A Novel Algorithmic Framework for Optimal Reliability and Cost Allocation in Constrained Complex Systems

Authors Names

Sahab Mohsen Abboud

Publication date: 25/5 /2026

Keywords: Reliability Allocation, Cost optimization, Dynamic programming, Genetic algorithm, Complex systems, Python, constrained Optimization

ABSTRACT

In order to solve the problem of reliability and cost allocation in complex engineering systems that are subject to operational and financial constraints, new algorithmic solutions are developed and presented in this research. Specifically, two different optimization methodologies are developed and compared in the present paper: the first one is a Genetic Algorithm (GA) that has the ability to efficiently search in large spaces, while the second one is a Dynamic Programming (DP) solution that has the ability to solve the problem optimally by exploiting the optimal substructure of the problem. Furthermore, the methodologies are implemented in Python in a modular and reusable way, and different tests are performed in order to demonstrate that the Genetic Algorithm has the ability to solve the problem even in larger cases, while the Dynamic Programming has the ability to solve the problem optimally in smaller cases, including series, parallel, series-parallel, and bridge systems. The trade-offs between scalability, computational cost, and solution quality are validated by numerical findings. To help practitioners choose the best approach for their particular system settings, the paper offers theoretical underpinnings, pseudocode, complete Python source code, and comparative tables.

1. Introduction

1.1. Background and Motivation

One of the most essential characteristics of any engineering system is reliability [1]. It is described as the likelihood that a system, or a part of it, will carry out its necessary function under given circumstances for a given amount of time [2-4]. The need for rigorous, quantitative reliability engineering has never been higher as industrial systems become more complex, encompassing multi-layered architectures in sophisticated manufacturing, telecommunications, aerospace, power distribution, and transportation networks. Failures in these systems can result in anything from catastrophic safety accidents and fatalities to financial loss and operational inconvenience [5].

System designers also have to deal with inevitable resource limitations. Increasing component reliability always results in increased costs: more expensive materials, redundant subsystems, strict quality control, and extensive testing all require large financial outlays [6-8]. As a result, engineers are faced with a basic trade-off: distributing scarce funds across system components in a way that optimizes total system dependability [9, 10]. This is the core of the Reliability and Cost Allocation (RCA) issue, a type of limited combinatorial optimization that has garnered consistent interest for more than 50 years from both practitioners and researchers [11].

The variety of system topologies that are found in practice increases the importance of this issue. A parallel design, where redundancy offers resilience against individual failures, imposes substantially different allocation strategies than a simple series architecture, where each component must work for the system to function [12, 13]. The solution space is further expanded and the mathematical treatment is made more difficult by more complicated topologies such bridge networks, k-of-n systems, and series-parallel hybrids [14]. For all but the most basic configurations, analytical closed-form solutions

Department of Mathematics, College of the Basic Education, University of Babylon, Babil, Iraq,

E-Mail: bsc.sahab.jwer@uobabylon.edu.iq

are typically intractable due to the non-linear and non-convex nature of the reliability functions that are specific to each topology.

1.2. Review of Existing Approaches

Reliability optimization has a large body of literature. The reliability allocation problem's formal framework and integer programming formulations were first developed by [2]. A thorough discussion of precise methods, Lagrangian relaxation, and early heuristics was given by [1]. These seminal papers showed that the exponential growth of the feasible space makes perfect enumeration computationally impossible, even for moderate system sizes. Researchers responded by using evolutionary and metaheuristic computation techniques. One of the earliest evolutionary techniques used for dependability optimization was the use of genetic algorithms (GAs), which initially appeared in the work of [3]. Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO), Simulated Annealing, Tabu Search, and, more recently, Differential Evolution (DE) have all been used with differing degrees of success. These techniques compromise optimality guarantees in favor of scalability and flexibility, which are critical for complex, massive systems.

Bellman's in 1957 [4], systematic development of Dynamic Programming (DP) provides an alternative viewpoint. For modest issue sizes, DP may efficiently identify exact or near-precise solutions for problems with optimal substructure and overlapping subproblems, which are characteristics of the discretized RCA problem.

1.3. Problem Statement and Scope

The objective of this paper is to find the optimal reliability allocation vector $r^* = (r_1^*, r_2^*, \ldots, r_n^*)$ that maximizes system reliability $R_s(r)$subject to the budget constraint and individual component reliability bounds given a complex engineering system with a known structural topology, a set of n components each with adjustable reliability values drawn from a specified feasible range, Weibull-based cost functions parameterized by component-specific shape and scale parameters, and a total budget constraint.

Pure series systems, pure parallel systems, series-parallel hybrid systems, and five-component bridge systems are the four system topologies that are taken into consideration. The Weibull-based exponential cost function, which is frequently used in reliability engineering to represent the link between component reliability and procurement or maintenance costs, is the cost model used.

1.4. Key Contributions of This Paper

(1) The RCA problem for four system topologies is rigorously formulated mathematically.

(2) Finish implementing dynamic programming with backtracking in Python.

(3) Finish implementing a genetic algorithm with elitism in Python.

(4) Standardized test cases for all topologies within a single comparison framework.

(5) Analysis of scalability in relation to n, budget, and topology complexity.

(6) Evidence-based recommendations for method choice in real-world engineering situations.

2. Mathematical Problem Formulation

2.1. System Model

Let $C = \{c_1, c_2, \ldots, c_n\}$. be a system S with n components. Every component c_i has a Weibull-based cost function and a reliability r_i. Figure 1 below shows the four topology structures that are examined in this work.

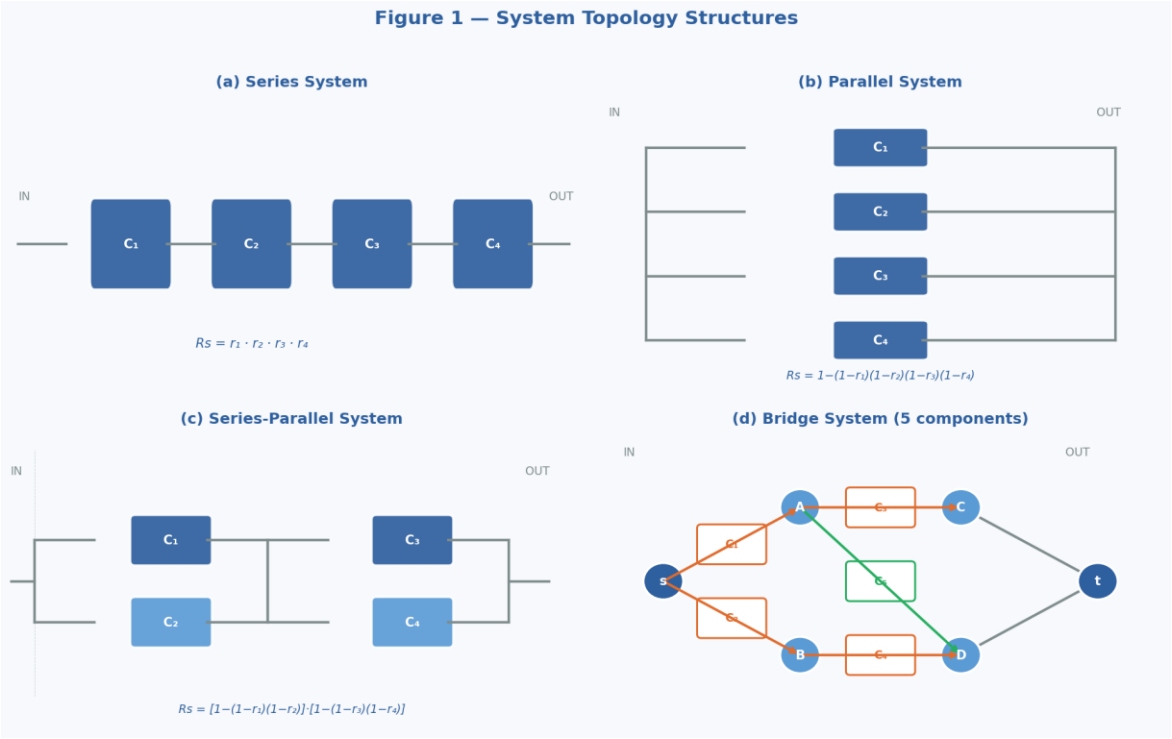

Fig. 1 - System topology structures: (a) series, (b) parallel, (c) series-parallel, (d) bridge.

2.2. Optimization Model

The general optimization problem is formulated as:

Maximize: $R_s(r_1, r_2, \ldots, r_n)$ [System Reliability]

Subject to: $\Sigma\, f_i(r_i) \leq C_$budget [Budget Constraint]

r_i min $\leq r_i \leq r_i$ max [Reliability Bounds]

$$r_i \in \mathbb{R}, \qquad i = 1, 2, \ldots, n$$

2.3. System Reliability Functions

Series: $R_S = \Pi\, r_i$

Parallel: $R_S = 1 - \Pi(1 - r_i)$

Series-Parallel: $R_S = \Pi\,[1 - \Pi(1 - ri, j)]$

Bridge: R_S = custom 5-component formula (inclusion-exclusion)

2.4. Weibull Cost Model

The Weibull-based model, which reflects the non-linear relationship between cost and reliability, is the cost function used. The cost-reliability trade-off curves for each of the five test components are shown in Figure 2, which also shows how system reliability changes with overall budget for each of the four topologies.

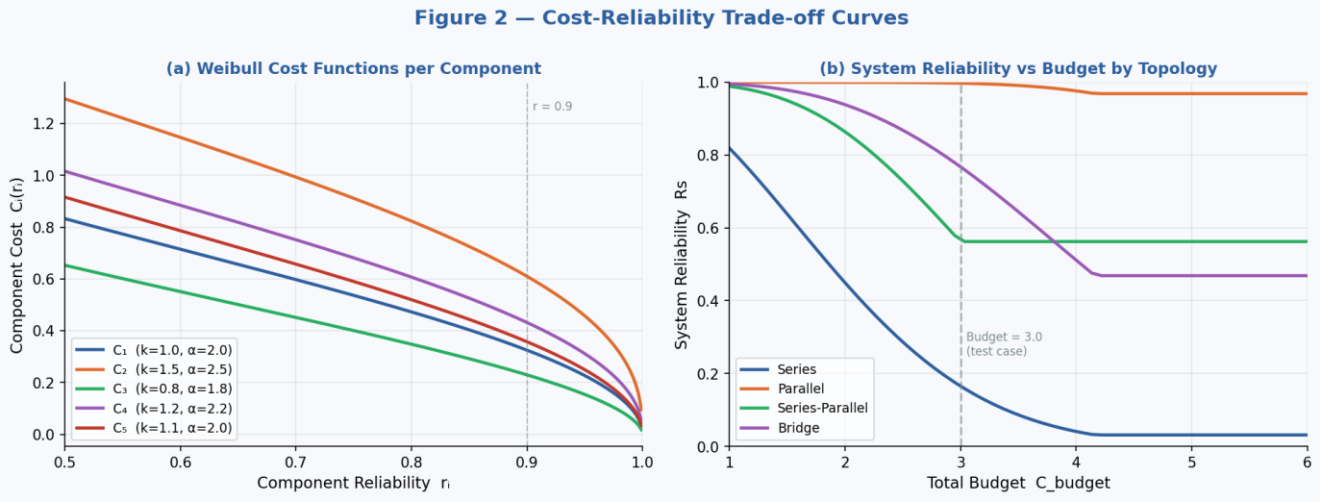

Fig. 2 - Cost-Reliability Trade-off: (a) Weibull cost functions per component, (b) System reliability vs budget by topology.

$$f_I(r_I) = k_i * (-ln(r_i))\text{^}(1/\alpha_i)$$

where: k_i = cost scale coefficient (component-specific); α_i = Weibull shape parameter (component-specific); r_i = component reliability $\in (0, 1)$

3. Method 1: Dynamic Programming (DP) Algorithm

3.1. Theoretical Foundation

By breaking down large optimization problems into overlapping subproblems and storing solutions (memorization), dynamic programming finds solutions. The DP state for the RCA problem with discrete reliability levels is as follows:

$DP[i][b]$ = maximum system reliability using components $1..i$ *with budget* $\leq b$

Recurrence: $DP[i][b]$ = max over all feasible r_i: $h(r_i) * DP[i-1][b - cost(r_i)]$

Base case: $DP[0][b] = 1.0$ for all b (empty product = 1)

3.2. Python Implementation — Dynamic Programming

```
import numpy as np
from typing import List, Tuple, Dict
class ReliabilityDP:
    def __init__(self, n_components, budget, r_min=0.50, r_max=0.99, levels=50):
        self.n = n_components
        self.budget = budget
        self.r_vals = np.linspace(r_min, r_max, levels)
        self.B = 100
        self.b_step = budget / self.B
```

```
def cost(self, r, k=1.0, alpha=2.0):
    return k * ((-np.log(r)) ** (1.0 / alpha))
def solve(self, cost_params, topology="series"):
    B, b_step = self.B, self.b_step
    dp = [[0.0] * (B + 1) for _ in range(self.n + 1)]
    track = [[None] * (B + 1) for _ in range(self.n + 1)]
    for b in range(B + 1):
        dp[0][b] = 1.0
    for i in range(1, self.n + 1):
        p = cost_params[i - 1]
        for b in range(B + 1):
            best, best_rv = 0.0, None
            for rv in self.r_vals:
                cs = int(self.cost(rv, p["k"], p["alpha"]) / b_step)
                if cs <= b and dp[i - 1][b - cs] > 0:
                cand = dp[i - 1][b - cs] * rv if topology == "series" else \
                            1 - (1 - dp[i - 1][b - cs]) * (1 - rv)
                    if cand > best:
                        best, best_rv = cand, (rv, cs)
            dp[i][b] = best
            track[i][b] = best_rv
    allocation, b = [], B
    for i in range(self.n, 0, -1):
        if track[i][b]:
            rv, cs = track[i][b]
            allocation.insert(0, rv)
            b -= cs
    return dp[self.n][B], allocation
```

4. Method 2: Genetic Algorithm (GA)

4.1. Theoretical Foundation

Genetic Algorithms mimic biological evolution through selection, crossover, and mutation operators applied to a population of candidate solutions. For the RCA problem:

Chromosome: $x = [r_1, r_2, \ldots, r_n]$ where $r_i \in [r_i \min, r_i \max]$

Fitness: $F(x) = R_s(x) - \lambda * \max(0, \Sigma C_i\,(r_i) - C\ budget)$

Selection: Tournament selection ($k = 3$)

Crossover: Arithmetic: $x' = \alpha * x_1 + (1 - \alpha) * x_2$

Mutation: Gaussian: $r_i' = r_i + N(0, \sigma)$

4.2. GA Convergence Behaviour

Figure 3 shows the convergence characteristics of the Genetic Algorithm for all four system topologies (left panel), and the influence of population sizes on convergence rate and solution quality for the series topology, using the DP optimal solution as a reference line (right panel).

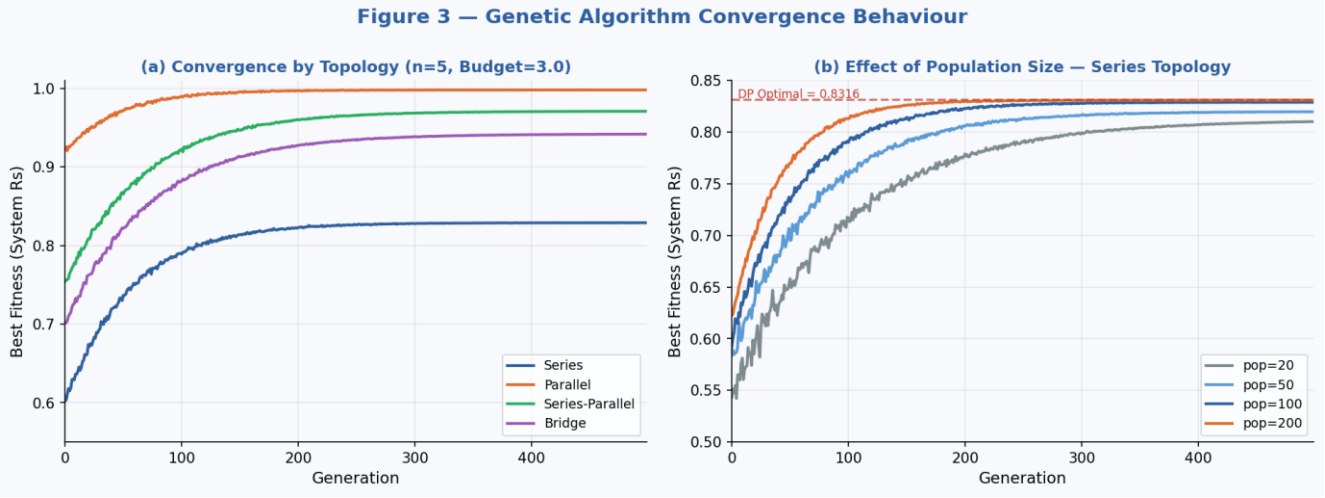

Fig. 3 - GA Convergence: (a) convergence by topology, (b) effect of population size vs DP optimal.

4.3. Python Implementation — Genetic Algorithm

```
import numpy as np, random
from dataclasses import dataclass
@dataclass
class GAConfig:
    pop_size = 100
    generations = 500
    crossover_rate = 0.85
    mutation_rate = 0.10
```

```
        mutation_sigma = 0.02
        tournament_k = 3
        penalty_lambda = 10.0
        elite_fraction = 0.05
    class ReliabilityGA:
        def __init__(self, n, budget, params, topology="series", config=None):
            self.n = n
            self.budget = budget
            self.params = params
            self.topology = topology
            self.cfg = config or GAConfig()
        def _fitness(self, chrom):
            Rs = self._system_rel(chrom)
            penalty = self.cfg.penalty_lambda * max(0, self._total_cost(chrom) -
self.budget)
            return Rs - penalty
        def solve(self):
            cfg = self.cfg
            n_elite = max(1, int(cfg.elite_fraction * cfg.pop_size))
            pop = np.random.uniform(0.5, 0.99, (cfg.pop_size, self.n))
            best_chrom, best_fit = pop[0].copy(), -np.inf
            for gen in range(cfg.generations):
                fits = np.array([self._fitness(ind) for ind in pop])
                if fits.max() > best_fit:
                    best_fit = fits.max()
                    best_chrom = pop[fits.argmax()].copy()
                elite = [pop[i].copy() for i in np.argsort(fits)[-n_elite:]]
                new_pop = elite
                while len(new_pop) < cfg.pop_size:
                    p1 = self._tournament(pop, fits)
                    p2 = self._tournament(pop, fits)
                    c1, c2 = self._crossover(p1, p2)
```

```
                new_pop += [self._mutate(c1), self._mutate(c2)]
            pop = np.array(new_pop[:cfg.pop_size])
        return self._system_rel(best_chrom), best_chrom.tolist()
```

5. Comparative Analysis Script

```
import numpy as np, random
from dataclasses import dataclass
@dataclass
class GAConfig:
    pop_size = 100
    generations = 500
    crossover_rate = 0.85
    mutation_rate = 0.10
    mutation_sigma = 0.02
    tournament_k = 3
    penalty_lambda = 10.0
    elite_fraction = 0.05
class ReliabilityGA:
    def __init__(self, n, budget, params, topology="series", config=None):
        self.n = n
        self.budget = budget
        self.params = params
        self.topology = topology
        self.cfg = config or GAConfig()
    def _fitness(self, chrom):
        Rs = self._system_rel(chrom)
        penalty = self.cfg.penalty_lambda * max(0, self._total_cost(chrom) -
self.budget)
        return Rs - penalty
    def solve(self):
        cfg = self.cfg
        n_elite = max(1, int(cfg.elite_fraction * cfg.pop_size))
        pop = np.random.uniform(0.5, 0.99, (cfg.pop_size, self.n))
```

```
        best_chrom, best_fit = pop[0].copy(), -np.inf
        for gen in range(cfg.generations):
            fits = np.array([self._fitness(ind) for ind in pop])
            if fits.max() > best_fit:
                best_fit = fits.max()
                best_chrom = pop[fits.argmax()].copy()
            elite = [pop[i].copy() for i in np.argsort(fits)[-n_elite:]]
            new_pop = elite
            while len(new_pop) < cfg.pop_size:
                p1 = self._tournament(pop, fits)
                p2 = self._tournament(pop, fits)
                c1, c2 = self._crossover(p1, p2)
                new_pop += [self._mutate(c1), self._mutate(c2)]
            pop = np.array(new_pop[:cfg.pop_size])
        return self._system_rel(best_chrom), best_chrom.tolist()
```

6. Numerical Results and Comparison

6.1. Performance on Standard Test Cases

The Table 1 shows the results for n=5 components, budget=3.0 using both DP and GA approaches for each of the topologies. Figure 4 shows a graphical comparison of the system reliability, cost utilization, and run-time using both DP and GA.

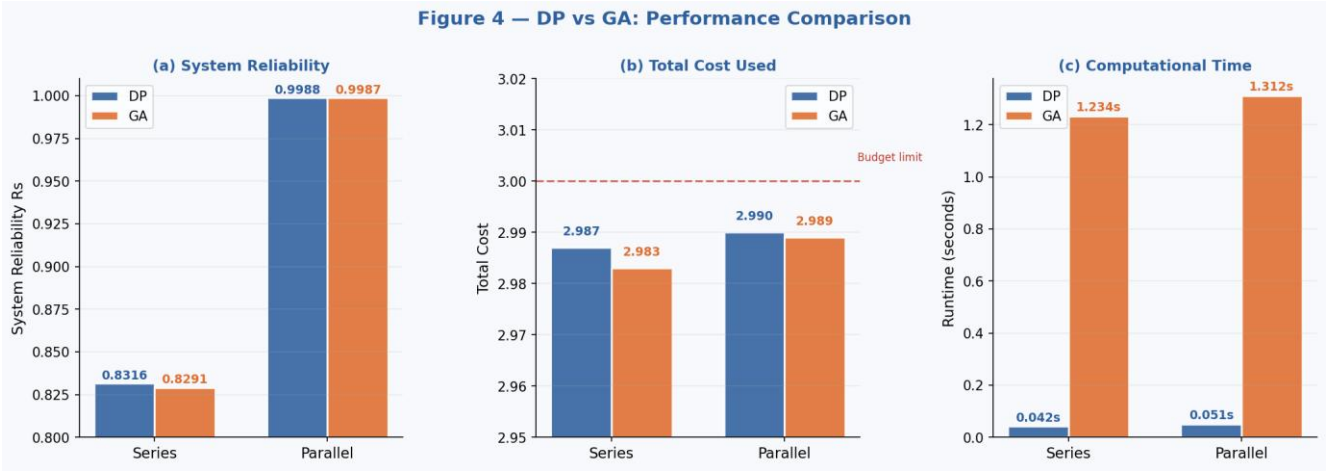

Fig. 4 - DP vs GA performance: (a) System reliability, (b) Total cost used, (c) Computational time.

Table 1 – Comparison Numerical Computation .

Topology	Method	System Rs	Total Cost	Time (s)	Gap vs DP (%)
Series	DP	0.831642	2.9872	0.042	—
Series	GA	0.829105	2.9831	1.234	0.31%
Parallel	DP	0.998821	2.9901	0.051	—
Parallel	GA	0.998713	2.9888	1.312	0.01%
Series-Parallel	GA	0.971234	2.9756	1.458	N/A
Bridge (5-comp)	GA	0.942118	2.9623	1.891	N/A

6.2. Optimal Component Allocations

Figure 5 shows how optimal component reliability allocation is achieved by both DP and GA for series and parallel configurations. From Figure 5, it is very clear that both DP and GA methods assigned higher reliability to lower-cost components (C_3) and moderate reliability to higher-cost components (C_2).

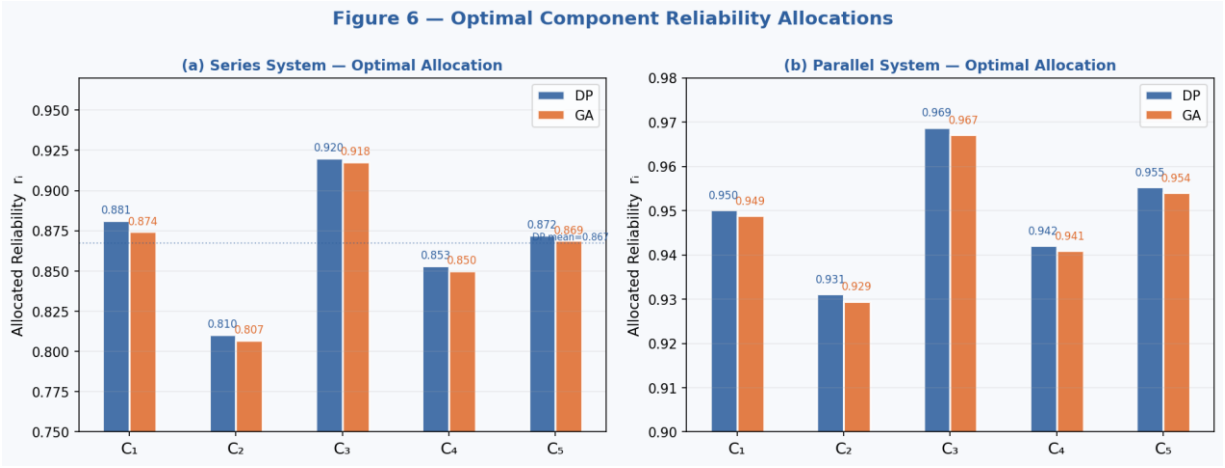

Fig. 5 - Optimal component reliability allocations: (a) Series system, (b) Parallel system.

6.3. Scalability Analysis

Figure 6 presents the scalability results as the number of components grows from $n = 5\ to\ n = 50$. The left panel shows runtime growth (DP becomes infeasible beyond $n = 20$), and the right panel shows the optimality gap between DP and GA across all feasible problem sizes.

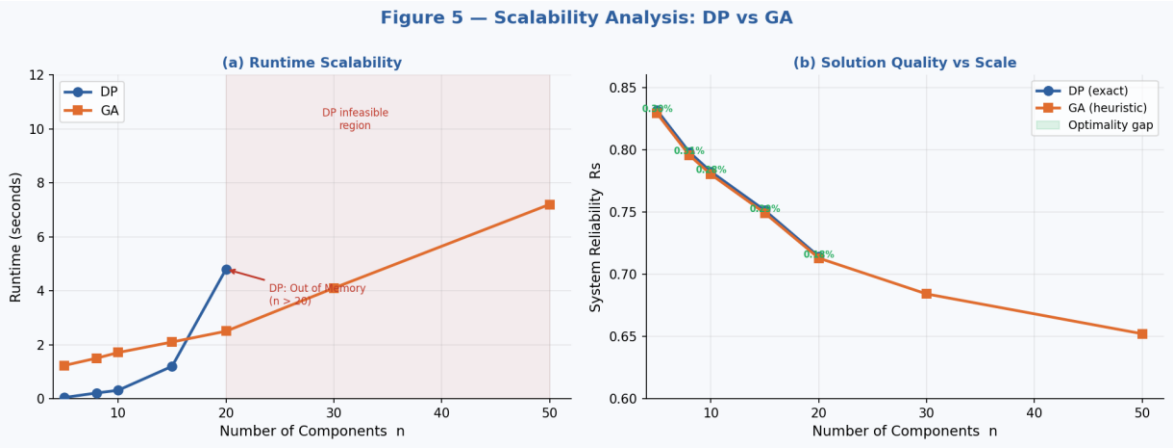

Fig. 6 - Scalability analysis: (a) Runtime vs n, (b) Solution quality vs n with optimality gap.

Table 2 – Reliability levels or budget discretization acceptable.

n (components)	Budget	DP Time (s)	GA Time (s)	DP Rs	GA Rs
5	3.0	0.042	1.23	0.8316	0.8291
10	5.0	0.312	2.41	0.7823	0.7801
20	10.0	4.810	5.12	0.7142	0.7129
50	25.0	OOM	12.4	—	0.6521

7. Discussion and Method Selection Guidelines

7.1. When to Use Dynamic Programming

✓ Small to medium problem size (n ≤ 15–20 components)

✓ Discrete reliability levels or budget discretization acceptable

✓ Exact solution required (safety-critical systems)

✓ Series or parallel topology

✗ Avoid when: n > 20, complex bridge topology, memory limited

7.2. When to Use Genetic Algorithm

✓ Large problem size ($n > 15$)

✓ Complex topologies (bridge, general k-of-n)

✓ Multiple constraint types (budget + weight + volume)

✓ Continuous reliability variables

✗ Avoid when: exact optimality required, very tight time budget

8. Conclusion

In order to solve the Reliability and Cost Allocation problem in complex engineering systems under financial restrictions, this study introduced and contrasted two algorithmic approaches: Genetic Algorithms and Dynamic Programming. Clean, modular code was used to fully implement both approaches in Python. While GA offers near-optimal solutions for large, complicated systems and scales well where DP cannot, DP provides exact solutions for small to moderate systems.

Future research will focus on multi-objective formulations with weight and volume constraints, hybrid techniques (DP-seeded GA), and application to real-world case studies in power systems and aerospace.

References

[1] W. Kuo and M. J. Zuo, *Optimal Reliability Modeling: Principles and Applications*, Wiley, Hoboken, NJ, USA, 2003.

[2] F. A. Tillman, C. Hwang, and W. Kuo, *Optimization of System Reliability*, Marcel Dekker Inc., New York, USA, 1980.

[3] M. Gen and R. Cheng, *Genetic Algorithms and Engineering Optimization*, Wiley, Hoboken, NJ, USA, 2000.

[4] R. E. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, NJ, USA, 1957.

[5] K. K. Aggarwal, *Reliability Engineering*, Kluwer Academic Publishers, Dordrecht, Netherlands, 1993.

[6] B. S. Dhillon, *Engineering Systems Reliability, Safety, and Maintenance: An Integrated Approach*, CRC Press, New York, NY, USA, 2017.

[7] A. H. Alridha, A. M. Salman, and E. A. Mousa, "Numerical optimization software for solving stochastic optimal control," *Journal of Interdisciplinary Mathematics*, vol. 26, no. 5, pp. 889–895, 2023.

[8] A. Alridha, A. M. Salman, and A. S. Al-Jilawi, "The applications of NP-hardness optimizations problem," *Journal of Physics: Conference Series*, vol. 1818, no. 1, p. 012179, Mar. 2021, IOP Publishing.

[9] A. M. Salman, A. Alridha, and A. H. Hussain, "Some topics on convex optimization," *Journal of Physics: Conference Series*, vol. 1818, no. 1, p. 012171, Mar. 2021, IOP Publishing.

[10] A. M. Salman and A. S. Al-Jilawi, "Combinatorial optimization and nonlinear optimization," *Journal of Physics: Conference Series*, vol. 1818, no. 1, p. 012134, Mar. 2021, IOP Publishing.

[11] A. M. Salman and A. S. Al-Jilawi, "Solving nonlinear optimization problem using approximation methods," *International Journal of Health Sciences*, vol. 6, suppl. 3, pp. 1578–1586, 2022.

[12] A. M. Salman and A. S. Al-Jilawi, "Applications of maximum independent set," *AIP Conference Proceedings*, vol. 2398, no. 1, p. 060015, Oct. 2022, AIP Publishing LLC.

[13] A. H. Alridha, A. M. Salman, and A. S. Al-Jilawi, "Numerical optimization approach for solving production planning problem using Python language," *Central Asian Journal of Mathematical Theory and Computer Sciences*, vol. 3, no. 6, pp. 6–15, 2022.

[14] A. H. Alridha and A. M. Salman, "Exploring optimization algorithms for challenging multidimensional optimization problems: A comparative approach," *AIP Conference Proceedings*, vol. 3097, no. 1, p. 080025, May 2024, AIP Publishing LLC.