# COS314 Assignment 2

## Yi-Rou Hung u22561154

### Generic algorithm:

For the given scenario:

1. Initialization: We will first start by initializing the populations and then represent a potential combination of items to be placed in the knapsack. These solutions are generated randomly from the individuals.
2. Evaluation: We will need to evaluate the population individually by using the fitness function. In the knapsack problem, the fitness function will be assessed from the total value of the individual item compared to the constraint capacity, which is the weight of the item that does not exceed the knapsack’s capacity. If the individual value is greater than the capacity then it will receive a lower fitness score.
3. Selection: We then select two individuals from the population based on their fitness score and then do the crossover function to ‘reproduce’ and create a better individual. The higher fitness score will be likely to be selected first.
4. Crossover: The selected paired individuals are then combined to create new offspring. This is done by using the crossover function, which helps to change the genetics and usually give a better result than before.
5. Mutation: With the new population generated from before, we randomly change the individual by flipping the status of a certain item. This helps to introduce new genetic diversity into the population.
6. Replacement: The new offspring that were created are then added to the generation, but only the best solutions are chosen so the size of the population will remain constant and the best solution is preserved.
7. Termination: The algorithm will continue to iterate through all the steps starting from steps 3-6 until the termination criteria is met. The termination criteria is reaching the maximum number of generations generated.
8. Output: When the termination condition is met, it will return the best generation that was generated.

### Generic Algorithm and Local Search:

The configuration for the Genetic algorithm and Local Search implementation is in the GS\_LS.java file which involves behaviours of both algorithm performances:

Genetic algorithm:

* The GA initializes a population of candidate solutions, represented as binary arrays indicating whether an item is included or not.
* It iteratively evolves the population through selection, crossover, and mutation operations to produce new generations of solutions.
* Fitness evaluation is based on the total value of selected items and adherence to the knapsack's weight limit.

Local Search:

* After running the GA, the LS component further improves solutions by iteratively exploring their neighbourhoods.
* It randomly mutates the current solution and accepts the mutation if it improves fitness, leading to gradual improvement.

GA Parameters:

* Population Size: The number of candidate solutions in each generation.
* Crossover Rate: The probability of crossover between parent solutions.
* Mutation Rate: The probability of mutation in each individual.
* Max Generations: The maximum number of generations for the GA.

Local Search Parameters: Max Iterations: The maximum number of iterations for local search.

Input Data: The input file contains information about the knapsack instance, including item weights, values, and the knapsack's capacity.

### With local search, it explores the solution space by iterating from one solution to a neighbouring solution in search of an optimal or near-optimal solution. Not like the global optimization methods such as genetic algorithms, which explore the entire solution space it focuses on improving a single search. It is effective in exploiting local optima by iteratively improving a single solution through small modifications. The integration of local search into the GA + LS algorithm enhances its effectiveness in finding high-quality solutions to the Knapsack problem by combining global exploration with local refinement strategies.

### The solution comparison is in the table below (question 5)

**Parameters for GA:**

* Population Size (popSize): 100
* Crossover Rate (crossoverRate): 0.8
* Mutation Rate (mutationRate): 0.01
* Maximum Generations (maxGenerations): 1000

**Parameters for GA + LS:**

* Population Size (popSize): 100
* Crossover Rate (crossoverRate): 0.8
* Mutation Rate (mutationRate): 0.01
* Maximum Generations (maxGenerations): 1000
* Maximum Iterations for Local Search: 1000

### 5.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Problem Instance | Algorithm | Seed Value | Best solution | Known Optimum | Runtime (seconds) |
| f1− ld− kp− 10− 269 | GA-LS  GA | 1234  28082529891358361 | [0, 1, 0, 1, 1, 1, 1, 0, 0, 1] which is,  [10&4, 5&32, 4&23, 50&72, 8&80, 87&46]  [0, 0, 0, 0, 1, 0, 0, 0, 0, 0] which is,  [4&23] | 295  23 | 0.009  0.023 |
| f2− l− d− kp− 20− 878 | GA-LS  GA | 12  123 | Line 1, 2, 3, 5, 7, 9  [1, 0, 1, 0, 0, 1, 1, 0, 1, 1] which is,  [44&92, 90&43, 40&68, 75&92, 8&6, 54&44] | 1024  345 | 0.035  0.022 |
| f3\_l-d\_kp\_4\_20 | GA-LS  GA | 23  12 | Line 3  [1, 0, 0, 0] which is,  [9&6] | 35  6 | 0.034  0.025 |
| f4\_l-d\_kp\_4\_11 | GA-LS  GA | 34  344 | Line 4  [1, 0, 0, 0] which is,  [6&2] | 23  2 | 0.034  0.024 |
| f5\_l-d\_kp\_15\_375 | GA-LS  GA | 234 | [1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0] which is,  Item 0: Weight = 0.125126, Value = 56.358532  Item 1: Weight = 19.330423, Value = 80.87405  Item 2: Weight = 58.50093, Value = 47.987305  Item 3: Weight = 35.029144, Value = 89.59624  Item 4: Weight = 82.284004, Value = 74.660484  Item 8: Weight = 9.140294, Value = 36.445206  Item 9: Weight = 14.731285, Value = 16.589863  Item 10: Weight = 98.8525, Value = 44.56923  Item 11: Weight = 11.908322, Value = 0.466933 | 442 | 0.025 |
| f6\_l-d\_kp\_10\_60 | GA-LS  GA | 555  1234 | Line 1, 2, 3, 4, 6, 7, 9, 10  [0, 0, 0, 0, 0, 0, 1, 1, 1, 0] which is,  [5&5, 3&2, 1&1] | 8 | 0.037  0.026 |
| f7\_l-d\_kp\_7\_50 | GA-LS  GA | 666  1298 | Line 1  [1, 1, 0, 0, 0, 1, 0, 1, 0, 1] which is,  [981&983, 980&982, 976&978, 974&976, 485&486] | 4405 | 0.034  0.024 |
| f9\_l-d\_kp\_5\_80 | GA-LS  GA | 999  555 | Line 1,2  [1, 0, 1, 0, 1] which is,  [33&15, 36&17, 12&31] | 63 | 0.035  0.026 |
| f10− l− d− kp− 20− 879 | GA-LS  GA | 222  1000 | Line 1, 3, 10  [1, 0, 1, 0, 1, 0, 0, 1, 0, 0] which is  [91&84, 90&43, 55&44, 75&92] | 1025  263 | 0.036  0.025 |
| knapPI\_1\_100\_1000\_1 | GA-LS  GA | 344 | Line 1, 3, 4, 6, 9, 10 | 9147 | 0.037 |

### 7.

The best solutions found by GA-LS are relatively close to the known optimum values. For example, in problem instance f1\_ld\_kp\_10\_269, the best solution found by GA-LS ([0, 1, 0, 1, 1, 1, 1, 0, 0, 1]) has a known optimum of 295. The difference between the best solution's fitness (269) and the known optimum (295) is reasonable, indicating a good approximation. Comparing the results of the Genetic Algorithm with Local Search (GA-LS) to the results of the Genetic Algorithm alone can provide insights into the effectiveness of incorporating local search. If GA-LS consistently outperforms GA in terms of solution quality (closeness to the known optimum) and runtime, it suggests that the local search heuristic helps improve the algorithm's performance. Cross different problem instances, GA-LS demonstrates robustness and scalability. For example, in problem instances f2\_ld\_kp\_20\_878 and f10\_ld\_kp\_20\_879, which have larger problem sizes and capacities, GA-LS consistently finds solutions close to the known optima. This indicates that GA-LS is effective across a range of problem complexities.

The runtime of GA-LS executions varies across problem instances but generally remains reasonable. For instance, in problem instance f1\_ld\_kp\_10\_269, GA-LS achieves a runtime of 0.023 seconds, which is relatively efficient considering the complexity of the problem. However, runtime efficiency can be influenced by factors such as problem size and algorithm parameters.

GA-LS is effective in finding near-optimal solutions for knapsack optimization problems. By incorporating local search heuristics, GA-LS demonstrates improved solution quality compared to the Genetic Algorithm alone while maintaining reasonable runtime efficiency.