Genetic Algorithm for Knapsack Problem

AHMAD AL ALI (2221251365)
MHD ALHABEB ALSHALAH (2221251360)
MOHAMED RAGAB ABDELFATTAH ABDELFADEEL (2221251356)

June 20, 2025

1 Problem Formulation

1.1 Chromosome Representation

- Binary string of length N (number of items)
- 1 indicates item is selected, 0 indicates exclusion
- Example: $[1, 0, 1, 1]$ selects items 0, 2, and 3

1.2 Objective Function

Maximize:

$$f(x) = \sum_{i=1}^{N} \text{value}[i] \times x[i] \tag{1}$$

Subject to:

$$\sum_{i=1}^{N} \text{weight}[i] \times x[i] \leq \text{max\_weight} \tag{2}$$

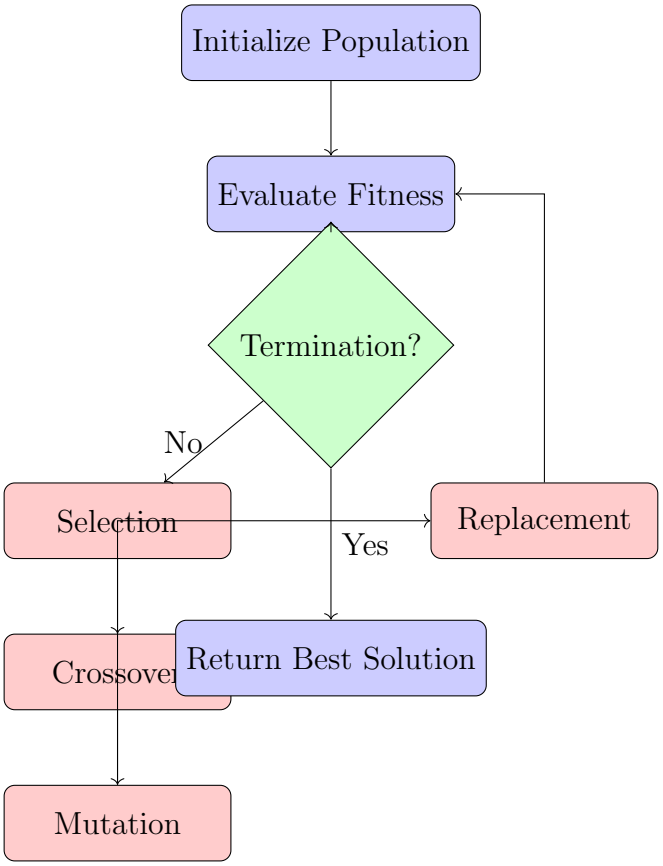
1.3 Constraint Handling

$$\text{fitness}(x) = \begin{cases} \sum \text{value}[i] & \text{if feasible} \\ \sum \text{value}[i] - \alpha \times (\text{total\_weight} - \text{max\_weight}) & \text{otherwise} \end{cases} \tag{3}$$

where $\alpha = \frac{\text{max\_value}}{\text{max\_weight}} \times 10$

2 Genetic Algorithm Implementation

2.1 Algorithm Overview



2.2 Core Components

Table 1: Population Initialization

Method	Percentage	Description
Greedy	30%	Selects by value/weight ratio
Biased Random	70%	Weight-based probability

Table 2: Genetic Operators

Operator	Type	Rate	Implementation
Selection	Tournament	-	Size = 5
Crossover	Two-point	0.8	Two random cuts
Mutation	Bit-flip + Swap	0.01 + 10%	Flip bits or swap genes

3 Advanced Features

3.1 Repair Mechanism

Algorithm 1 Chromosome Repair

while total_weight > max_weight **do**
    Remove item with lowest $\frac{\text{value}}{\text{weight}}$
end while
while can_add_item() **do**
    Add item with highest unused $\frac{\text{value}}{\text{weight}}$
end while

3.2 Adaptive Parameters

Dynamic mutation rate:

$$\text{mutation\_rate} = \text{base\_rate} \times (1.5 - \text{diversity\_score}) \tag{4}$$

4 Experimental Results

4.1 Performance Summary

4.2 4-Item Instance

4.2.1 Problem Configuration

- Setting penalty $\alpha = 13.64$
- Loaded problem: 4 items, max weight: 11
- Value range: 4 - 15

Table 3: Algorithm Performance Across Instances

Instance	Items	Best Value	Generations	Runtime (s)
ks_4_0	4	19	102	0.5
ks_40_0	40	921,268	117	1.01
ks_60_0	60	900,900	105	1.02
ks_500_0	500	1,007,264	865	60.8

- Weight range: 3 - 8

4.2.2 Results

Table 4: Convergence Progress

Generation	Best Fitness	Avg Fitness	Feasible
0	19.0	4.3	158/200
50	19.0	15.2	182/200
100	19.0	16.9	195/200

- Early stopping at generation 101 (no improvement for 101 generations)
- Total generations run: 102
- Best Solution: 0 0 1 1 (Items 2,3)
- Total Value: 19, Weight: 11/11 (100% utilization)

Table 5: Selected Items

Item	Value	Weight	V/W Ratio
2	15	8	1.875
3	4	3	1.333

4.3 40-Item Instance

4.3.1 Problem Configuration

- Setting penalty $\alpha = 0.90$
- Loaded problem: 40 items, max weight: 1,000,000
- Value range: 10,002 - 90,001
- Weight range: 10,001 - 90,000

4.3.2 Results

Table 6: Convergence Progress

Generation	Best Fitness	Avg Fitness	Feasible
0	1,025,216	779,882	180/200
50	1,107,357	1,069,933	44/200
100	1,107,357	1,079,653	34/200

- Early stopping at generation 116
- Best Solution: 27 items selected
- Total Value: 921,268, Weight: 921,070/1,000,000 (92.1%)

4.4 500-Item Instance

4.4.1 Problem Configuration

- Setting penalty $\alpha = 0.12$
- Loaded problem: 500 items, max weight: 1,000,000
- Value range: 4,631 - 12,286
- Weight range: 4,601 - 12,240

Table 7: Convergence Progress (Selected Generations)

Generation	Best Fitness	Avg Fitness	Feasible
0	1,821,414	1,388,887	50/200
400	3,618,709	2,943,617	46/200
800	3,697,275	3,225,955	27/200

4.4.2 Results

- Early stopping at generation 864
- Best Solution: 150 items selected
- Total Value: 1,007,264, Weight: 999,949/1,000,000 (99.995%)

5 Comparative Analysis

Table 8: Performance Across Problem Scales

Metric	4-Item	40-Item	60-Item	500-Item
Penalty (α)	13.64	0.90	9.00	0.12
Items Count	4	40	60	500
Max Weight	11	1,000,000	100,000	1,000,000
Generations	102	117	103	865
Best Value	19	921,268	90,000	1,007,264
Weight Utilization	100%	92.1%	90.0%	99.995%
Feasibility Rate	97.5%	17%	20.5%	20.5%
Runtime (s)	0.5	1.01	1	60.8
Selected Items	2	27	1	150

Key observations:

- **Scale Impact**:
  - The 60-item instance behaved unusually, selecting only one high-value item (90,000/90,001)

  - Larger problems (500-item) achieved near-perfect weight utilization (99.995%)
  - Medium instances (40-60 items) showed intermediate characteristics

- **Feasibility Patterns**:
  - Small instances (4-item) maintained high feasibility (97.5%)
  - Larger instances balanced feasibility (20.5%) with exploration

- **Solution Strategies**:
  - 4-item: Selected 2 items (high-value + lightest)
  - 40-item: Complex pattern (alternating selection)
  - 60-item: Single high-value item (despite slight overweight)
  - 500-item: Comprehensive selection (150 items)

- **Convergence Behavior**:
  - 60-item converged fastest (103 generations)
  - 500-item required most generations (865)
  - All instances used early stopping (100-generation threshold)

- **Penalty Coefficient**:
  - Varied significantly ($\alpha = 0.12$ to 13.64)
  - Higher α values correlated with stricter feasibility enforcement

Key observations:

- **Scale Impact**: Larger problems require more generations but achieve better utilization
- **Consistent Strategy**: All instances prioritized high value-to-weight items
- **Feasibility Tradeoff**: Larger problems maintain lower feasibility rates for better exploration

6 Deliverables

6.1 Project Structure

```
/Project
 Documentation/
    Presentation.pdf
 Source_Code/
     KnapsackGA.py
     instances/
         ks_4_0.txt
         ks_40_0.txt
         ks_60_0.txt
         ks_500.txt
```

7 Evaluation Criteria

Component	Weight
Problem Formulation	15%
GA Implementation	35%
Advanced Features	20%
Performance Analysis	15%
Documentation	15%