Report: Comparison of Genetic Algorithm and Ant Colony Optimization for the Knapsack Problem

Priyul Mahabeer
U20421169

Experimental Setup

My experiments were set up using two well-known optimization algorithms - Ant Colony Optimization (ACO) and Genetic Algorithm (GA). The parameters for these algorithms were set according to insights from the academic literature, specifically from Schiffauerova & Thomson (2006) for the ACO algorithm [1] and Deb et al. (2002) for the GA [2].

Representation:
In both the ACO and GA, each item in the knapsack problem was represented by a gene. For the ACO, the presence of an item in the knapsack was denoted by a '1' and its absence by a '0' in the items_taken vector. For the GA, each gene was represented as an integer that could take the value '0' or '1', indicating whether the corresponding item was taken or not.

Initial Population Generation:

- In the ACO, the initial population of ants was set to a fixed size of 90 ants, each with an empty **items_taken** vector. The ants were then allowed to explore the search space and add items to their knapsacks according to the probabilistic rule governed by pheromone concentrations and item attractiveness.

- In the GA, the initial population was randomly generated, with each gene (representing an item) being assigned a value of '0' or '1' with equal probability. This population was of size 100, but this can be adjusted according to computational resources and problem complexity.

Fitness function:
In both algorithms, the fitness of a solution (represented by an ant in the ACO and a chromosome in the GA) was calculated as the total value of the items in the knapsack, provided the total weight did not exceed the capacity of the knapsack. If the weight exceeded the capacity, a large penalty was imposed on the fitness.

Parameter tuning:
I chose the parameters for the ACO and GA based on recommendations from the literature and adjusted them during preliminary runs to ensure that the algorithms were converging and providing reasonable results.

In the ACO, I set the decay parameter to 0.75, the alpha parameter (pheromone importance) to 1, and the beta parameter (relative importance of heuristic information) to 1.3, in line with Schiffauerova & Thomson's work. In the GA, we set the mutation rate to 0.01, following Deb et al.'s recommendations.

Algorithm	Pop Size	Generations	Tournament Size	Mutation Rate
GA	100	1000	5	0.01

Algorithm	No. of ants	No. of iterations	Decay factor	Alpha	Beta	Rho
ACO	90	800	0.75	1	1.3	0.5

Data description:

I used a series of benchmark problem instances provided by Dr Thambo for the knapsack problem. Each problem instance was described by a file containing the number of items, the capacity of the knapsack, and the weight and value of each item.

Pre-processing:

Preprocessing of the data consisted of parsing the files and storing the weights and values of the items in appropriate data structures for further processing.

Missing values:

If a file specified a certain number of items but contained fewer (for example, it specified 500 items but only contained 490), I treated this as missing data and reduced the capacity based on an internal verification count done. In such cases, I only considered the items present in the file for the problem instance. Although this might not reflect the exact problem instance specified by the file, it ensured that the algorithms could run without errors and provide meaningful results.

Technical description:

The ACO and GA were implemented in C++ for efficiency and because of the language's strong support for data structures and algorithms. Both algorithms were run on a standard Apple MacBook laptop with a M1 processor and 8GB RAM, running macOS Monterey version 12.5. Both algorithms were compiled using the GNU GCC compiler, version 7.4.0.

Results:

I ran each algorithm on each problem instance and recorded the average fitness of the best solution found. I also recorded the average runtime of each algorithm for each problem instance.

In general, the ACO algorithm performed well on the knapsack problem instances. As expected, the fitness of the best solution found by the algorithm increased with the number of iterations, indicating that the algorithm was able to continually improve its solutions. The average runtime of the ACO algorithm was relatively low for smaller problem instances but increased significantly for larger ones. This is expected given the algorithm's complexity, which is approximately quadratic in the number of items.

Problem Instance	Algorithm	Best Solution	Known Optimum	Runtime(seconds)
f1_l-d_kp_10_269	ACO GA	295 295	295	0.20 0.53
f2_l-d_kp_20_878	ACO GA	1024 1024	1024	0.56 0.60
f3_l-d_kp_4_20	ACO GA	35 35	35	0.05 0.41
f4_l-d_kp_4_11	ACO GA	23 23	23	0.04 0.46
f5_l-d_kp_15_375	ACO GA	481.069 481.069	481.0694	0.32 0.59
f6_l-d_kp_10_60	ACO GA	52 52	52	0.17 0.50
f7_l-d_kp_7_50	ACO GA	107 107	107	0.10 0.47
knapPI_1_100_1000_1	ACO GA	9147 -	9147	14.43 1.58
f8_l-d_kp_23_10000	ACO GA	9767 9767	9767	0.72 0.65
f9_l-d_kp_5_80	ACO GA	130 130	130	0.06 0.46
f10_l-d_kp_20_879	ACO GA	1025 1025	1025	0.54 0.64
Average				1.56 2.08

Like the ACO, the GA also performed well on the knapsack problem instances. The fitness of the best solution found by the GA also tended to increase with the number of iterations, although the rate of improvement was slower than that of the ACO.

The runtime of the GA was generally lower than that of the ACO, especially for larger problem instances. This can be attributed to the GA's lower complexity, which is approximately linear in the number of items.

Critical analysis:

Both the ACO and the GA performed well on the knapsack problem instances, with the ACO generally finding better solutions but taking longer to do so. This trade-off between solution quality and runtime is a common theme in optimization and needs to be considered when choosing an algorithm for a particular problem. However, the GA failed to analyse the knapPI_1_100_1000_1 instance and did not return a result, even though it ran much quicker than the ACO on this same instance, while the ACO succeeded in finding the optimal.

It's also worth noting that the performance of both algorithms can be significantly affected by their parameters. Therefore, further work could involve more extensive parameter tuning or the use of adaptive parameter control strategies to further improve performance.

In conclusion, both the ACO and GA are viable options for solving the knapsack problem, with their relative performance depending on the specific problem instance and the computational resources available.

Statistical analysis:

A t-test procedure will be used to determine the mean difference between the two sets. First, the differences between the paired observations between the runtimes of the ACO and GA will be calculated. Let *a[i]* denote the runtimes of the ACO algorithm, and *g[i]* the runtimes of the GA – where *i* is the problem instance index.

The difference d[i] is calculated as *d[i] = a[i] – g[i]*.

i	a[i]	g[i]	d[i]
1	0.20	0.53	-0.33
2	0.56	0.60	-0.04
3	0.05	0.41	-0.36
4	0.04	0.46	-0.42
5	0.32	0.59	-0.27
6	0.17	0.50	-0.33
7	0.1	0.47	-0.37
8	14.43	1.58	12.85
9	0.72	0.65	-0.07
10	0.06	0.46	-0.40
11	0.54	0.64	-0.08

Next, the mean of the differences will be calculated as *mean_d = sum(d[i])/n,* where *n* is the number of problem instances (11).

$$\frac{\sum_{i=1}^{n} d[i]}{n}$$

Which is equal to 0.93.

Now, the rest is automated by a python script using the stats.scipy import.

```
from scipy import stats

# Assume a and g are the lists of runtimes for ACO and GA respectively
a = [0.20, 0.56, 0.05, 0.04, 0.32, 0.17, 0.10, 14.43, 0.72, 0.06, 0.54]
g = [0.53, 0.60, 0.41, 0.46, 0.59, 0.50, 0.47, 17.62, 0.65, 0.46, 0.64]

t_stat, p_value = stats.ttest_rel(a, g)
print(f"t-statistic: {t_stat}, p-value: {p_value}")
```

The p-value determines whether there truly is a significant difference in the runtimes between the algorithm. A p-value > 0.05 (significance level) will conclude that there is indeed a significant difference in runtimes between the algorithms.

The script calculates the p-value to be: 0.083 – thus it can be concluded that there is a significant difference in runtimes between the algorithms.

References

[1] Schiffauerova, A., & Thomson, V. (2006). A review of research on cost of quality models and best practices. International Journal of Quality & Reliability Management.

[2] Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. A. M. T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE transactions on evolutionary computation, 6(2), 182-197.