Assignment 2

COS 314

Mr. TM Whitaker
U22744968

Question 1:

The Genetic Algorithm (GA) in this code is configured with the following parameters:

- ***populationSize***: The number of chromosomes in the population. Set to 100. Population size was chosen due to this Research Gate (Research Gate, n.d.) paper
- ***selectionSize***: The number of chromosomes selected for the tournament selection process. Set to 4. This size was chosen in conjunction with the lecture slides.
- ***maxGenerations***: The maximum number of generations the GA will run for. Set to 5, but it's multiplied by ***numItems*** of the Knapsack if local search is not enabled. Originally this was set to 100 in conjunction with this Research Gate (Research Gate, n.d.) paper but I adapted this to work with the ***numItems*** as smaller Item sets need less generations to find optima when compared to larger item sets.
- ***crossoverRate***: The probability of crossover occurring. Set to 0.85. This rate was obtained from this paper (AG-knapsack, n.d.) where multiple rates were tested, this value worked well so I kept it.
- ***baseMutationRate:*** The base mutation rate used in conjunction with the ***numItems*** in the Knapsack. Set to 0.03.
- ***maxMutationRate:*** The max possible Mutation Rate. Set to 0.6.
- ***mutationRate***: The probability of mutation occurring. Set to the min(***baseMutationRate*** * ***numItems***, ***maxMutationRate***). Originally, I had a rate of 0.15 as in the lecture slides but found this to be too low for Knapsacks where there were lots of items. I therefore made it based on the ***numItems*** to encourage exploration in large item sets.

The GA uses a binary representation for the chromosomes, where each gene is a boolean indicating whether the corresponding item is included in the knapsack or not. The fitness of a chromosome is the total value of the included items, or 0 if the total weight exceeds the maximum weight.

The GA uses tournament selection to select parents for crossover, where ***selectionSize*** chromosomes are randomly selected and the one with the highest fitness is chosen this is done twice to obtain two parents. Crossover is performed at a single random point on both parents to create two children, each child then has a chance to mutate whereby a random gene bit is flipped. This is repeated to create each new population.

The GA runs for ***maxGenerations*** generations, in each of which a new population is generated through crossover and mutation, and the fitness of the population is updated. The best chromosome after each generation is stored and printed to the console.

If the fitness of the best chromosome after a generation is 0, a new chromosome with all genes set to false is added to the population, and a random chromosome is removed. This is intended for Knapsacks with many items of high weight, as it introduces less items into the chromosomes allowing for more chromosomes under the weight limit and therefore allowing a good solution to be found.

Question 2:

The GA with local search is configured with the same parameters as the GA without local search, but with the addition of a local search operation that is performed on each child after crossover and mutation. The actual implementation of the algorithm is spoken about more in Question 3.

If local search is enabled, maxGenerations is set to ***numItems***, which means the GA will run ***numItems*** times. This is because local search can potentially find good solutions more quickly than the GA alone, so fewer generations may be needed.

Question 3:

The local search implemented in this code is a form of hill climbing, which is a simple and efficient local search algorithm. It works by iteratively making small changes to a solution and keeping the changes if they improve the solution.

In the context of the knapsack problem, the solution is a chromosome, and a small change is the flipping of a gene (i.e., changing the inclusion status of an item). The local search function iterates over each gene in a chromosome, flips the gene, and calculates the new fitness. If the new fitness is not higher than the old fitness, the gene is flipped back. This process is repeated for each gene in the chromosome. This chromosome is the new updated child used to form the new population.

Hill climbing was chosen for its simplicity and efficiency. It's a straightforward method that can effectively refine the solutions generated by the Genetic Algorithm (GA), by exploring the immediate neighbourhood of each solution.

Question 4:

The program was run on a machine with an Intel Core i7-8700k processor and 16GB of RAM. The operating system was Windows 10. The compiler used was GCC version 9.3.0.

The program was tested with different sets of items for the knapsack problem. Each set of items has a different number of items (***numItems***), each with weights and values. Both GA and GA-LS were run (which the user can pick) , and the seed used was the current time, to ensure different results for each run. Users can also pick to run Z-Tests which run multiple tests to compare means of both GA and GA-LS and displays the Z score for the problem.

The time to complete each problem was recorded as well as the best chromosome after each generation. 2 options are also available.

Parameter	**Without Local Search**	**With Local Search**
Population Size	100	100
Selection Size	4	4
Max Generations	5 * numItems	numItems
Crossover Rate	0.85	0.85
Base Mutation Rate	0.03	0.03
Max Mutation Rate	0.6	0.6
Mutation Rate	Min((***baseMutationRate*** * ***numItems***), maxMutationRate)	Min((***baseMutationRate*** * ***numItems***), ***maxMutationRate***)

Question 5:

Problem Instance	Algorithm	Seed Value	Best Solution	Known Optimum	Runtime (seconds)
f1_l-d_kp_10_269	GA-LS	1714307583	2 3 4 8 9 10	295	0.0166368
	GA	1714307651	2 3 4 8 9 10	295	0.0480643
f2_l-d_kp_20_878	GA-LS	1714307721	1 2 3 4 5 6 7 8 9 10 11 12 13 15 17 19 20	1024	0.0742206
	GA	1714307845	1 2 3 4 5 6 7 8 9 10 11 12 13 15 17 19 20	1024	0.137053
f3_l-d_kp_4_20	GA-LS	1714307883	1 2 4	35	0.0031061
	GA	1714307974	1 2 4	35	0.0128604
f4_l-d_kp_4_11	GA-LS	1714308004	2 4	23	0.0030457
	GA	1714308090	2 4	23	0.0137423
f5_l-d_kp_15_375	GA-LS	1714308154	3 5 7 8 10 11 12 14 15	481.069	0.0401142
	GA	1714308221	3 5 7 8 10 11 12 14 15	481.069	0.0903042
f6_l-d_kp_10_60	GA-LS	1714308404	3 4 6 7 8 9 10	52	0.0169
	GA	1714308464	3 5 6 7 8 9 10	52	0.0468622
f7_l-d_kp_7_50	GA-LS	1714308610	1 4	107	0.0079345
	GA	1714308688	1 4	107	0.0269513
f8_l-d_kp_23_10000	GA-LS	1714308731	1 2 3 4 5 6 7 8 10 16 17	9767	0.108316
	GA	1714308774	1 2 3 4 5 6 7 8 11 16 17	9767	0.172377
f9_l-d_kp_5_80	GA-LS	1714308828	1 2 3 4	130	0.0046651
	GA	1714308889	1 2 3 4	130	0.0174292
f10_l-d_kp_20_879	GA-LS	1714308933	1 2 3 4 5 6 7 8 9 11 12 13 14 16 18 19 20	1025	0.0791088
	GA	1714309062	1 2 3 4 5 6 7 8 9 11 12 13 14 16 18 19 20	1025	0.135869
knapPI_1_100_1000_1	GA-LS	1714309111	7 11 14 24 26 31 33 38 39 49 54 61	9147	5.52614
	GA	1714309172	7 11 14 24 26 31 33 38 39 49 54 61	9147	1.31656

This table shows each run using time as a seed value hence the increase in seed value for each problem. The best solution consists of the indices of the items in the given problem separated by a space. The know optimum is the best fitness found throughout the GA/GA-LS process and the runtime is the time taken for the algorithm to complete in seconds.

Question 6:

Problem Instance	Z-score	Fail to Reject Null Hypothesis (Means are equivalent at 5% confidence)
f1_l-d_kp_10_269	1.3838	True
f2_l-d_kp_20_878	0	True
f3_l-d_kp_4_20	0	True
f4_l-d_kp_4_11	0	True
f5_l-d_kp_15_375	-1.0171	True
f6_l-d_kp_10_60	1.46385	True
f7_l-d_kp_7_50	0.947106	True
f8_l-d_kp_23_10000	-2.63087	True
f9_l-d_kp_5_80	0	True
f10_l-d_kp_20_879	-3.30289	True
knapPI_1_100_1000_1	0.238373	True

The Z-score in this is computed comparing 30 tests on GA-LS vs GA with each test using a random seed. The value being compared is the known optimum of each test as this is the metric that I have chosen for performance, as measuring time does not take into consideration whether the solution is correct. Since this is a single tailed test, the high negative values still fail to reject the null hypothesis.

Question 7:

In conclusion we see that GA and GA-LS were able to produce correct known optimums throughout the problems. GA-LS had a runtime between 25% and 60% that of GA for all problems except the last problem in which the runtime was about 400% that of GA. This is likely due to the way in which the local search works by iterating over all genes in a chromosome meaning that problems like the last, with lots of items, run much slower. If the runtime had to stop when the actual solution was found then these runtime values would likely change significantly but since the optimum will not normally be readably known (as this would make the algorithm almost useless), this is not applicable to this specific issue. Looking at the Z-scores of the single tailed hypothesis testing we can see that for all cases we fail to reject the null hypothesis test (that the means are equivalent at 5% confidence) this means that the results for GA are normally comparable with GA-LS however we do have high negative z scores meaning that GA-LS sometimes performs worse at finding the correct solution then GA. In conclusion this shows that using this current configuration, GA-LS most often executes faster then GA, but GA is more often closer to the correct solution. Values could be changed that would increase GA-LS runtime but also increase its ability to find the correct solution, and same for GA but those changes would depend on whether efficiency or correctness is preferred.

References

(n.d.). Retrieved from Research Gate: https://www.researchgate.net/publication/339771109_Solving_Knapsack_Problem_with_Genetic_Algorithm_Approach

AG-knapsack. (n.d.). Retrieved from EHU: http://www.sc.ehu.es/ccwbayes/docencia/kzmm/files/AG-knapsack.pdf