COS 314 Assignment 2
Keelan Matthews – u21549967

Technical Specification

Objective

The aim of this assignment is to implement the Genetic Algorithm and Ant Colony Optimization Algorithm and evaluate their effectiveness on the given datasets. The objective is to optimize these algorithms to obtain as many optimal solutions as possible, while minimizing the computation time, as well as critically analyze and compare the two meta-heuristics.

Hardware Used

The assignment was completed on an Asus Vivobook Pro 17 laptop with the following specifications:

- CPU: Intel Core i7-8550U, with a clock speed of up to 4.0 GHz
- RAM: 16GB
- Operating System: Windows 11
- Development Environment: Visual Studio Code

GA Configuration

The Genetic Algorithm was initially configured with the POPULATION_SIZE to be 50 (Michalewicz, 2013) and the TOURNAMENT_SIZE to be half of the current population. The MUTATION_RATE was 0.01 and the CROSSOVER_RATE was 0.9 (Goldberg, 1989). Lastly, the MAX_GENERATIONS was set to 1000.

The configuration values were adjusted for better performance. Specifically, the POPULATION_SIZE was set to be the same as the number of items in the knapsack. This made the population size dynamic based on the specific problem instance, which is important for achieving optimal results. Additionally, the TOURNAMENT_SIZE was set to be a quarter of the current population to include a wider variety of chromosomes in the tournament. This was because setting it to half the population resulted in the tournament being too full of top fitness solutions. The MUTATION_RATE was also increased to 0.05 to help escape local optimum. Finally, the MAX_GENERATIONS was lowered to 500 to reduce computation time, as the optimal solution was often found before reaching 1000 generations.

Experimental setup (testing)

Initially there was a high tournament selection size. This resulted in the algorithm hitting a local optimum and producing subpar results. This was emphasized with a low mutation rate. This was resolved when the mutation rate was increased and the tournament size was decreased, introducing a wider range of solutions into the population. The population size was also set to the number of items at a stage but was later halved to promote a higher generation count.

Parameter	Initial Value	Final value
POPULATION_SIZE	50	No. of items / 2
TOURNAMENT_SIZE	Population / 2	Population / 4
MUTATION_RATE	0.01	0.05
CROSSOVER_RATE	0.9	0.9
MAX_GENERATIONS	1000	500

ACO Configuration

The Ant Colony Optimization Algorithm was initially configured with the number of ants to be 50 (M. Dorigo, 1996). The pheromone deposit amount (Q) was equal to 1.0, the BETA value equal to 5.0, the RHO set to 0.5 and the ALPHA value equal to 1.0 (M. Dorigo, 1996). Lastly, the MAX_ITERATIONS was set to 1000.

The configuration values were adjusted for better performance. Specifically, MAX_ITERATIONS was lowered to 500 to reduce computation time, as the optimal solution was often found before reaching 1000 iterations. The heuristic information factor (BETA) was lowered to 3.0 to place more emphasis on the pheromone trails. Lastly, the number of ants was set equal to the number of items in the knapsack.

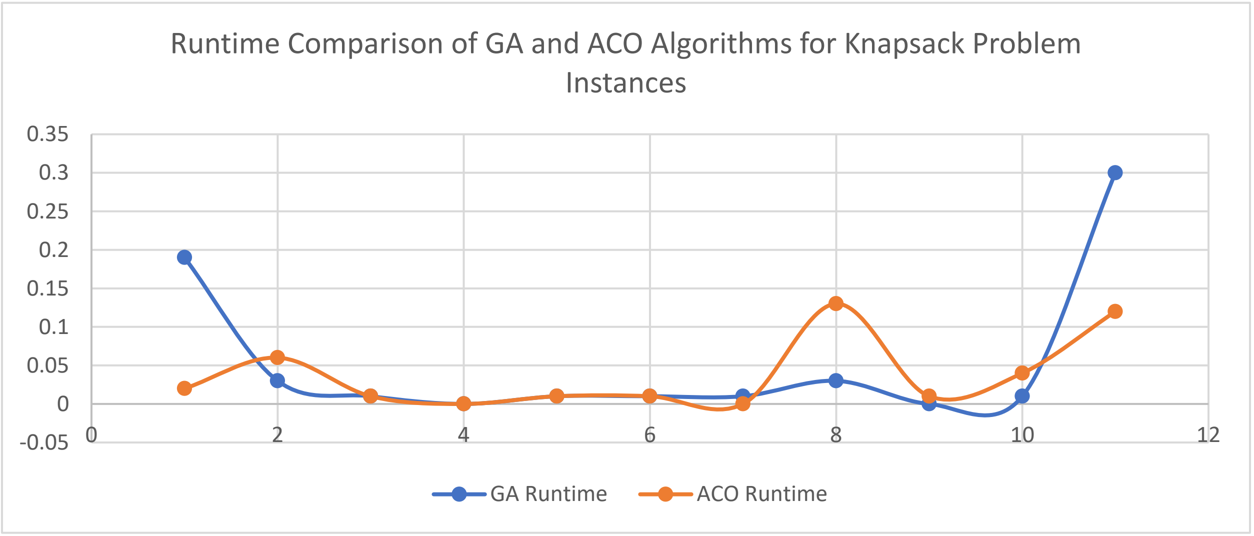
Experimental Setup (testing)

Initially, the pheromone matrix was only updated where the current solution was better than the previous solution. This caused the algorithm to get stuck at local optimums as it was not encouraged to explore potentially worse solutions. This was solved by updating the pheromone matric even if it was a worse solution, but only updating the pheromone values a tenth of the amount as opposed to if the solution was better. The number of ants was also set to the number of items at a stage, but that proved to be way too many. I managed to decrease the number of ants all the way down to 2 whilst still achieving optimal results.

Parameter	Initial Value	Final Value
numAnts	50	2
Q	1.0	1.0
BETA	5.0	3.0
ALPHA	1.0	1.0
RHO	0.5	0.5
MAX_ITERATIONS	1000	500

Evaluation Metrics

Problem Instance	Algorithm	Best Solution	Known Optimum	Runtime (seconds)	Iterations to optimal
f1_l-d_kp_10_269	ACO	295	295	0.02	3
	GA	295		0.19	38
f2_l-d_kp_20_878	ACO	1024	1024	0.06	4
	GA	1024		0.03	120
f3_l-d_kp_4_20	ACO	35	35	0.01	1
	GA	35		0.01	2
f4_l-d_kp_4_11	ACO	23	23	0.0	14
	GA	23		0.0	199
f5_l-d_kp_15_375	ACO	481.0694	481.0694	0.01	0
	GA	481.0694		0.01	106
f6_l-d_kp_10_60	ACO	52	52	0.01	1
	GA	52		0.01	79
f7_l-d_kp_7_50	ACO	107	107	0.0	8
	GA	100		0.01	454
f8_l-d_kp_23_10000	ACO	9757	9767	0.13	369
	GA	9759		0.03	269
f9_l-d_kp_5_80	ACO	130	130	0.01	0
	GA	130		0.0	118
f10_l-d_kp_20_879	ACO	1025	1025	0.04	28
	GA	1025		0.01	140
knapPI_1_100_1000_1	ACO	9147	9147	0.12	9
	GA	0		0.3	0
			Total time:	**ACO**: **0.41**	
				GA: 0.6	



Results

The results of the experiments conducted indicate that both ACO and GA algorithms are effective in solving the knapsack problem with small datasets. This is because both algorithms were able to generate optimal solutions within an average of 0.03 seconds, with only minor differences between their total runtimes. However, due to the stochastic nature of these algorithms, both ACO and GA produced sub-optimal solutions for some instances.

When it comes to larger datasets, ACO outperformed GA significantly. In particular, ACO was able to generate an optimal solution for knapPI_1_100_1000_1 in just over 0.1 seconds, whereas GA was unable to produce a single valid solution with the same parameters. This result highlights the advantages of ACO over GA in dealing with complex, high-dimensional search spaces.

One reason for ACO's superior performance is its ability to use pheromone trails to guide the search process. These trails help to identify promising regions in the search space, which in turn facilitates a more efficient exploration of the solution space. In contrast, GA's dependence on genetic operators can lead to premature convergence and difficulty in exploring large search spaces.

Conclusion

In conclusion, the comparative analysis of ACO and GA for solving the knapsack problem reveals that both algorithms are efficient for smaller search spaces, producing optimal solutions within milliseconds. However, ACO outperforms GA for larger search spaces due to its ability to guide the search using pheromone trails, whereas GA's reliance on genetic operators may cause difficulty in exploring larger search spaces. Therefore, it is recommended to use ACO for solving complex combinatorial optimization problems that involve large search spaces. On the other hand, GA may be more suitable for simpler problems that involve smaller search spaces. Overall, the choice of algorithm depends on the problem requirements and the size of the search space.

References

[1]. Goldberg, D. E., 1989. *Genetic algorithms in search, optimization, and machine learning..* 75 Arlington Street, Suite 300 Boston, MA United States: Addison-Wesley Longman Publishing Co., Inc..

[2]. M. Dorigo, V. M. a. A. C., 1996. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 26, no. 1,* pp. 29-41.

[3]. Michalewicz, Z. a. D. B. F., 2013. *How to solve it: modern heuristics..* Berlin: Springer, Berlin, Heidelberg.