A Genetic Algorithm Approach to a Dynamic Optimization Problem

Technical Report (Academic Portfolio)

Milad PourAhmad

December 2025

Abstract

This technical report documents a practical application of a real-valued Genetic Algorithm (GA) to solve a constrained numerical optimization problem that arises in dynamic optimization coursework. The task is to split a positive real constant N into twelve positive, non-uniform components whose sum matches N. Instead of searching for a closed-form construction, we reformulate the task as an optimization problem and solve it using evolutionary operators (roulette-wheel selection, single-point crossover, value mutation, and elitism). The implementation is provided as reproducible C++ software and archived as a citable artifact on Zenodo.

1 Introduction

Many seemingly simple numerical tasks become non-trivial once additional constraints are imposed (positivity, non-uniformity, bounded values, or a need for controlled randomness). In this project, we consider the problem of decomposing a target value into multiple positive parts, motivated by scenarios where *controlled random partitions* are desirable.

Genetic Algorithms (GAs) are population-based metaheuristics inspired by biological evolution. They are particularly useful when (i) the search space is large, (ii) gradients are unavailable, or (iii) we want robust solutions under constraints and noise. This report describes the formulation, algorithm design, and implementation details of a GA-based solver for a 12-variable constrained partition problem.

2 Problem Formulation

Given a constant $N > 0$, find a vector $\mathbf{x} = (x_1, \ldots, x_{12})$ such that

$$\sum_{i=1}^{12} x_i = N, \tag{1}$$

subject to

$$x_i > 0 \quad \text{for all } i. \tag{2}$$

In practice, we also prefer the components to be *non-uniform* (not all equal). The GA naturally promotes diversity through random initialization, crossover, and mutation; therefore, explicit inequality constraints between all pairs are not required for obtaining non-uniform solutions in typical runs.

2.1 Optimization Objective

We define the objective as the absolute deviation from the equality constraint:

$$\mathrm{ObF}(\mathbf{x}) = \left| N - \sum_{i=1}^{12} x_i \right|. \tag{3}$$

The optimal value is $\mathrm{ObF}(\mathbf{x}) = 0$.

3 Genetic Algorithm Design

3.1 Representation

Each candidate solution is a chromosome of 12 real-valued genes:

$$\mathbf{x} = (x_1, \ldots, x_{12}).$$

To restrict the search space and avoid degenerate solutions, each gene is bounded:

$$x_i \in [\ell, u], \tag{4}$$

with $\ell = 0.01$ and $u = N/2$ in the reference implementation.

3.2 Fitness Function

Because GA selection typically rewards larger fitness values, we convert the minimization objective in (3) into a maximization fitness:

$$f(\mathbf{x}) = \frac{1}{1 + \left| N - \sum_{i=1}^{12} x_i \right|}. \tag{5}$$

The $+1$ prevents division by zero and stabilizes scaling.

3.3 Selection

We use roulette-wheel selection (fitness-proportionate selection). Let f_j be the fitness of chromosome j in a population of size P. Relative fitness is

$$r_j = \frac{f_j}{\sum_{k=1}^{P} f_k},$$

and cumulative fitness values define selection intervals on $[0, 1]$. Sampling a uniform random number determines which chromosome survives into the mating pool.

3.4 Crossover

Single-point crossover is applied with probability p_c. For a pair of parents, a crossover point $c \in \{0, \ldots, 11\}$ is chosen uniformly at random. Genes on the left side $\{0, \ldots, c-1\}$ are swapped between parents to produce two offspring.

3.5 Mutation

Mutation is applied independently per gene with probability p_m. Instead of swapping genes (which may not change the objective), we mutate by *replacing a gene value* with a newly sampled random value in $[\ell, u]$. This introduces numerical diversity and helps escape local stagnation.

3.6 Elitism

To avoid losing the best solution found so far, we preserve an elite chromosome across generations. If the best chromosome in the new generation improves upon the previous elite, it becomes the new elite; otherwise, the old elite replaces the worst chromosome of the current population.

4 Implementation and Reproducibility

The reference implementation is written in C++ and uses real-valued genes. Key parameters in the provided code are:

Parameter	Value (default)
Population size P	5
Generations G	5000
Genes per chromosome	12
Crossover rate p_c	0.15
Mutation rate p_m	0.05
Target sum N	150
Gene bounds $[\ell, u]$	[0.01, $N/2$]

4.1 Build and Run

A typical build command (GCC/Clang) is:

```
g++ genetic_algorithm.cpp -O2 -std=c++17 -o ga
./ga
```

The program prints the best chromosome, its fitness, and the achieved sum $\sum_i x_i$.

5 Applications

A common need in practice is to generate *controlled, bounded random partitions* of a target quantity. Examples include:

- Splitting a budget or resource into multiple non-uniform allocations under an equality constraint.
- Creating randomized order sizes that sum to a fixed liquidity value (useful for simulations or stress tests).
- Generating diverse feasible candidates for downstream optimization or scheduling stages.

6 Discussion: Initialization Matters

A recurring observation in GA performance is the sensitivity to initial population quality. Purely uniform-random initialization can slow convergence, especially as the dimensionality grows or constraints tighten. Prior research in combinatorial optimization (e.g., Traveling Salesman Problem) has explored using clustering methods (such as k-means) to create more structured initial populations, often improving convergence speed. Similar "informed initialization" ideas could be explored for constrained real-valued problems as future work.

7 Conclusion

This report presented an end-to-end GA solution for a constrained 12-variable partition problem. By reformulating the task as minimization of constraint violation and using a stable fitness transformation, the GA consistently finds near-feasible (and often feasible) solutions. The implementation is published as a reproducible software artifact, suitable for inclusion in an academic portfolio.

Artifacts and Links

- Zenodo (software, DOI): https://doi.org/10.5281/zenodo.17950322
- GitHub repository: https://github.com/milito-78/genetic-algorithm-dynamic-optimization

References

[1] "A genetic algorithm with k-means clustering initialization for the Traveling Salesman Problem," *Computers & Operations Research*, 2007. Available via ScienceDirect (article identifier: S004579490700301X).

[2] M. PourAhmad, "Genetic Algorithm for Dynamic Optimization (v1.0.0) [Software]," Zenodo, 2025. DOI: https://doi.org/10.5281/zenodo.17950322.