COS 314 Practical 2
U19027372 Simphiwe Ndlovu
Assignment 2 - Search

The experiments were conducted on a machine with an Intel® Core™ i5-1035G1 CPU @ 1.00GHz × 8 and 8 GB of RAM . I implemented the algorithms in Java .

1. GA Configuration Description

This part of the report focuses on the implementation of a Genetic Algorithm (GA) to solve the Knapsack Problem. The GA aims to maximize the total value of selected items while respecting a given capacity constraint.

- **Population Size:**
  The population size is set to TOURNAMENT_SIZE, which is initialized as 4 in the code. This means that the initial population consists of 4 individuals**.**
- **Mutation Rate:**
  The mutation rate is defined as MUTATION_RATE, which is set to 0.1 in the code. This value represents the probability of a gene being mutated during the evolution process. In this case, each gene in an individual has a 10% chance of being mutated.
- **Maximum Iterations:**
  The maximum number of iterations is specified by MAX_ITERATIONS, set to 1000 in the code. The algorithm will terminate if this number of iterations is reached before convergence criteria are met.
- **Convergence Threshold and Iterations:**
  The algorithm employs a convergence criterion to detect when the best solution remains unchanged over multiple iterations, indicating convergence to a local optimum. The convergence threshold is defined as CONVERGENCE_THRESHOLD, set to 0.001 in the code. If the difference between the fitness of the current best individual and the fitness of the previous best individual falls below this threshold, it is considered a sign of convergence. The number of iterations required for convergence is set as CONVERGENCE_ITERATIONS, which is 20 in the code. If the convergence condition is met for CONVERGENCE_ITERATIONS consecutive iterations, the algorithm terminates.

- **Fitness Evaluation:**
  The fitness of each individual in the population is calculated based on the total value of selected items while considering the weight constraints. The fitness function penalizes solutions that exceed the capacity (CAPACITY) by setting their total value to 0.

- **Selection: Tournament Selection:**
  The tournament selection method is used to select parents for crossover. In each tournament selection, TOURNAMENT_SIZE individuals are randomly chosen from the population, and the one with the highest fitness is selected as a parent. This process is repeated to select the second parent.

- **Crossover: Single-Point Crossover:**
  The crossover operation is performed using a single-point crossover technique. A random crossover point is selected, and the genes before the crossover point are inherited from one parent, while the genes after the crossover point are inherited from the other parent. This process creates a new individual (child) with a combination of genes from both parents.

- **Mutation:**
  The mutation operator is applied to each gene in an individual with a probability of MUTATION_RATE. If the random value generated for a gene is less than the mutation rate, the gene's value is flipped (from 0 to 1 or vice versa), introducing small variations in the population.

- **Termination Criteria:**
  The algorithm terminates when either the maximum number of iterations (MAX_ITERATIONS) is reached or the convergence criterion is met (no improvement in the best solution for CONVERGENCE_ITERATIONS consecutive iterations).

- **Output:**
  The best solution found, including the genes and the fitness value, is printed at the end of the algorithm execution.

Justification based on common practices and considerations in genetic algorithm (GA) parameter selection.

Mutation Rate:

- The mutation rate determines the probability of a gene being mutated in each individual during the evolution process.
- A mutation allows exploration of new regions in the search space and introduces diversity within the population, which helps prevent premature convergence and increases the chances of finding optimal solutions.
- A mutation rate of 10% (0.1) is a commonly used value that strikes a balance between exploration and exploitation.

- Higher mutation rates can result in excessive exploration, which may slow down convergence or degrade the quality of solutions.
- Lower mutation rates can limit exploration, leading to premature convergence and getting trapped in local optima.
- Therefore, a mutation rate of 10% is often considered a reasonable starting point, allowing for effective exploration and maintaining diversity within the population.

Convergence Threshold (0.001):

- Sensitivity to Improvement: A threshold of 0.001 indicates that the GA will continue iterating until the improvement in fitness falls below this value. This suggests that the algorithm aims to achieve a relatively high level of precision in the solutions.
- Problem Complexity: The choice of convergence threshold can depend on the complexity of the problem being solved. More complex problems may require a smaller threshold to ensure that the GA converges to optimal or near-optimal solutions.
- Trade-off between Precision and Performance: A smaller threshold allows for more precise solutions but may require additional iterations and computational resources. By selecting 0.001, the code strikes a balance between solution accuracy and performance.

Convergence Iterations (10):

- Early Stopping: By checking for convergence after 10 iterations, the code aims to stop the algorithm if no significant improvement in the fitness value is observed. This helps prevent the algorithm from running indefinitely when there is little progress.
- Performance Optimization: Setting a smaller number of convergence iterations can improve the overall performance of the GA by reducing unnecessary iterations when convergence has already been achieved.

Overall, the selection of a convergence threshold of 0.001 and 10 convergence iterations aims to strike a balance between solution precision and performance while providing a mechanism to detect when the GA has converged to a satisfactory solution.

TOURNAMENT_SIZE : 4

- The selection of a tournament size of 4 aims to strike a balance between exploration and exploitation while considering computational efficiency. It allows for sufficient diversity in the selected individuals, enabling exploration of the solution space, while also favoring the fitter individuals. Furthermore, the chosen tournament size provides computational efficiency by reducing the computational complexity compared to other selection methods that consider the entire population.

1. GA Configuration Description

Ant Colony Optimization (ACO) is a metaheuristic algorithm inspired by the foraging behavior of ants. It has been successfully applied to various combinatorial optimization problems, including the Knapsack Problem. In this report, we will describe the configuration of the ACO algorithm for solving the Knapsack Problem and provide justifications for the chosen parameter values.

Number of Ants:

- The number of ants represents the population size in the ACO algorithm. Each ant constructs a solution and updates the pheromone trails based on its experience. The number of ants should be large enough to explore the solution space effectively. In our implementation, the number of ants is set to the same value as the number of items in the Knapsack Problem instance. This ensures that each item is considered by at least one ant during the construction process.

Pheromone Evaporation Rate:

- The evaporation rate determines the decay of pheromone trails over time. It controls the balance between exploitation (using the current information) and exploration (encouraging new paths). A higher evaporation rate reduces the influence of outdated pheromone trails, promoting exploration, while a lower rate emphasizes exploitation. In our configuration, the evaporation rate is set to 0.5, which strikes a balance between exploration and exploitation.

Alpha and Beta Values:

- Alpha and beta are parameters that control the importance of pheromone trails and heuristic information, respectively, in the ant's decision-making process. Alpha assigns weight to the pheromone trail, while beta assigns weight to the heuristic information (e.g., the value or weight of the items). The selection of alpha and beta values depends on the problem characteristics. In our implementation, both alpha and beta are set to 1.0, giving equal importance to both pheromone trails and heuristic information.

Initial Pheromone Level:

- The initial pheromone level represents the amount of pheromone deposited on each trail at the beginning of the algorithm. It provides a baseline for the ants to start their exploration. The initial pheromone level should be carefully chosen to prevent premature convergence and ensure sufficient exploration. In our configuration, the initial pheromone level is set to 1.0 divided by the product of the number of items and the initial pheromone parameter. This ensures a balanced initial distribution of pheromone.

Maximum Iterations:

- The maximum iterations define the termination condition for the algorithm. Once the specified number of iterations is reached, the algorithm stops, regardless of convergence. It is essential to set a reasonable value to avoid excessive

computation while allowing sufficient time for convergence. In our configuration, the maximum iterations are set to 1000, providing a substantial number of iterations for the algorithm to converge.

Convergence Threshold:

- The convergence threshold determines the number of iterations without improvement required to consider the algorithm converged. If the best solution does not improve for the specified number of iterations, the algorithm terminates. The convergence threshold helps avoid unnecessary computation when further improvement is unlikely. In our configuration, the convergence threshold is set to 20 iterations.

Justifications for the chosen parameter values:

- The number of ants is set equal to the number of items to ensure sufficient exploration of the solution space.
- A moderate evaporation rate of 0.5 balances exploration and exploitation.
- Alpha and beta are both set to 1.0 to give equal importance to pheromone trails and heuristic information.
- The initial pheromone level is calculated to provide a balanced distribution of pheromone at the start.
- The maximum iterations are set to 1000 to allow for a substantial number of iterations.
- The convergence threshold of 20 iterations helps terminate the algorithm when further improvement is unlikely.

3. Experimental setup.(including table of parameters)

This section provides an overview of the specific parameter values used, the dataset used for testing, and any preprocessing steps performed before running the algorithms.

GA

Parameter Values:

The following table presents the specific parameter values used for the GA algorithm:

Parameter	Value
Tournament Size	4
Mutation Rate	0.1
Maximum Iterations	1000

Convergence Threshold	0.001
Convergence Iterations	100

Dataset and Problem Instance:

The specific problem instance used for testing includes the following details:

- Number of Items: Variable (determined by the problem instance)
    - Represents the total number of items available for selection.
- Knapsack Capacity: Variable (determined by the problem instance)
    - Specifies the maximum weight the knapsack can hold.
- Item Values: Array of double values
    - Represents the values associated with each item.
- Item Weights: Array of double values
    - Represents the weights associated with each item.

Preprocessing and Data Preparation:

Before running the GA algorithm, the following preprocessing steps or data preparation are performed:

- Problem Instance Setup:
    - The number of items, knapsack capacity, item values, and item weights are extracted from the problem instance data and assigned to the corresponding variables in the algorithm.

ACO

Experimental parameter configuration table for the ACO

Parameter	Value
Number of Ants	Equal to the number of items in the Knapsack Problem instance
Pheromone Evaporation Rate	0.5
Alpha (Pheromone Trail Weight)	1.0
Beta (Heuristic Information Weight)	1.0
Initial Pheromone Level	1.0 divided by (Number of items * Initial Pheromone parameter)

Maximum Iterations	1000
Convergence Threshold	20

Dataset/Problem Instance:

- We used a dataset of Knapsack Problem instances for testing the ACO algorithm. Each instance consists of a set of items with associated values and weights, as well as a knapsack capacity. The dataset includes various problem sizes and characteristics to evaluate the algorithm's performance across different scenarios. The specific details of the problem instances, such as the number of items, capacity, were provided in the input to the algorithm.

Preprocessing and Data Preparation:

Before running the ACO algorithm, the problem instances were prepared as follows:

- The items' values and weights were extracted from the dataset and stored in a suitable data structure, such as a 2D array .
- The knapsack capacity was obtained from the dataset and used to set the capacity parameter in the ACO algorithm.
- Had to change the array types to a double to accommodate one of the problem instances.

4. A table (exemplified below) presenting the results.

Table 1: Comparison of ACO and GA on 10 knapsack problem instances

Problem Instance	Algorithm	Best Solution	Known Optimum	Runtime ((nano second))
f1_l-d_kp_10_269	ACO GA	246.0 295.0	295	18 31
f2_l-d_kp_20_878	ACO GA	943.0 876.0	1024	14 25
f3_l-d_kp_4_20	ACO GA	35.0 35.0	35	104 178
f4_l-d_kp_4_11	ACO GA	23.0 16.0	23	147 217
f5_l-d_kp_15_375	ACO	279.0	481,0694	56

	GA	418.124943		67
f6_l-d_kp_10_60	ACO GA	52.0 52.0	52	176 119
f7_l-d_kp_7_50	ACO GA	92.0 105.0	107	270 194
knapPI_1_100_1000_1	ACO GA	5000.0 0.0	9147	20 16
f8_l-d_kp_23_10000	ACO GA	9751.0 9713.0	9767	97 79
f9_l-d_kp_5_80	ACO GA	130.0 130.0	130	486 397
f10_l-d_kp_20_879	ACO GA	972.0 961.0	1025	128 104

5. Statistical analysis of differences in performance need to be presented.

6. A critical analysis of the results.

The results obtained from the evaluation of the Ant Colony Optimization (ACO) and Genetic Algorithm (GA) for solving the Knapsack Problem are presented as follows:

ACO Results:

Total Best Solutions: 17,523
Total Time: 1,516
GA Results:

Total Best Solutions: 12,601
Total Time: 1,427
Now let's analyze these results and draw some critical insights:

Solution Quality:

- ACO: The ACO algorithm found a higher number of best solutions (17,523) compared to GA (12,601). This indicates that ACO was able to explore a larger portion of the solution space and discover more favorable solutions.

- GA: Although GA found a smaller number of best solutions, it's important to consider the quality of those solutions. If the best solutions found by GA are of higher fitness or closer to the optimal solution, it can still be considered effective.

Performance:

- Time Efficiency: Both ACO and GA have comparable total times, with ACO taking 1,516 units of time and GA taking 1,427 units of time. Since the time difference is relatively small, it suggests that the two algorithms have similar efficiency in terms of computational resources used.
- Scalability: The total time does not provide information about the algorithm's scalability with problem size. It would be beneficial to evaluate how the algorithms perform with larger problem instances to understand their scalability characteristics.

Algorithm Selection:

Solution Diversity: ACO demonstrated a higher number of best solutions, indicating that it explored a more diverse set of solutions. This can be advantageous in scenarios where finding a wide range of good solutions is desired.
Optimization Precision: The quality of the best solutions found by GA is worth considering. If GA consistently finds solutions that are close to the optimal solution, it may be a better choice when precision is crucial.
Future Directions:

Fine-tuning: Both algorithms can potentially be fine-tuned by adjusting their parameter values to improve performance. Experimenting with different parameter settings might lead to better results for either ACO or GA.
Problem-Specific Analysis: Analyzing the characteristics of the Knapsack Problem and the specific problem instances used could provide insights into which algorithm is better suited for this particular problem domain.

Conclusion

The ACO algorithm demonstrated a higher number of best solutions, indicating better solution diversity. However, the GA algorithm should not be disregarded, as its best solutions may still be of high quality. Further analysis and experimentation are necessary to determine the strengths and weaknesses of each algorithm and make an informed decision regarding algorithm selection for the Knapsack Problem.

Reference of the configurations

The Knapsack Problem & Genetic Algorithms - Computerphile

Genetic Algorithms in Search, Optimization, and Machine Learning" by David E. Goldberg

Title: "Ant Colony Optimization"
Author: Marco Dorigo and Thomas Stützle
Publisher: The MIT Press
Year: 2004
ISBN: 978-0262042197