

In The Name of God



Navid Zare

Evolutionary Computing

Assignment 1: The 8-Queens Problem

Task 2: Parameter Sensitivity Analysis

Mutation Probability:

- **Success Rate:** All mutation probabilities (0.2, 0.5, 1.0) achieved 100% success rate for $N=8$
- **Convergence Speed:** Higher mutation rates led to faster convergence:
 - 0.2: ~231 generations average (high variance: ± 668)
 - 0.5: ~89 generations average (moderate variance: ± 96)
 - 1.0: ~59 generations average (low variance: ± 65)
- **Evaluations:** Decreased with higher mutation rates ($562 \rightarrow 277 \rightarrow 218$)
- **Analysis:** Higher mutation probability (1.0) provides better exploration in the permutation search space. The swap mutation preserves the permutation property naturally, making it effective for the N-Queens problem. The large variance at 0.2 indicates inconsistent performance, while 1.0 shows more reliable convergence.
- **Best Setting:** `mutation_prob = 1.0`

Recombination Rate:

- **Success Rate:** Both rates (0.5, 1.0) achieved 100% success rate
- **Convergence Speed:**
 - 0.5: ~145 generations average (high variance: ± 202)
 - 1.0: ~85 generations average (low variance: ± 83)
- **Evaluations:** 389 vs 270 evaluations
- **Analysis:** Higher recombination rate (1.0) ensures every generation produces offspring through crossover, leading to faster convergence and more consistent results. The 0.5 rate shows larger spread and variance in the box plot, indicating less predictable behavior.
- **Best Setting:** `recombination_rate = 1.0`

Mutation Type:

- **Bitwise Mutation:** Not applicable for permutation representation. When attempted, bitwise mutation (changing a gene to a random value) breaks the permutation constraint, requiring constant chromosome repair. This dramatically slows the GA, making it computationally infeasible for testing. The repair operation adds significant overhead and disrupts the evolutionary process.

- **Swap Mutation:** Naturally preserves permutation validity by simply exchanging two positions, making it the ideal choice for the N-Queens problem.
-

Task 3: Crossover Strategy Exploration

Performance Comparison:

- **Success Rate:** All crossover strategies achieved 100% success rate for N=8
- **Convergence Speed:**
 - cut_and_fill: ~ 81 generations (± 93)
 - pmx: ~ 57 generations (± 78)
 - 2_cut: ~ 86 generations (± 107)
 - 3_cut: ~ 82 generations (± 76)

Key Findings:

- **Cut-and-Fill :** Cut-and-fill consistently converges faster than PMX in most runs and shows overall lower variance, although the winner can still change from run to run.
- **PMX:** PMX achieves a strong average generation count in this dataset, but its variance is high, meaning outcomes are inconsistent. In some runs it converges quickly, and in others it takes much longer.
- **N-cut analysis:** Increasing cuts from 2 to 3 actually **worsens** performance (78 \rightarrow 89 generations), showing diminishing returns. More cut points create more disruption without proportional improvement in offspring quality.

Why Cut-and-Fill May be better for N-Queen Problem:

Cut-and-fill preserves a large block from one parent and fills remaining positions sequentially. For N-Queens, this often produces children with fewer diagonal conflicts because:

- Large consistent segments remain intact
- The filling step avoids destructive conflicts
- Diversity is maintained without excessive disruption

This balance makes it effective for permutation-based constraint problems like N-Queens.

Task 4: Survival Strategy Comparison

Performance Results:

- **Success Rate:** All strategies achieved 100% success rate
- **Convergence Speed:**
 - **Elitism:** ~2.3 generations (± 2.0) - **FASTEST** (~20% faster than generational)
 - **Generational:** ~3.0 generations (± 1.9)
 - **Fitness-based:** ~94 generations (± 136) - Significantly slower

Analysis:

Elitism (Best Performer):

- Preserves top 2 individuals each generation
- Rapid convergence by exploiting best solutions
- Low variance indicates stable, predictable behavior
- Uses exploitation technique to maintain and refine optimal solutions

Generational Replacement:

- Replaces entire population each generation
- Slightly slower than elitism but still very fast
- Higher genetic diversity through complete population turnover
- Good for early exploration phases

Fitness-based:

- Much slower convergence (~94 gen vs ~2-3 gen)
- Higher variance (± 136) shows inconsistent performance
- Allows weaker solutions to persist longer
- Better for avoiding premature convergence in complex landscapes

When to Use Each:

Use Elitism when:

- Solution quality is critical
- Fast convergence is desired
- Problem has clear fitness landscape (like N-Queens)
- Preventing loss of best solutions is important

Use Generational Replacement when:

- Need high genetic diversity
- Early optimization exploration
- Complex/dynamic solution spaces
- Risk of premature convergence to local optima

Best Practice: Start with Generational Replacement for exploration, transition to Elitism as solutions converge.

Task 5: Scalability Study

Results Summary:

N	Success Rate	Avg Generations	Avg Evaluations	Best Fitness
8	100.0%	113 (± 161)	325 (± 322)	0 (optimal)
10	100.0%	394 (± 505)	888 (± 1010)	0 (optimal)
12	83.3%	592 (± 458)	1283 (± 916)	-1 (near-optimal)
20	86.7%	2625 (± 1915)	5449 (± 3830)	-1 (near-optimal)

Scalability Analysis:

What Gets Worse:

1. **Success Rate Drops:** From 100% ($N \leq 10$) to ~83-87% ($N \geq 12$)
2. **Generations Increase Exponentially:** $113 \rightarrow 2625$ (23x increase from $N=8$ to $N=20$)

3. **Variance Explodes:** Standard deviation grows dramatically, showing increasingly unpredictable behavior
4. **Solution Quality Degrades:** $N \geq 12$ sometimes fails to find optimal solution (fitness = -1)

Why It Gets Worse:

- **Search Space Explosion:** $N!$ possible permutations grow exponentially ($8! = 40,320$ vs $20! = 2.4 \times 10^{18}$)
- **Complexity Growth:** The normalized complexity chart shows near-quadratic growth (Generations/ N^2), confirming the problem difficulty scales with board size
- **Fixed Population:** Using pop_size=100-200 becomes insufficient for larger N values
- **Fitness Landscape:** More queens create exponentially more potential conflicts, making the fitness landscape more rugged

Observations:

- $N=8$ and $N=10$ are solved reliably with current settings
- $N=12$ shows the transition point where success rate drops
- $N=20$ requires significantly more computational resources (50,000 max evaluations vs 10,000) yet still struggles
- High variance at larger N indicates the GA behavior becomes increasingly stochastic and unpredictable