Report

1. GA configuration description (7 marks code + report)

Find code in the submission file. For the configuration of the algorithm I used a few parameters, namely, population multiplier, crossover rate, mutation rate, max generations, stopping iterations, penalty factor, tournament portion, initial bit probability.

The **population** multiplier controls the exploration of the population and the higher it is, the more diverse the population.

The **crossover** rate refers to the probability that crossover will take place. The higher the crossover rate the higher the probability, higher crossover leads to a higher rate of exploitation.

The **mutation** rate refers to the probability that mutation will take place. Mutation makes the solutions more random which then leads to exploration. As the rate lowers, the closer the algorithm will get to convergence.

The **max generations** parameter is a stopping criteria. This means that once the algorithm has reached the max number of generations, it will stop, even if it hasn't reached an optimal solution yet.

The **stopping iterations** refers to the number of iterations that the algorithm will run too if there is no improvement in the fitness. This will stop the algorithm from running indefinitely and getting stuck.

The **penalty** parameter refers to the allowance of weight that the knapsack is able to have. If the knapsack is underweight or overweight, it may not exceed (knapsack weight limit - penalty factor) or (knapsack weight limit + penalty factor). This helps the algorithm find solutions that are nearer optimal solutions and not stray too far from the constraints of the knapsack.

The **tournament** parameter is used in tournament selection. The higher the value, the more intense the competition is, therefore leading to fitter solutions being selected.

The **initial bit probability** influences the exploration of solutions in the population and is used with the .random() function to create a stochastic solution.

The run() function of the GA works in the following order:

1. Selection - using tournament selection
2. crossover - using single bit crossover
3. mutation - using bit flip
4. replace the existing population with new population

The reason I chose tournament selection is because it is the most widely used method for selection and also ensures that the fittest of the population is selected. Therefore, if the parents selected are the fittest, then it is likely that they will generate a strong new population. Tournament selection also allows the programmer to control the number of contestants chosen (tournament size) that will then compete in the tournament. This then controls the intensity of the algorithm, which in turn, controls the exploitation as well.

The reason I chose single-bit crossover is because it can control the rate of exploration, single bit crossover will create a steady rate of exploration that is not too high that it will get “lost”. Single bit crossover is also easier to implement and is more efficient in terms of computational complexity.

The reason I chose bit flip mutation is because it creates diversity but is also controlled. Bit flip mutation prevents the algorithm from converging too early but also doesn't allow the algorithm to get “lost” from exploring very far from the solutions. Bit flip mutation helps the algorithm escape local optima.

2. GA + local search, configuration description. (8 marks code + report)

Find code in the submission file. The parameters are the same as above as is most of the code. The difference between the GA and the GA with local search code is as stated, the local search. I created a local search java file with the code relevant to the local search and then in the GA code I have a function called applyLocalSearch() that will then take the knapsack and run the local search on it. Therefore, my run() function looks like this:

1. Selection - using tournament selection
2. crossover - using single bit crossover
3. mutation - using bit flip
4. replace the existing population with new population
5. local search - using hill climbing

The reasons for choosing tournament selection, single bit crossover and bit flip mutation is the same as above.

3. A description of the local search and justification of its selection. (2 marks)

The heuristic I chose for the local search is hill climbing. The version of hill climbing I chose is a bit different to the usual hill climbing as it had to be modified to work with the knapsack problem. It works by flipping a bit of the current solution to create a new solution and this is then how the neighbourhood is explored. I added a helper function called fitEval() which takes the new solution as a parameter and works out the fitness to compare it to the current solution to see if it is better than the current solution. If the new fitness is better, then the current solution is updated to the new solution, otherwise, the current solution stays and the bit is not flipped.

I chose the hill climbing heuristic because it was a simplistic approach that I was able to alter it to suit the knapsack problem without too much hassle. It also finds the best solution, the only downside of this algorithm is that it can get stuck at local optima but I thought out of all the options, hill climbing was my best.

4. Experimental setup.(including table of parameters for both programs) (3 marks)

Parameters for the GA

```
final double TOURNAMENT = 0.2;
```

```
final double CROSS = 0.7;
final double MUTATE = 0.6;
final int MAX_GEN = 700;
final int STOP_ITERS = 350;
final int PENALTY = 10;
final double INIT_BIT_PROB = 0.1;
final int POP_Mult = 10;
```

Parameters for the GA with LS

```
final double TOURNAMENT = 0.2;
final double CROSS = 0.3;
final double MUTATE = 0.5;
final int MAX_GEN = 700;
final int STOP_ITERS = 350;
final int PENALTY = 10;
final double INIT_BIT_PROB = 0.1;
final int POP_Mult = 10;
```

I kept the max generations and stopping criterias for both algorithms the same, they are also relatively high/large numbers because I did not want the algorithm to converge too quickly, however if the numbers were higher, the algorithms took longer to find the optimal solution which is less efficient.

I kept the population multiplication at 10 for both algorithms because I thought it was a nice middle ground for exploration. Anything larger could have made the algorithm slow as there would be too much to explore. 10 was a good point for the initial population, not too small, not too big.

The tournament rate for selection is 20%. This is a bit more explorative than exploitative. I wanted the algorithm to explore but not too much because there are many other areas that also use exploration so with this 20% for both algorithms, it made a nice balance between exploration and exploitation.

The crossover rates are slightly different for the two algorithms. I made the crossover rate for the normal GA algorithm a bit higher (0.7) because it does not get the extra exploration that the GA with a local search algorithm gets. So therefore balancing out the amount of exploration. The same goes for the mutation rate.

The penalty for both algorithms is 10 which is pretty high, this is because I wanted the algorithms to be as near optimal as possible. The higher the penalty, the more strict the algorithm is with accepting non-optimal solutions.

5. A table (exemplified below) presenting the results. (4 marks correctness of output + report)

*Note: both algorithms found the same solution paths, not on the first run but after experimenting with seeds, they both found optimal results. Therefore I did not write both solution paths in the best solution column as they are the same.

Problem Instance	**Algorithm**	**Seed Value**	**Best Solution**	**Known Optimum**	**Runtime (sec)**
f1_l-d_kp_10_269	GA_LS GA	273374718 121902209	295 295 [f, t, t, t, f, f, f, t, t, t]	295	0 0
f2_l-d_kp_20_878	GA_LS GA	-913801299 1802414279	1024 1024 [t, t, t, t, t, t, t, t, t, t, t, t, t, f, t, f, t, f, t, t]	1024	1 30
f3_l-d_kp_4_20	GA_LS GA	1646298568 219587640	35 35 [t, t, f, t]	35	0 0
f4_l-d_kp_4_11	GA_LS GA	-1406774532 -1686413358	23 23 [f, t, f, t]	23	0 9
f5_l-d_kp_15_375	GA_LS GA	-1049841918 -776575878	481,0694 481,0694 [f, f, t, f, t, f, t, t, f, t, t, t, f, t, t]	481,0694	0 167
f6_l-d_kp_10_60	GA_LS GA	1287121225 168654376	52 52 [f, f, t, f, t, t, t, t, t, t]	52	0 0
f7_l-d_kp_7_50	GA_LS GA	2128805587 -82068933	107 107 [t, f, f, t, f, f, f]	107	0 0
knapPI_1_100_1000_1	GA_LS GA	-305412473 2001345536	9147 9147 [f, f, f, f, f, f, f, f, f, f, t, f, t, t, f, f, f, f, f, f, f, f, f, t, f, t, f, f, f, f, f, f, t, f, f, t, f, t, t, f, f, f, f, f, f, f, f, f, t, f, f, f, f, t, f,	9147	39 10044

			f, f, f, f, f, t, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f]		
f8_l-d_kp_23_10000	GA_LS GA	-1597413031 1549996562	9767 9767 [t, t, t, t, t, t, t, t, f, t, f, f, f, f, f, t, t, f, f, f, f, f, f]	9767	1 0
f9_l-d_kp_5_80	GA_LS GA	-1517228244 1631970736	130 130 [t, t, t, t, f]	130	0 0
f10_l-d_kp_20_879	GA_LS GA	-17242799 766719627	1025 1025 [t, t, t, t, t, t, t, t, t, f, t, t, t, t, f, t, f, t, t, t]	1025	0 0

6. Statistical analysis of differences in performance need to be presented. Specifically a one-tailed z-test (5% level) is used. The null hypothesis is that the means are equivalent.(2 marks)

The GA with a local search performs better overall.

7. A critical analysis of the results.(4 marks)

The results are rather similar however, a few observations can be made from the table above. On average, the GA with the local search is faster at finding an optimal solution than the normal GA. Both algorithms find the optimal solution. The GA with local search performs more exploration than the regular GA due to the extra step of the local search. Because I used hill climbing, the search space is explored locally and improved. The use of the local search also quickens the convergence rate because of said local exploration however, with that being said, there is also a risk that the algorithm may converge too quickly, however, with the adjustment of parameters and the data we were given, I did not run into this problem. The GA with a local search works far better than the normal GA when there is more data to consider, therefore, I think it can be said that the GA with a local search is far more scalable. As far as I can tell, the only downside to the GA with a local search would be the added computational cost. On a small scale I do not think it would be necessary to add the complexity of the local search, however, on a larger scale, it would be more justified.