Evolutionary Algorithm for Knapsack Problem

The code implements an evolutionary algorithm to solve the 0-1 knapsack problem. In this problem, each item is assigned a value (1, 2, 3, …) and a randomly generated weight (between 1 and 10). The goal is to pick a subset of items that maximize total value without exceeding a knapsack capacity (half the total weight of all items). The candidate solution is represented as a bitstring, where each bit indicates whether an item is included.

**Design Choices and Implementation:**

* **Genotype Representation:**  
  The candidate solutions are represented as bitstrings of length equal to the number of items. This encoding shows which items to include/exclude for the knapsack.
* **Fitness Function:**  
  The fitness function is the sum of values for selected items. If the total weight is within the capacity, the fitness simply equals the total value. If the weight is over the capacity limit, the fitness is penalized by scaling the total value by . The multiplication by 10 was needed to reduce the frequency of infeasible solutions as now the scaling has a bigger impact on the fitness.
* **Crossover and Mutation:**  
  A single-point crossover operator recombines parental bitstrings at a randomly chosen point, while mutation randomly flips bits with a fixed probability. These were chosen for simplicity and effectiveness.
* **Selection Strategy:**  
  Tournament selection is employed by randomly sampling 40 individuals from the population and choosing the one with the highest fitness. This provides robust selection and preserves diversity.

**Testing:**

After implementation, I conducted some testing:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test | Population | # Items | Generations | Mutation prob | Crossover prob |
| A | 100 | 20 | 100 | 0.1 | 0.8 |
| B | 100 | 100 | 100 | 0.1 | 0.8 |
| C | 500 | 40 | 200 | 0.1 | 0.8 |
| D | 1000 | 40 | 200 | 0.1 | 0.8 |
| E | 500 | 40 | 200 | 0.2 | 0.4 |
| F | 500 | 40 | 300 | 0.4 | 0.4 |

A graph with numbers and lines

Description automatically generatedA graph with numbers and a bar

Description automatically generatedA graph with numbers and a number of people

Description automatically generated with medium confidenceA graph with numbers and lines

Description automatically generatedA graph with a line going up

Description automatically generatedA graph with blue lines

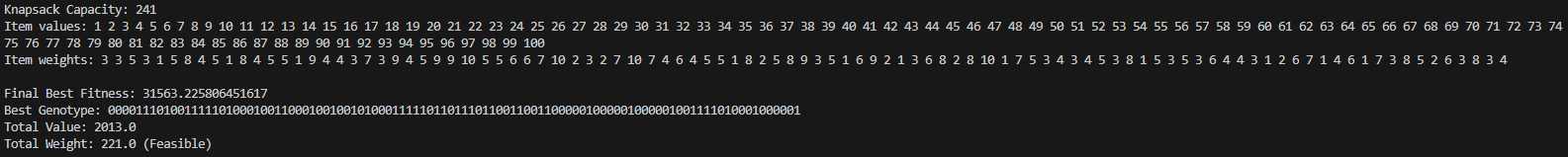
Description automatically generated

A screenshot of a computer

Description automatically generated**Results and Analysis:**

* **Test A:**

Convergence was very fast, with the best fitness stabilizing after just a few generations. The final solution was feasible with a total value of 91 and a weight of 35.

* **Test B:**

With more items, we see how the algorithm is making step improvements, eventually finding the optimal solution with high fitness.

* **Test C:**

A black background with numbers

Description automatically generated

A black screen with white numbers

Description automatically generated

Increasing the population size and generations produced better solutions. Test D (large population) achieved a final best fitness of approximately 6026, with a total value of 370 and weight of 85.

* **Test E:**

A black background with numbers

Description automatically generated

With a higher mutation rate, the final best fitness was slightly lower. This suggests that while higher mutation can improve exploration, it may as well disrupt convergence.