break
            if improved:
                break
    return [city + 1 for city in tour], D, improving_solutions, count

# Run the 2-Opt algorithm
final_tour, final_cost, improving_solutions, num_improvements =
two_opt(initial_tour, M)

# Print results
print("Final Tour:", final_tour)
print("Final Cost:", final_cost)
print("Number of Improvements:", num_improvements)
print("Improving Solutions:")
for idx, (route, cost) in enumerate(improving_solutions):
    print(f" {idx+1}. Route: {route}, Cost: {cost}")
```

Program output

```
Final Tour: [2, 1, 7, 6, 5, 4, 3]
Final Cost: 10.5
Number of Improvements: 3
Improving Solutions:
 1. Route: [2, 4, 1, 7, 6, 5, 3], Cost: 23.3
 2. Route: [2, 4, 5, 6, 7, 1, 3], Cost: 11.8
 3. Route: [2, 1, 7, 6, 5, 4, 3], Cost: 10.5
```

Question 2: Knapsack Problem (QKP) using binary coded Genetic Algorithm

Code Implementation

```
"""
Luke Renton (2540440)
Question 2: Knapsack Problem (QKP) using binary coded Genetic Algorithm
"""
import random
import math

# Assignment values
v = [7, 6, 13, 16, 5, 10, 9, 23, 18, 12]
```

```
w = [13, 14, 14, 15, 15, 9, 26, 24, 13, 11]
W = 40
P = [
    [0,12,7,6,13,8,11,7,15,23],
    [12,0,15,13,10,15,9,10,8,17],
    [7,15,0,11,16,6,8,14,13,4],
    [6,13,11,0,10,13,14,14,17,15],
    [13,10,16,10,0,9,7,25,12,6],
    [8,15,6,13,9,0,2,13,12,16],
    [11,9,8,14,7,2,0,8,18,4],
    [7,10,14,14,25,13,8,0,9,16],
    [15,8,13,17,12,12,18,9,0,15],
    [23,17,4,15,6,16,4,16,15,0]
]

# GA parameters
N = 30
m = 8
n = 10
max_iteration = 100
pc = 1.0
pm = 1e-3
R = 200

# Seed for reproducibility
random.seed(1000)

# Define a function to calculate the function value f(x)
def obj(x):
    sum_val = 0
    phi = 0
    weight = 0
    # For i=1 to n do → range(0, n) in Python
    for i in range(0, n):
        sum_val = sum_val + v[i] * x[i]
        weight = weight + w[i] * x[i]

    # Add quadratic terms - for i=1 to n, j=i+1 to n
    for i in range(0, n):
        for j in range(i+1, n):
            sum_val = sum_val + P[i][j] * x[i] * x[j]

    if weight > W:
        phi = weight - W
    else:
        phi = 0

    sum_val = sum_val - R * phi
    return sum_val

# Initialize the population set POP with N solutions
pop = [[0 for j in range(n+1)] for i in range(N)]  # N x (n+1) array

# For i=1 to N do → range(0, N)
```

```
for i in range(0, N):
    # For j=1 to n do → range(0, n)
    for j in range(0, n):
        if random.random() < 0.5:
            pop[i][j] = 1
        else:
            pop[i][j] = 0

# Calculate fitness for each solution and store in last column
for i in range(0, N):
    pop[i][n] = obj(pop[i][:n])  # Use first n elements as x vector

# Sort POP with lower to higher function f(x*) at the last column
# (start of main GA)
# For i=1 to max_iteration → range(0, max_iteration)
for iteration in range(0, max_iteration):

    # Store children
    child = [[0 for j in range(n+1)] for k in range(m)]  # m x (n+1) array
    child_count = 0

    # For K=1 to m do (increment K by 2) K=1,3,5,etc → K=0,2,4,6 in Python
    for K in range(0, m, 2):

        # For j=1 to 2 do → range(0, 2)
        for j in range(0, 2):

            # Parent selection n1=0; n2=0
            n1 = 0
            n2 = 0
            while n1 == n2:
                # 1 + floor(random * N) → floor(random * N) in 0-based
                n1 = math.floor(random.random() * N)
                n2 = math.floor(random.random() * N)

            if pop[n1][n] > pop[n2][n]:  # Compare fitness values
                P1 = n1
            else:
                P1 = n2

            # Get second parent
            n1 = 0
            n2 = 0
            while n1 == n2:
                n1 = math.floor(random.random() * N)
                n2 = math.floor(random.random() * N)

            if pop[n1][n] > pop[n2][n]:
                P2 = n1
            else:
                P2 = n2

            # Create x and y vectors from parents
            x = [0] * n
```

```
            y = [0] * n
            temp = [0] * n

            # Copy parent genes
            for l in range(0, n):
                x[l] = pop[P1][l]
                y[l] = pop[P2][l]
                temp[l] = pop[P2][l]

            # Crossover operation
            if random.random() < pc:
                # Generate crossover points n1, n2
                n1 = math.floor(random.random() * n)
                n2 = math.floor(random.random() * n)

                if n1 < n2:
                    for l in range(n1, n2+1):
                        x[l] = y[l]
                        y[l] = temp[l]
                else:
                    for l in range(n2, n1+1):
                        x[l] = y[l]
                        y[l] = temp[l]

            # Mutation operation - For j=1 to n do → range(0, n)
            for gene_idx in range(0, n):
                if random.random() < pm:
                    x[gene_idx] = 1 - x[gene_idx]
                if random.random() < pm:
                    y[gene_idx] = 1 - y[gene_idx]

            # Store children
            if j == 0:  # First child
                for l in range(0, n):
                    child[child_count][l] = x[l]
                child[child_count][n] = obj(x)  # Calculate fitness
                child_count += 1
            else:  # Second child
                for l in range(0, n):
                    child[child_count][l] = y[l]
                child[child_count][n] = obj(y)  # Calculate fitness
                child_count += 1

    # Elitism process - For j=1 to m do → range(0, m)
    # Combine parents and children
    combined_pop = []

    # Add all parents
    for i in range(0, N):
        combined_pop.append((pop[i][:], pop[i][n]))

    # Add all children
    for j in range(0, m):
        combined_pop.append((child[j][:], child[j][n]))
```

```
    # Sort by fitness (descending order)
    combined_pop.sort(key=lambda x: x[1], reverse=True)

    # Keep top N individuals
    for i in range(0, N):
        pop[i] = combined_pop[i][0][:]

# Print results
x_star = pop[0][:n]  # Best solution (first n elements of best individual)

# Calculate final objective value and weight
final_value = 0
final_weight = 0
for i in range(0, n):
    final_value += v[i] * x_star[i]
    final_weight += w[i] * x_star[i]

for i in range(0, n):
    for j in range(i+1, n):
        final_value += P[i][j] * x_star[i] * x_star[j]

print("Best solution x* =", x_star)
print("Objective value f(x*) =", int(final_value))
print("Total weight =", int(final_weight))
```

Program output

```
Best solution x* = [1, 0, 1, 0, 0, 0, 0, 0, 1, 0]
Objective value f(x*) = 73
Total weight = 40
```