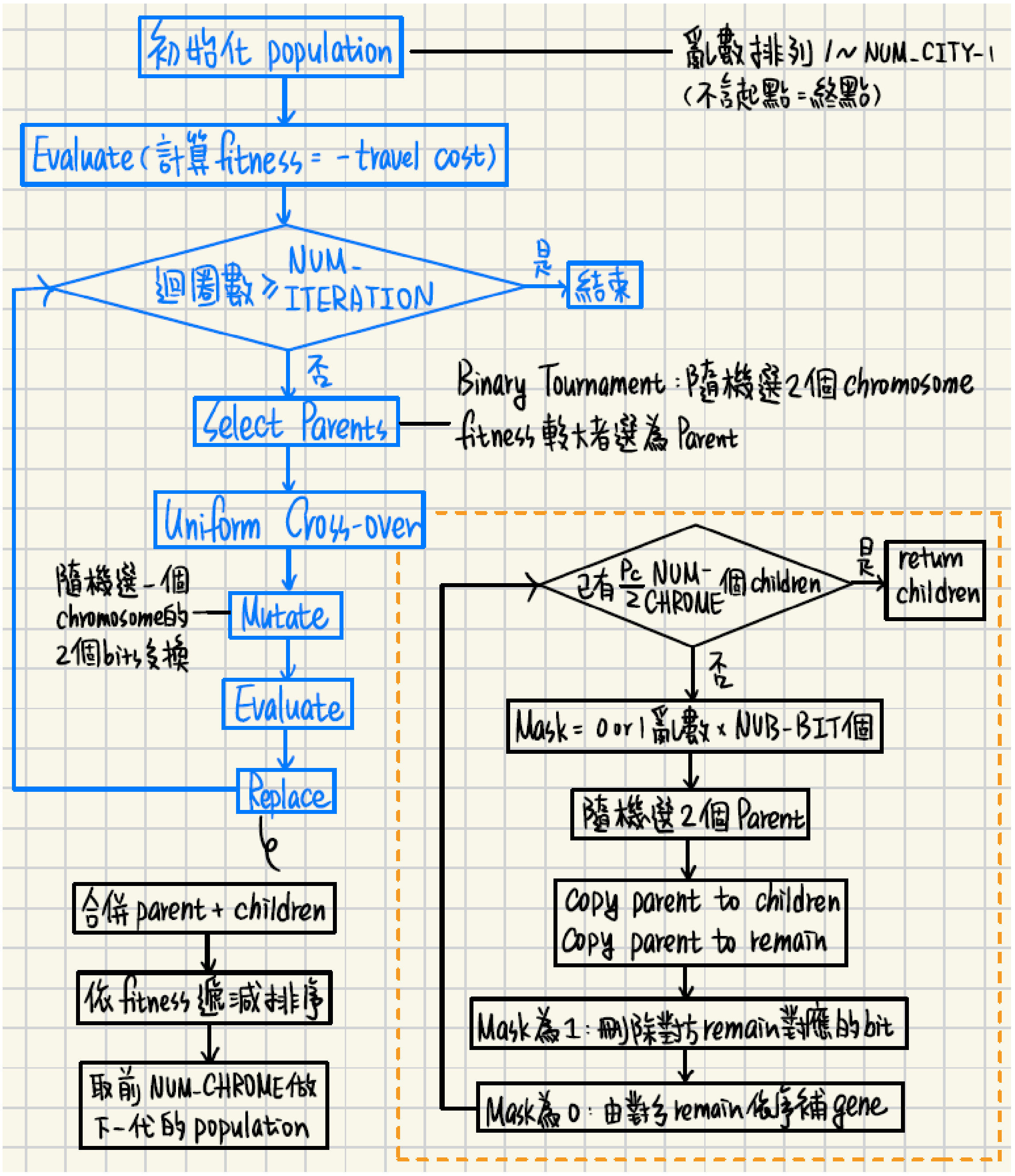
Metaheuristic Algorithm Assignment #1
112701006 運管大三 吳昭泓

1. Genetic Algorithm

a. Flow Chart of GA-TSP



初始化 population
亂數排列 1~NUM_CITY-1
(不含起點=終點)
Evaluate (計算 fitness = -travel cost)
迴圈數 > NUM_ITERATION
是
結束
否
Select Parents
Binary Tournament : 隨機選2個 chromosome
fitness 較大者選為 Parent
Uniform Cross-over
已有 $\frac{Pc}{2}$ NUM-CHROME 個 children
是
return children
否
Mask = 0 or 1 亂數 x NUB-BIT 個
隨機選 2 個 Parent
Copy parent to children
Copy parent to remain
Mask 為 1 : 刪除對方 remain 對應的 bit
Mask 為 0 : 由對方 remain 依序補 gene
隨機選一個 chromosome 的 2個 bits 交換
Mutate
Evaluate
Replace
合併 parent + children
依 fitness 遞減排序
取前 NUM_CHROME 個做 下一代的 population

b. Hamiltonian Cycle

Please refer to the code GA-HC.py. To be more specific, we set the distance of two cities to be 1 if they are connected; big-M = 100 otherwise.

c. Parameter tuning

We use stepwise parameter tuning. First, we adjust the number of iterations. Then, using the previous optimized parameter result, we sequentially tune the population size, crossover rate, and mutation rate. Each experiment is repeated 10 times, with both the average and the best objective value recorded.

The following table shows that increasing the number of iterations leads to better performance. This is intuitive, as more iterations implies more search time. Since the improvement appears to converge, we set the number of iterations to be 300, to strike a balance between performance and efficiency.

number of iterations	20	50	100	200	300	500
average obj.	881.3	633.8	524.9	465.5	406.1	396.2
best obj.	812	317	317	218	218	218
Average CPU time (sec.)	0.012	0.020	0.040	0.079	0.122	0.209

The following table shows that increasing the population size leads to better performance, since larger population size means more diversity at the beginning. Consider the trade-off between the performance and efficiency, we set the population size to be 200.

population size	20	50	100	150	200	300
average obj.	396.2	356.6	307.1	287.3	198.2	158.6
best obj.	218	218	218	218	119	20
Average CPU time (sec.)	0.209	0.309	0.659	1.019	1.417	3.019

The following table shows that the best performance occurs at a moderate cross-over rate = 0.5, which balance the exploration and exploitation.

cross-over rate	0.1	0.3	0.5	0.7	0.9
average obj.	673.4	227.9	198.2	218.0	277.4
best obj.	515	119	119	119	218
Average CPU time (sec.)	0.899	1.159	1.417	1.676	1.922

The following table shows that the best performance occus at mutation rate = 0.02. We infer that low mutation rate lacks diversity, while the search process becomes too random under high mutation. Moreover, the theoretically optimal solution is found.

mutation rate	0.001	0.005	0.01	0.02	0.05	0.07
average obj.	287.3	257.6	227.9	208.1	237.8	406.1
best obj.	119	119	20	20	119	218
Average CPU time (sec.)	1.340	1.967	1.417	1.523	2.636	2.449

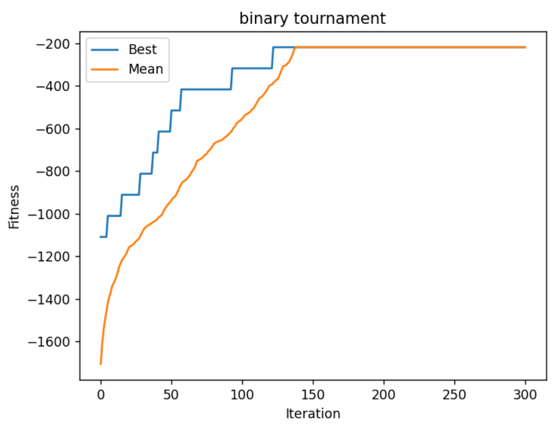
2. Bonus

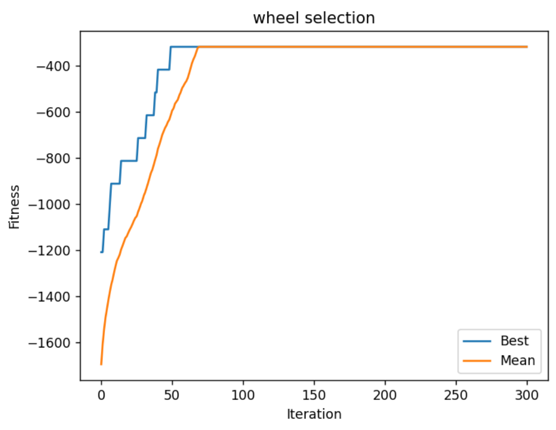
a. wheel selection and comparison

The sub-function is defined in GA-HC.py, with the signature *selection_wheel(p, p_fit)*. The parameter setting is as follows : number of iterations = 300, number of chromes = 200, cross-over rate = 0.5, and mutation rate = 0.02, which is the best setting in problem 1. Note that the fitness value is negative; thus we shift the mininum to 1 in wheel selection.

The following table shows that binary tournament performs better than wheel selection. Comparing the chart below, we infer that low-fitness individuals have an extremely small chance of being selected in wheel selection, leading to too fast convergence.

selection method	binary tournament	wheel selection
average obj.	208.1	287.3
best obj.	20	218
Average CPU time (sec.)	1.523	0.913





b. evaluation by rank and comparison

The sub-function is defined in GA-HC.py, with the signature *calculate_fit_rank(pop_fit)*. In binary tournament, the results of evaluation by fitness is similar to that by rank, as the selection only compares two individuals and chooses the better one. However, under wheel selection, evaluation by rank performs better, since it reduces the impact of extreme fitness values, ensuring that even low-fitness chromes have a reasonable chance of being selected.

selection method	binary tournament		wheel selection	
selection criterion	fitness	rank	fitness	rank
average obj.	208.1	208.1	287.3	247.7
best obj.	20	20	218	218
Average CPU time (sec.)	1.523	1.565	0.913	0.972