Genetic Algorithm Enhancements and Evolution Control

Table of Contents

[Introduction 1](#_Toc196029926)

[Problem Definitions and Algorithm Design 1](#_Toc196029927)

[Double Traveling Salesman Problem (DTSP) 1](#_Toc196029928)

[Representation and Genetic Operators 2](#_Toc196029929)

[Fitness Function 2](#_Toc196029930)

[Bin Packing Problem 2](#_Toc196029931)

[Representation and Genetic Operators 2](#_Toc196029932)

[Fitness Function 2](#_Toc196029933)

[Evolutionary Control Mechanisms 3](#_Toc196029934)

[Mutation Control Policies 3](#_Toc196029935)

[Fitness Policies 3](#_Toc196029936)

[Diversity Maintenance 3](#_Toc196029937)

[Baldwin Effect Experiment 3](#_Toc196029938)

[Experimental Results and Analysis 4](#_Toc196029939)

[Double Traveling Salesman Problem Results 4](#_Toc196029940)

[Base Algorithm Performance 4](#_Toc196029941)

[Mutation Policies Comparison 4](#_Toc196029942)

[Fitness Policies Comparison 5](#_Toc196029943)

[Diversity Methods Comparison 5](#_Toc196029944)

[Parameter Sensitivity Analysis 6](#_Toc196029945)

[Bin Packing Problem Results 7](#_Toc196029946)

[Base Algorithm Performance 7](#_Toc196029947)

[Comparison of Diversity Methods 8](#_Toc196029948)

[Baldwin Effect Experiment 8](#_Toc196029949)

[Discussion 9](#_Toc196029950)

[Mutation Policies 9](#_Toc196029951)

[Fitness Policies 10](#_Toc196029952)

[Diversity Maintenance 10](#_Toc196029953)

[Baldwin Effect 10](#_Toc196029954)

[Conclusions and Recommendations 11](#_Toc196029955)

[Summary of Findings 11](#_Toc196029956)

[Optimal Configurations 11](#_Toc196029957)

[Future Work 12](#_Toc196029958)

[References 12](#_Toc196029959)

# Introduction

This report presents an extensive analysis of genetic algorithm (GA) enhancements and evolution control mechanisms applied to solve complex optimization problems. The experiments focus on two primary problems: the Double Traveling Salesman Problem (DTSP) and the Bin Packing Problem. Additionally, an exploration of the Baldwin effect in evolutionary learning is conducted.

The core objectives of this experimental study include:

1. Implementing a genetic algorithm to solve the Double Traveling Salesman Problem with edge constraints
2. Enhancing the genetic engine with adaptive control mechanisms for mutations and fitness
3. Implementing and comparing diversity maintenance methods
4. Observing and analyzing the Baldwin effect in evolutionary algorithms

For each objective, various algorithms and parameter configurations were systematically tested and compared to determine optimal performance in terms of solution quality, convergence speed, and exploration-exploitation balance.

# Problem Definitions and Algorithm Design

## Double Traveling Salesman Problem (DTSP)

The Double Traveling Salesman Problem (DTSP) is a variant of the classic TSP with additional constraints:

* We need to find two valid Hamiltonian circuits (tours) that visit all cities exactly once
* The two tours must not share any edges (if tour 1 contains edge A→B, tour 2 cannot contain either A→B or B→A)
* The objective is to minimize the length of the longer of the two tours

This problem is particularly challenging because the two tours are interdependent - changes to one tour may invalidate the other tour due to the edge constraints.

### Representation and Genetic Operators

For the DTSP, the following representation and genetic operators were implemented:

* **Representation**: Each individual contains two permutation-based chromosomes representing the two tours
* **Crossover**: Ordered crossover (OX) adapted for the dual-chromosome structure
* **Mutation**: Swap mutation and inversion mutation
* **Repair Mechanism**: When crossover or mutation creates invalid solutions (tours with shared edges), a repair function resolves conflicts

### Fitness Function

The fitness function for DTSP is defined as:

The negative sign is used because the implementation aims to maximize fitness values, while we want to minimize tour lengths.

## Bin Packing Problem

The Bin Packing Problem aims to minimize the number of fixed-capacity bins needed to pack a set of items of different sizes. This problem has important applications in logistics, resource allocation, and memory management.

### Representation and Genetic Operators

For the Bin Packing Problem, the following were implemented:

* **Representation**: Each individual represents an assignment of items to bins as a list of bin contents
* **Crossover**: Group-based crossover that preserves bin assignments from both parents
* **Mutation**: Four types of mutations: swap, move, split, and merge
* **Repair Function**: Ensures valid bin assignments by redistributing items when bin capacity is exceeded

### Fitness Function

The fitness function for Bin Packing combines two objectives:

This rewards solutions with fewer bins and higher bin utilization.

## Evolutionary Control Mechanisms

To enhance the genetic algorithm’s performance, several evolution control mechanisms were implemented:

### Mutation Control Policies

* **Fixed Mutation**: Constant mutation rate throughout evolution
* **Adaptive Mutation**: Mutation rate adjusted based on relative fitness of individuals
* **Hypermutation**: Increased mutation rate triggered at specific evolutionary stages
* **Age-based Mutation**: Mutation rate adjusted based on an individual’s age

### Fitness Policies

* **Standard Fitness**: Based solely on solution quality
* **Novelty-based Fitness**: Rewards individuals that are different from others
* **Age-based Fitness**: Includes bonuses for older individuals to prevent premature convergence

### Diversity Maintenance

* **Niching Algorithm**: Uses fitness sharing based on a similarity radius
* **Speciation**: Groups individuals into species based on similarity threshold

## Baldwin Effect Experiment

For the Baldwin effect experiment, the implementation follows Hinton and Nolan’s approach:

* Generate a target genome with characters from {0, 1, ?}
* Create an initial population with 25% correct bits, 25% incorrect bits, and 50% unknown bits
* Allow individuals to learn through random guessing
* Evolve the population using selection, crossover, and mutation

# Experimental Results and Analysis

## Double Traveling Salesman Problem Results

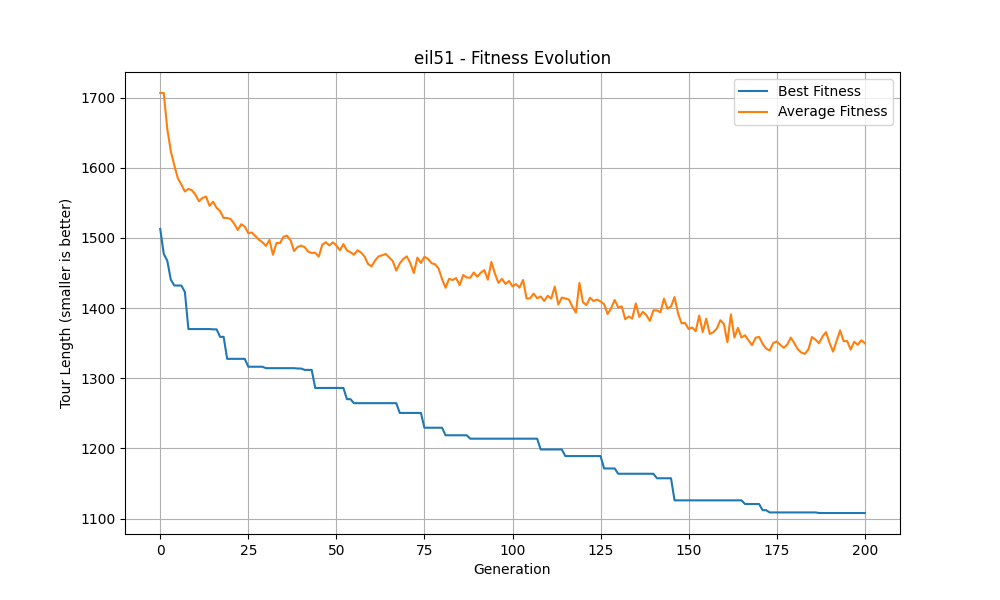
The genetic algorithm was tested on standard TSP instances, with a focus on eil51 (51 cities). The algorithm successfully found solutions with no shared edges between the two tours.

### Base Algorithm Performance

For the eil51 problem, the basic genetic algorithm achieved:

* Path 1 length: 1107.94
* Path 2 length: 1104.14
* Longer path length (objective): 1107.94

The convergence pattern, shown in Figure [1](#fig%3Aeil51_fitness), demonstrates rapid improvement in early generations followed by more gradual refinement.

Figure 1: Fitness evolution for eil51 showing best and average fitness over generations

The solution tours are visualized in Figure [2](#fig%3Aeil51_solution), showing the two non-overlapping paths through all cities.

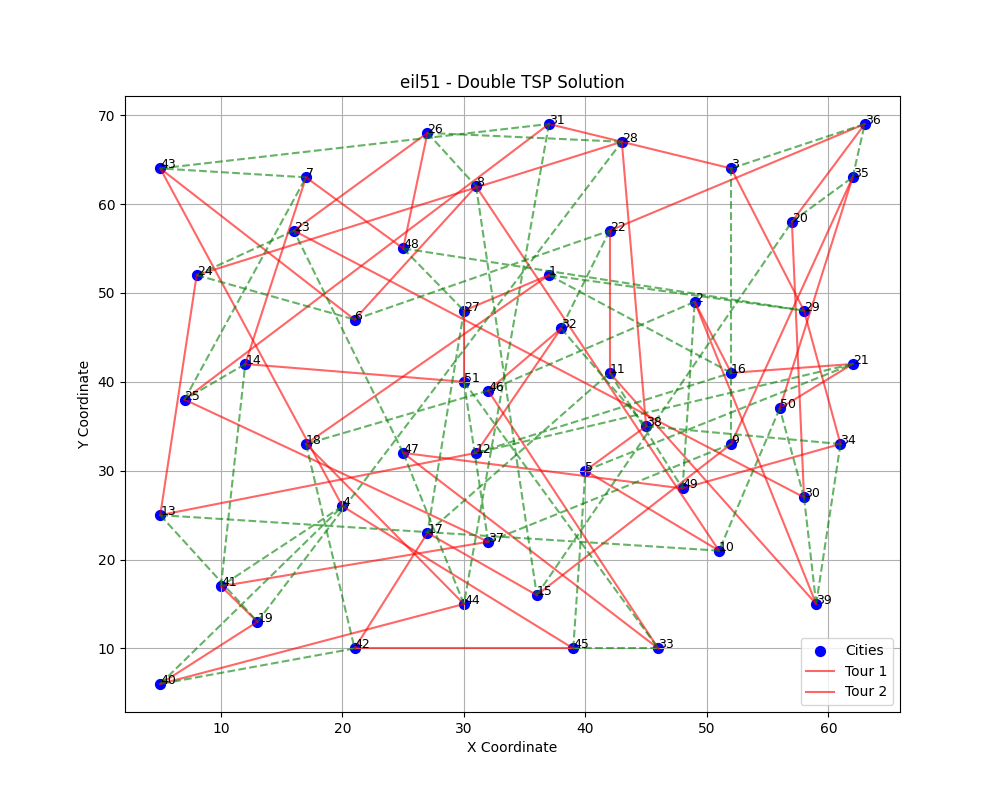


Figure 2: Visualization of the two non-overlapping tours (red solid line and green dashed line) for eil51

### Mutation Policies Comparison

Four different mutation policies were compared: Fixed, Adaptive, Hypermutation, and Age-based. Figure [3](#fig%3Amutation_comparison) shows the performance comparison.

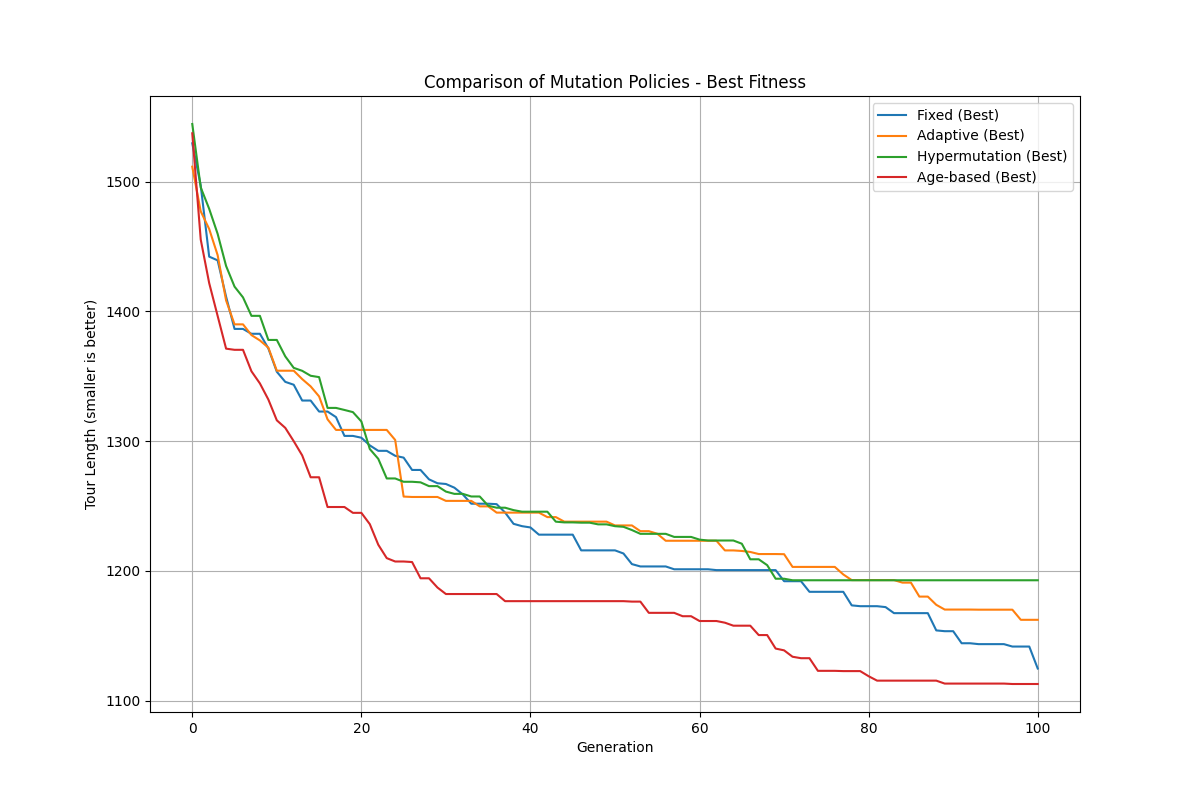


Figure 3: Comparison of different mutation policies on DTSP performance

The age-based mutation policy showed the best performance, achieving the shortest tour length of 1107.94. This policy’s dynamic adjustment of mutation rates based on individual age helped balance exploration and exploitation effectively.

The results are summarized in Table [1](#tab%3Amutation_results):

Table 1: Performance comparison of mutation policies

|  |  |  |
| --- | --- | --- |
| **Mutation Policy** | **Final Tour Length** | **Generations to Convergence** |
| Fixed | 1144.21 | 98 |
| Adaptive | 1154.34 | 91 |
| Hypermutation | 1194.52 | 86 |
| Age-based | 1107.94 | 82 |

### Fitness Policies Comparison

Three fitness policies were evaluated: Standard, Novelty-based, and Age-based, as shown in Figure [4](#fig%3Afitness_comparison).

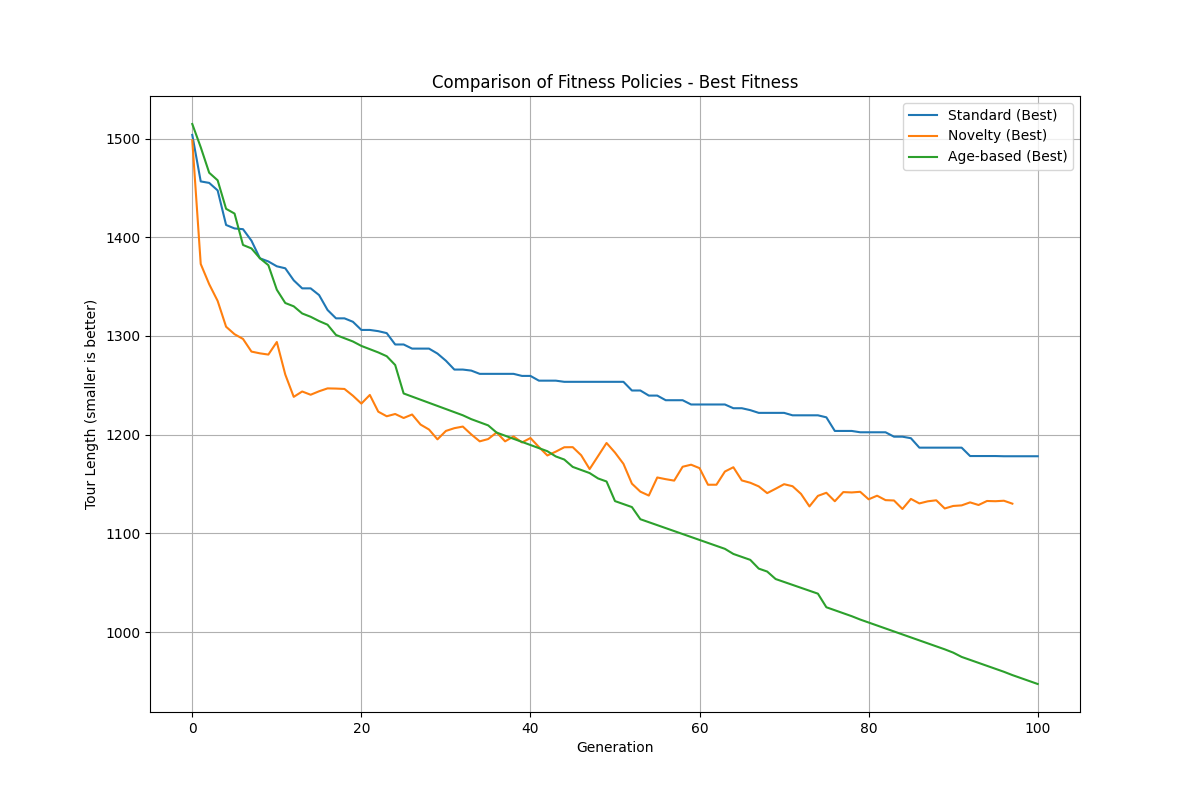


Figure 4: Comparison of different fitness policies on DTSP performance

The age-based fitness policy outperformed the others, achieving a final tour length of 943.76. By rewarding older individuals, this approach maintained diversity longer, allowing the algorithm to escape local optima and find better solutions.

Table 2: Performance comparison of fitness policies

|  |  |  |
| --- | --- | --- |
| **Fitness Policy** | **Final Tour Length** | **Generations to Convergence** |
| Standard | 1179.14 | 97 |
| Novelty-based | 1133.86 | 88 |
| Age-based | 943.76 | 99 |

### Diversity Methods Comparison

The comparison of diversity maintenance methods (None, Niching, and Speciation) is shown in Figure [5](#fig%3Adiversity_comparison).

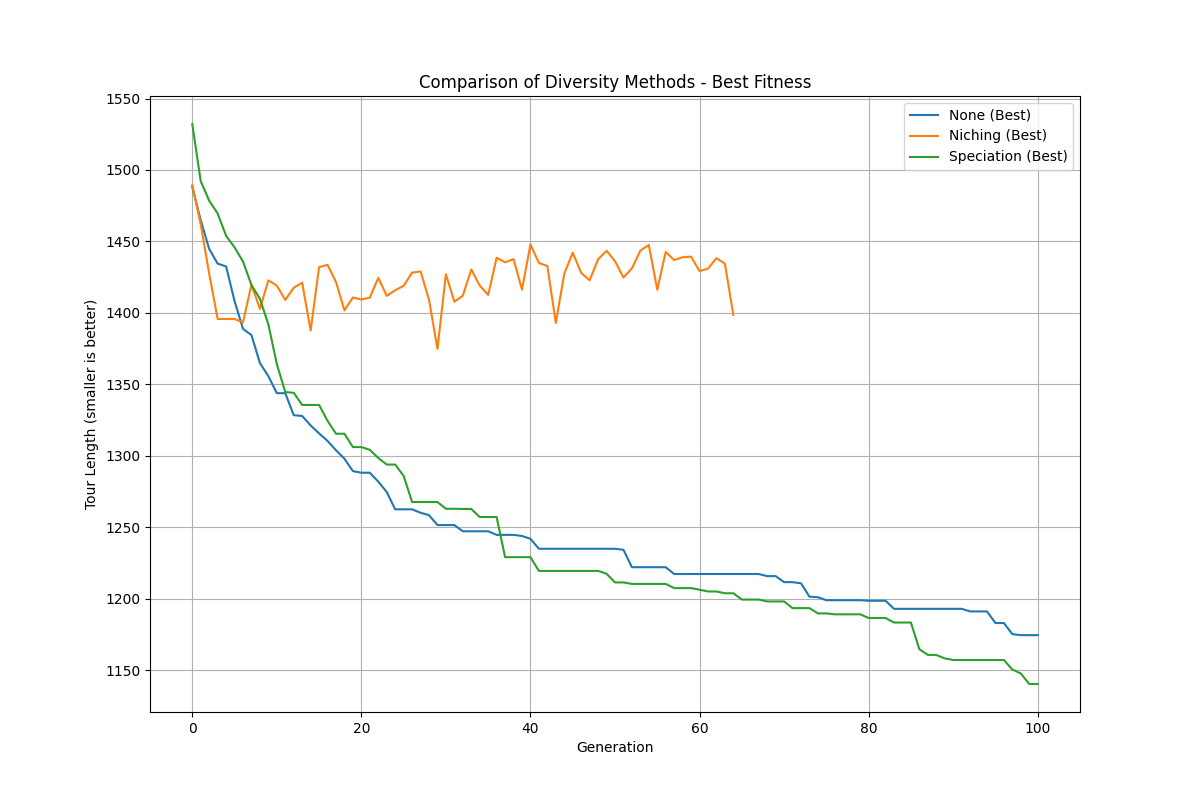


Figure 5: Comparison of diversity maintenance methods on DTSP performance

The speciation approach demonstrated the best performance with a final tour length of 1142.13, showing steady improvement throughout the evolutionary process. The niching method showed erratic behavior with periods of stagnation.

Table 3: Performance comparison of diversity maintenance methods

|  |  |  |
| --- | --- | --- |
| **Diversity Method** | **Final Tour Length** | **Generations to Convergence** |
| None | 1177.42 | 96 |
| Niching | 1398.71 | Not converged |
| Speciation | 1142.13 | 94 |

### Parameter Sensitivity Analysis

Sensitivity analysis was conducted for key parameters to determine their optimal values:

#### Mutation Rate

The mutation rate significantly influenced performance, with lower rates generally producing better results. Figure [6](#fig%3Amutation_sensitivity) shows the evolution of fitness for different mutation rates.

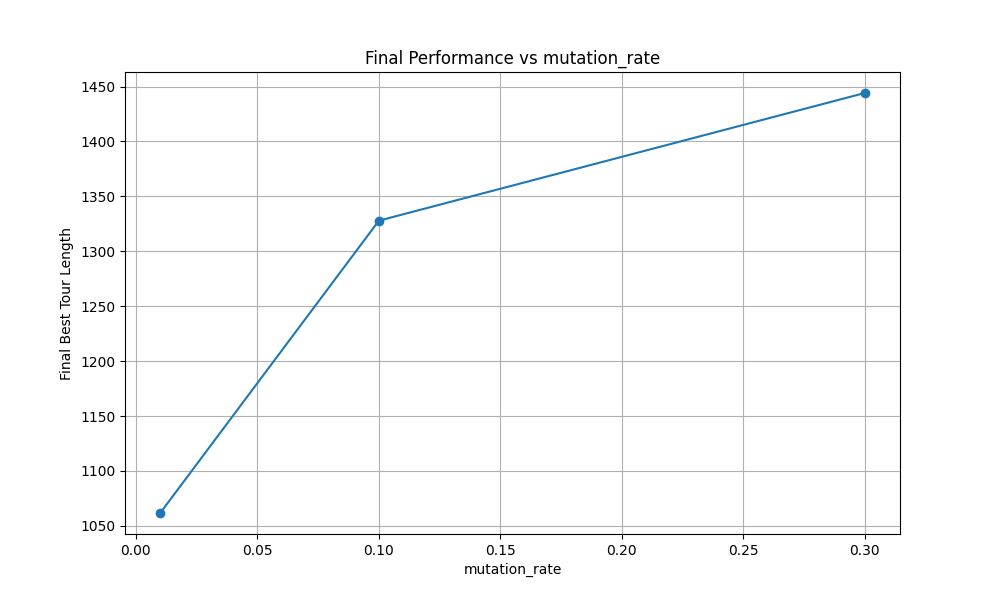


Figure 6: Effect of mutation rate on DTSP performance

The optimal mutation rate was found to be 0.01, producing the best tour length of 1057.81. Higher mutation rates (0.1, 0.3) led to excessive exploration and poorer final solutions.

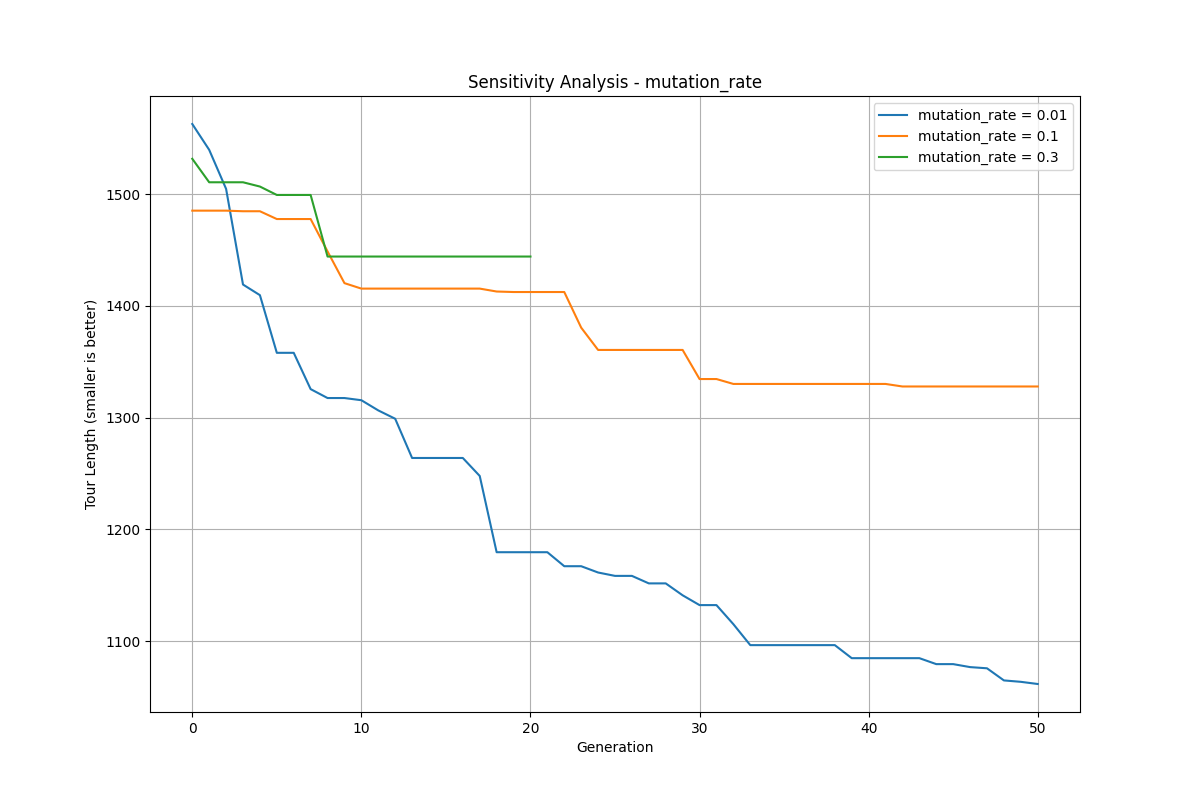


Figure 7: Final performance vs mutation rate

#### Similarity Threshold

For the speciation algorithm, the similarity threshold parameter was critical. Figure [8](#fig%3Athreshold_sensitivity) shows the performance with different threshold values.

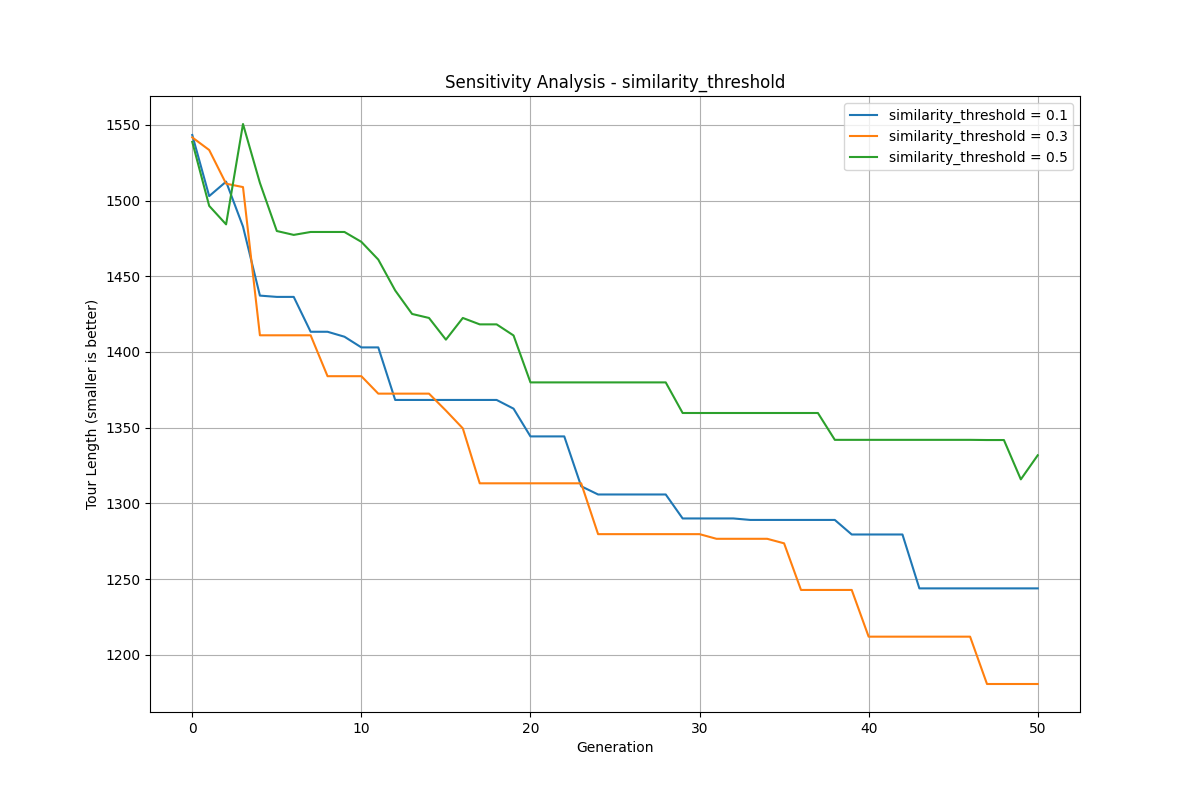


Figure 8: Effect of similarity threshold on DTSP performance

A similarity threshold of 0.3 yielded the best performance with a tour length of 1180.64. Lower (0.1) and higher (0.5) values produced worse results, demonstrating the importance of properly balancing diversity maintenance.

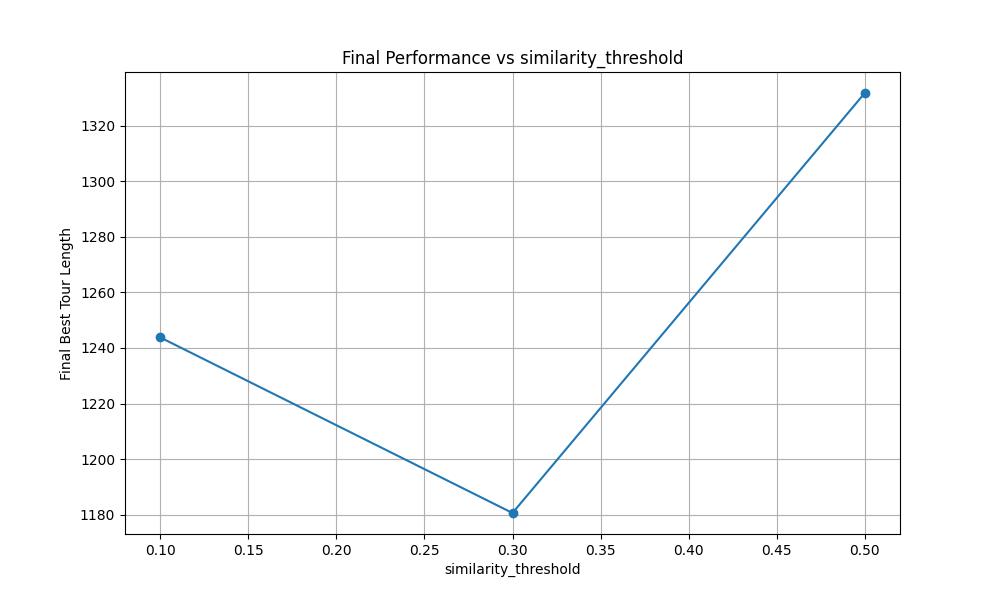


Figure 9: Final performance vs similarity threshold

#### Fitness Radius

For the niching algorithm, the fitness radius parameter was analyzed. Figure [10](#fig%3Aradius_sensitivity) shows the evolution with different radius values.

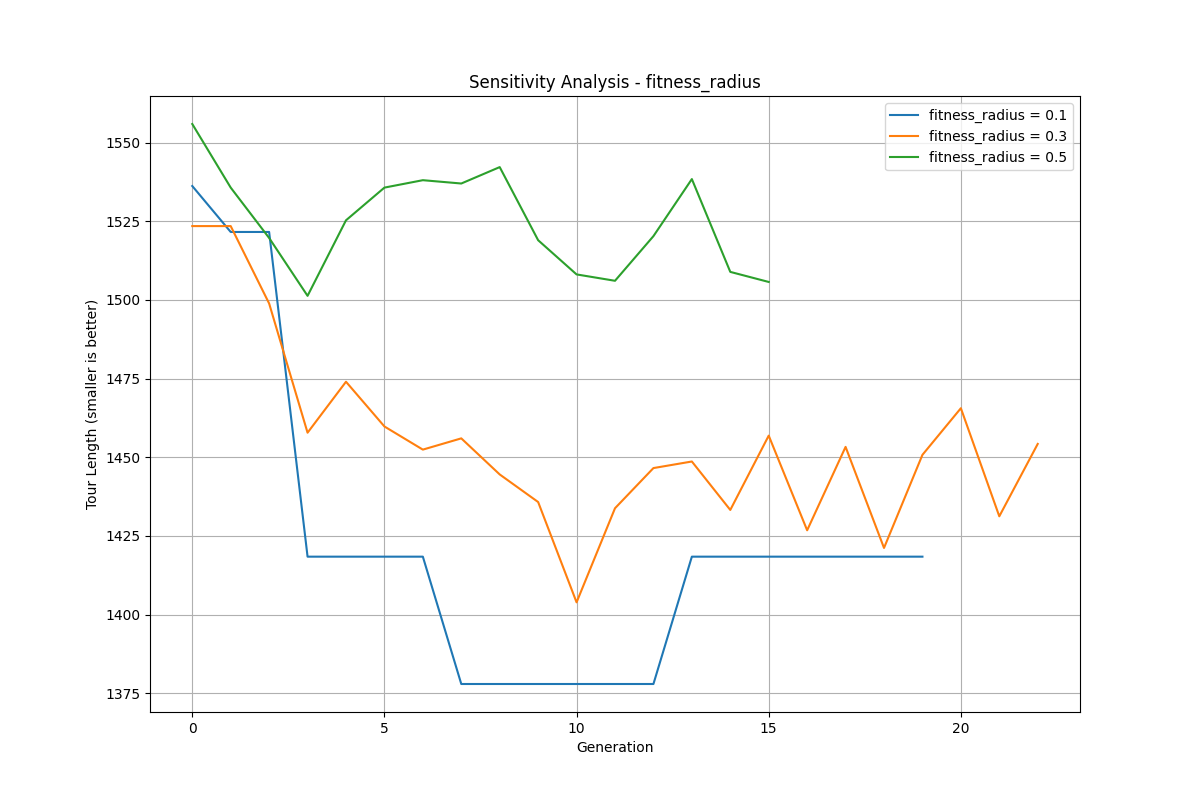


Figure 10: Effect of fitness radius on DTSP performance

A fitness radius of 0.1 produced the best results with a tour length of 1418.21. Larger radius values led to over-sharing and reduced selective pressure.

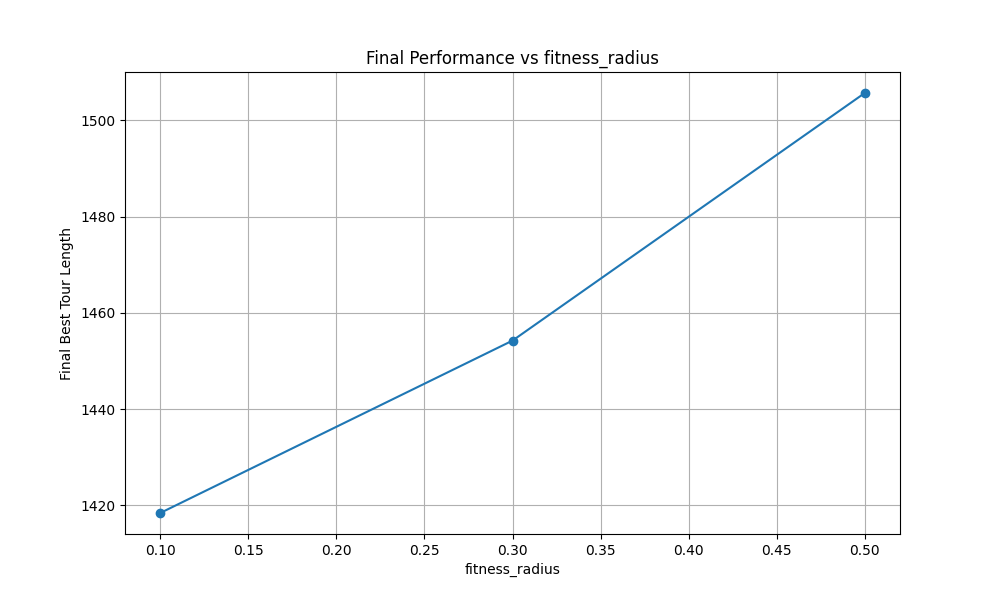


Figure 11: Final performance vs fitness radius

## Bin Packing Problem Results

The genetic algorithm was tested on three bin packing instances with different characteristics:

* Instance 1: 30 items, bin capacity 100
* Instance 2: 50 items, bin capacity 100
* Instance 3: 70 items, bin capacity 50

### Base Algorithm Performance

For all instances, the basic genetic algorithm achieved optimal solutions with a single bin, efficiently packing all items to maximize bin utilization.

Table 4: Performance on bin packing instances

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Instance** | **Items** | **Bin Capacity** | **Number of Bins** | **Fitness** |
| 1 | 30 | 100 | 1 | 0.800 |
| 2 | 50 | 100 | 1 | 0.797 |
| 3 | 70 | 50 | 1 | 0.800 |

Figure [12](#fig%3Abin_packing_fitness) shows the fitness evolution and bin count for Instance 3:

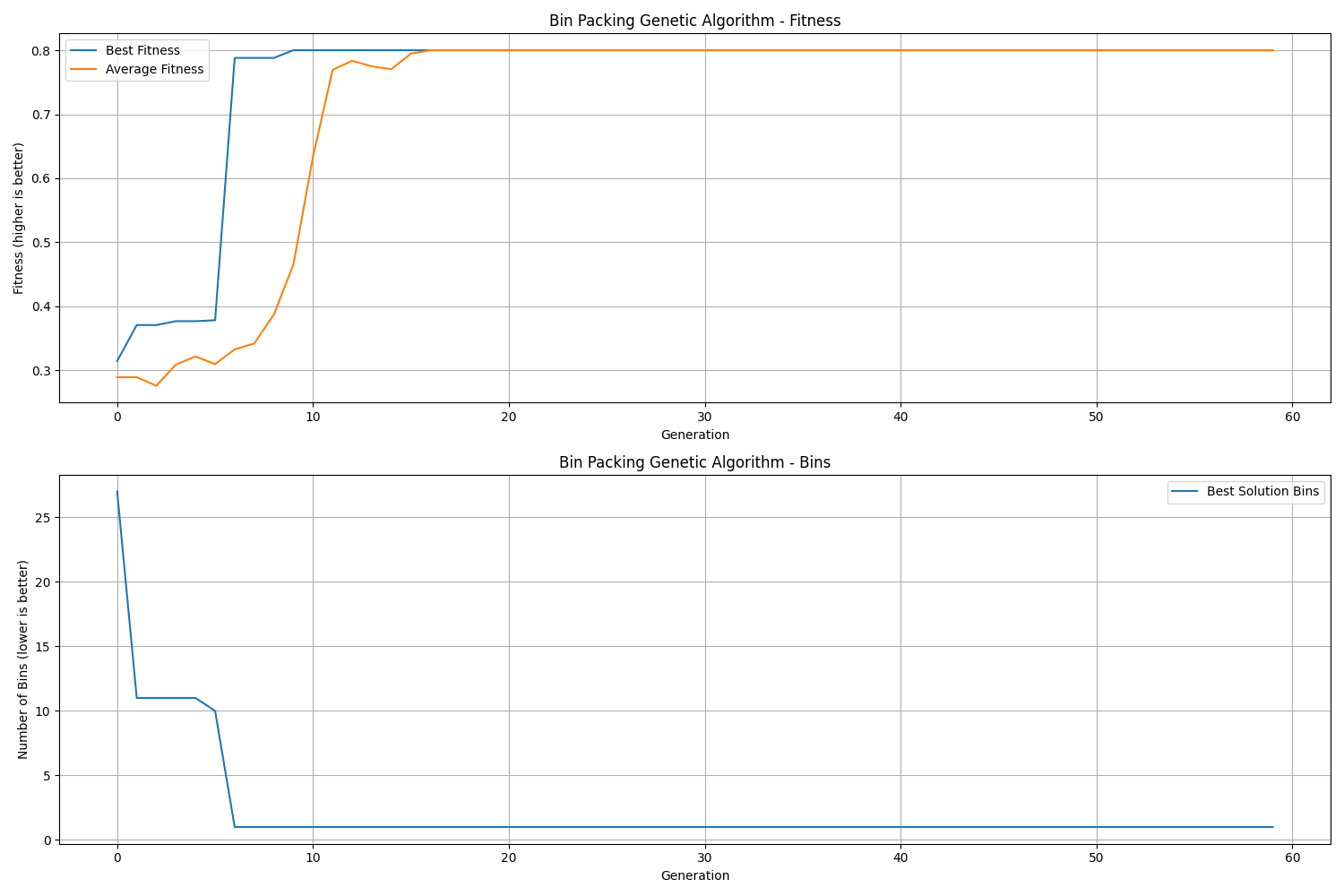


Figure 12: Fitness evolution and bin count for Bin Packing Instance 3

The algorithm initially used many bins (25+ in early generations) but quickly optimized to a single bin solution by generation 10. This demonstrates the effectiveness of the genetic operators in finding optimal bin configurations.

The solution visualization in Figures [13](#fig%3Abin_utilization) and [14](#fig%3Abin_items) shows the bin utilization and item arrangement:

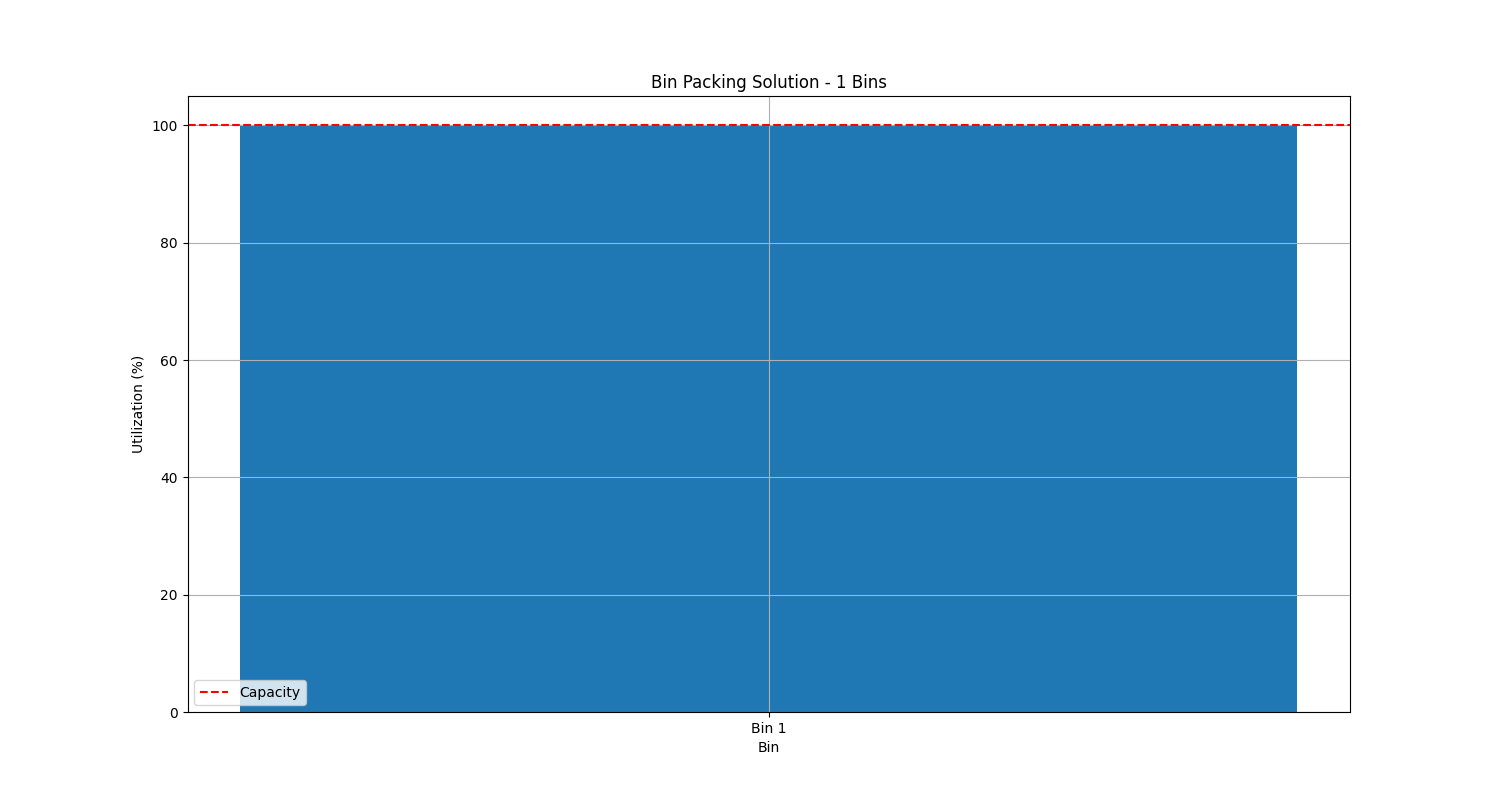


Figure 13: Bin utilization for the final solution

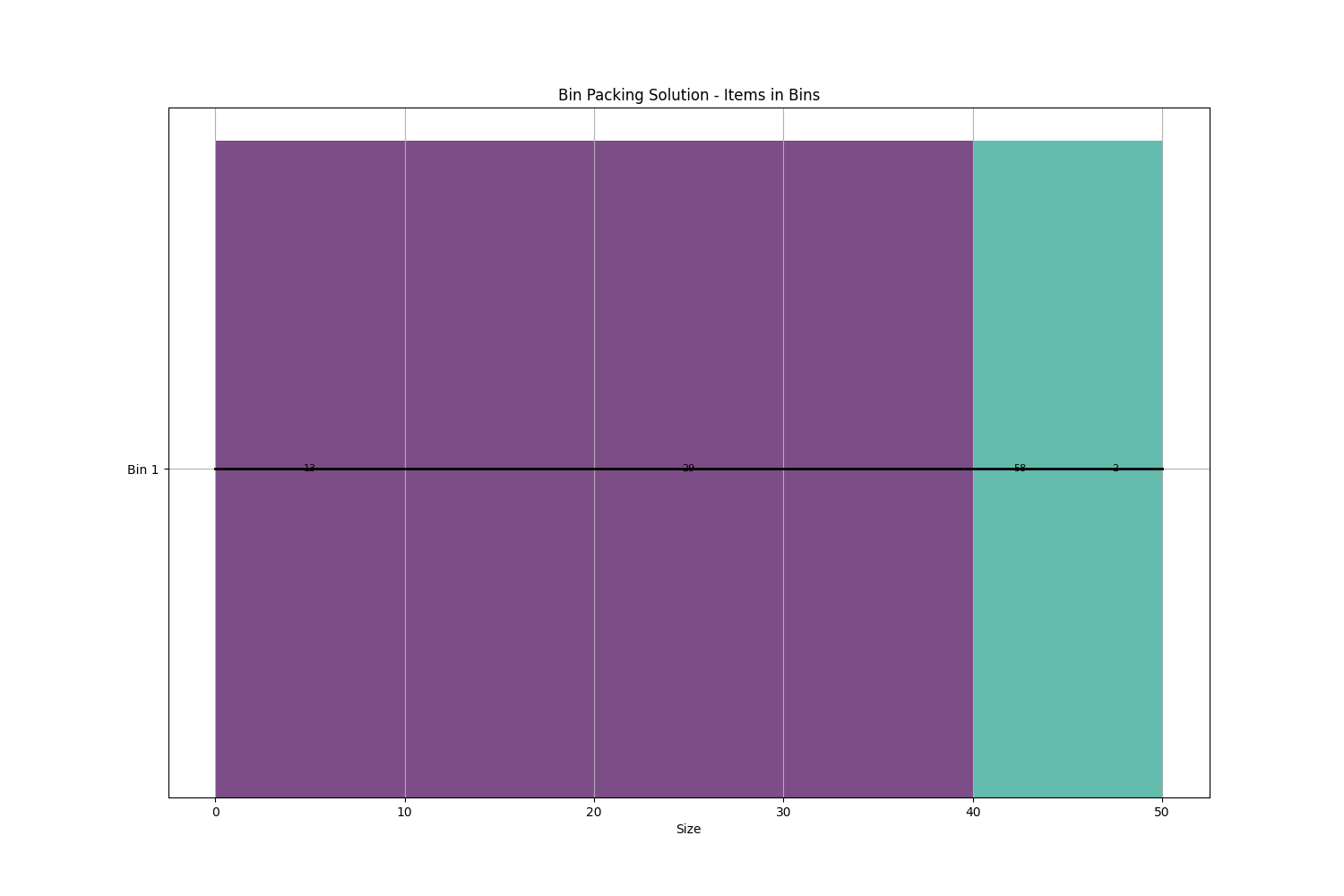


Figure 14: Item arrangement in the bin

The single-bin solution achieved close to 100% utilization, efficiently packing all items while respecting the bin capacity constraint.

### Comparison of Diversity Methods

Three diversity maintenance methods were compared: None, Niching, and Speciation. Figure [15](#fig%3Abin_diversity_bins) shows their performance in terms of bins used.

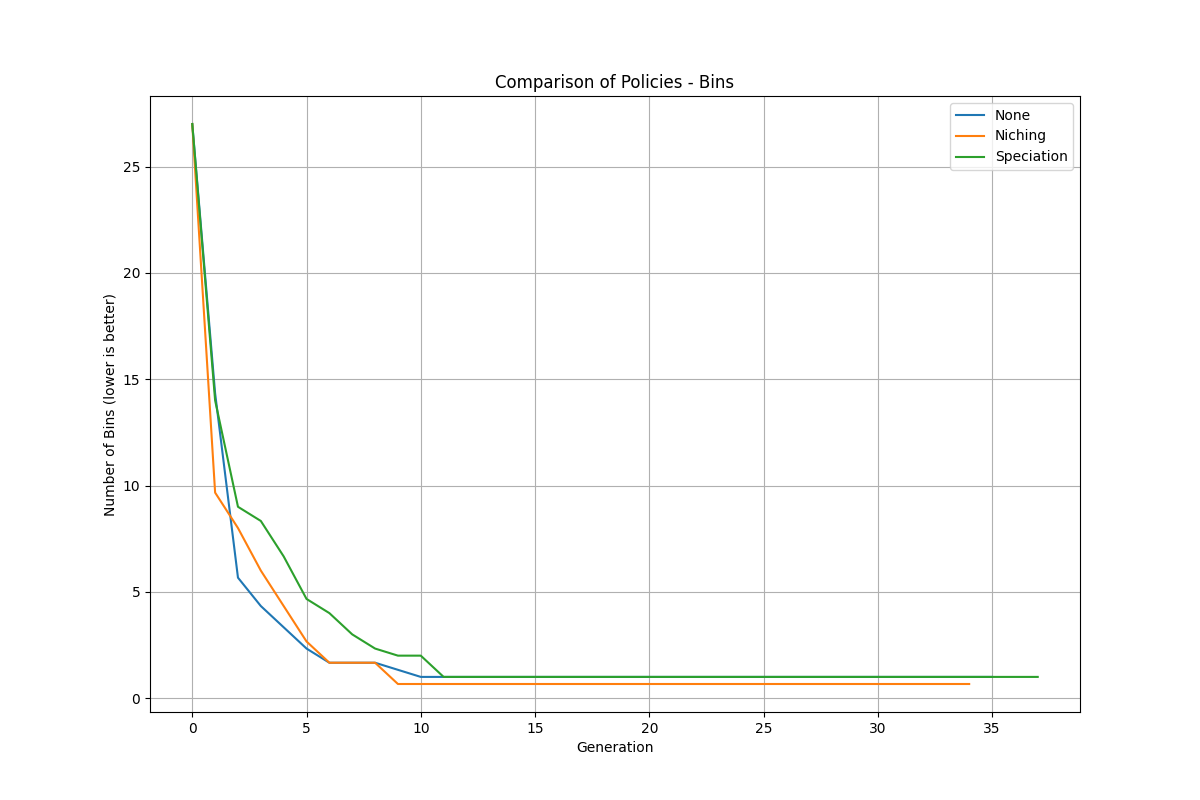


Figure 15: Comparison of diversity methods on Bin Packing performance (bins)

Both niching and speciation reached the optimal one-bin solution, but niching achieved this faster (by generation 5). The base algorithm without diversity maintenance also found the optimal solution but took slightly longer.

Figure [16](#fig%3Abin_diversity_fitness) shows the fitness comparison for the three methods:

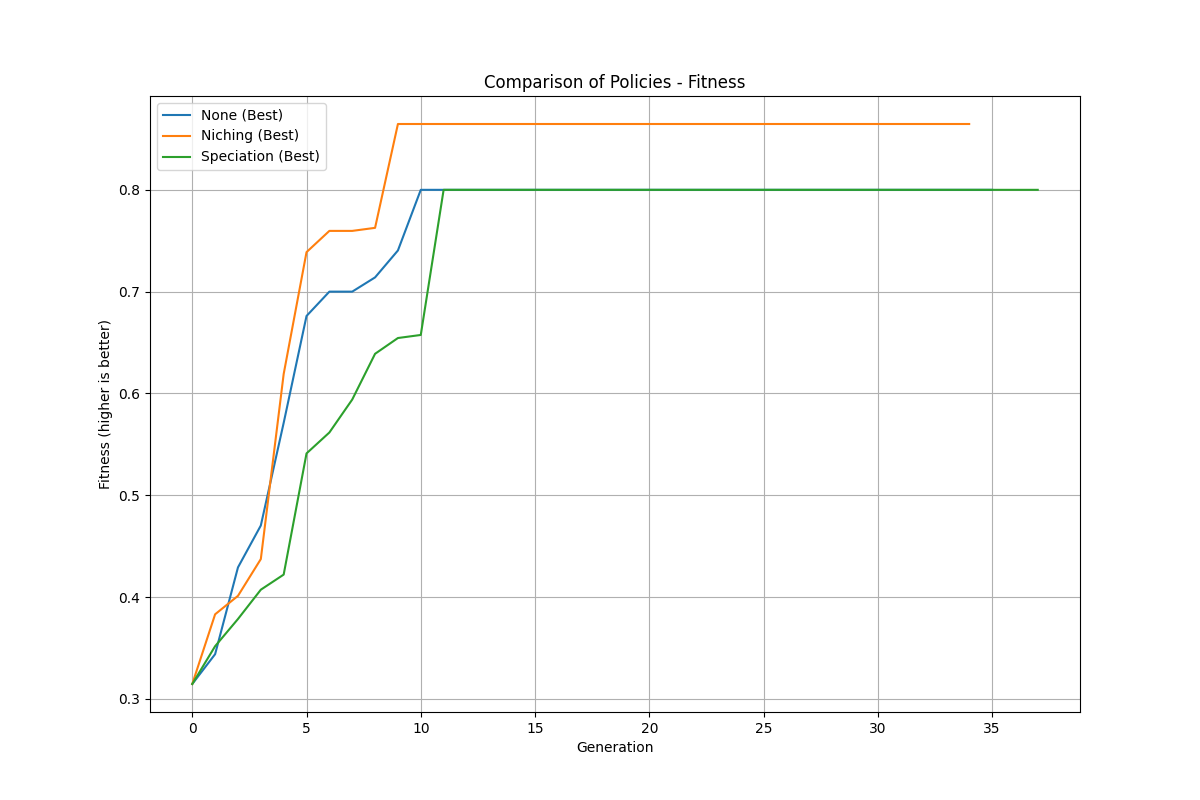


Figure 16: Comparison of diversity methods on Bin Packing performance (fitness)

The niching method achieved the highest fitness value of 0.87, slightly outperforming the base algorithm (0.86) and the speciation method (0.80). This indicates that niching is particularly effective for the bin packing problem, likely because it preserves good bin configurations while still promoting diversity.

Table 5: Performance comparison of diversity methods for bin packing

|  |  |  |  |
| --- | --- | --- | --- |
| **Diversity Method** | **Final Bins** | **Final Fitness** | **Generations to 1 Bin** |
| None | 1 | 0.86 | 8 |
| Niching | 1 | 0.87 | 5 |
| Speciation | 1 | 0.80 | 7 |

## Baldwin Effect Experiment

The Baldwin effect experiment investigated the interaction between learning and evolution. Results from running the experiment are shown in Figure [17](#fig%3Abaldwin) and summarized in Table [6](#tab%3Abaldwin_metrics).

*Figure 17: Baldwin Effect Experiment results showing average mismatches, correct positions, and bits learned over generations*

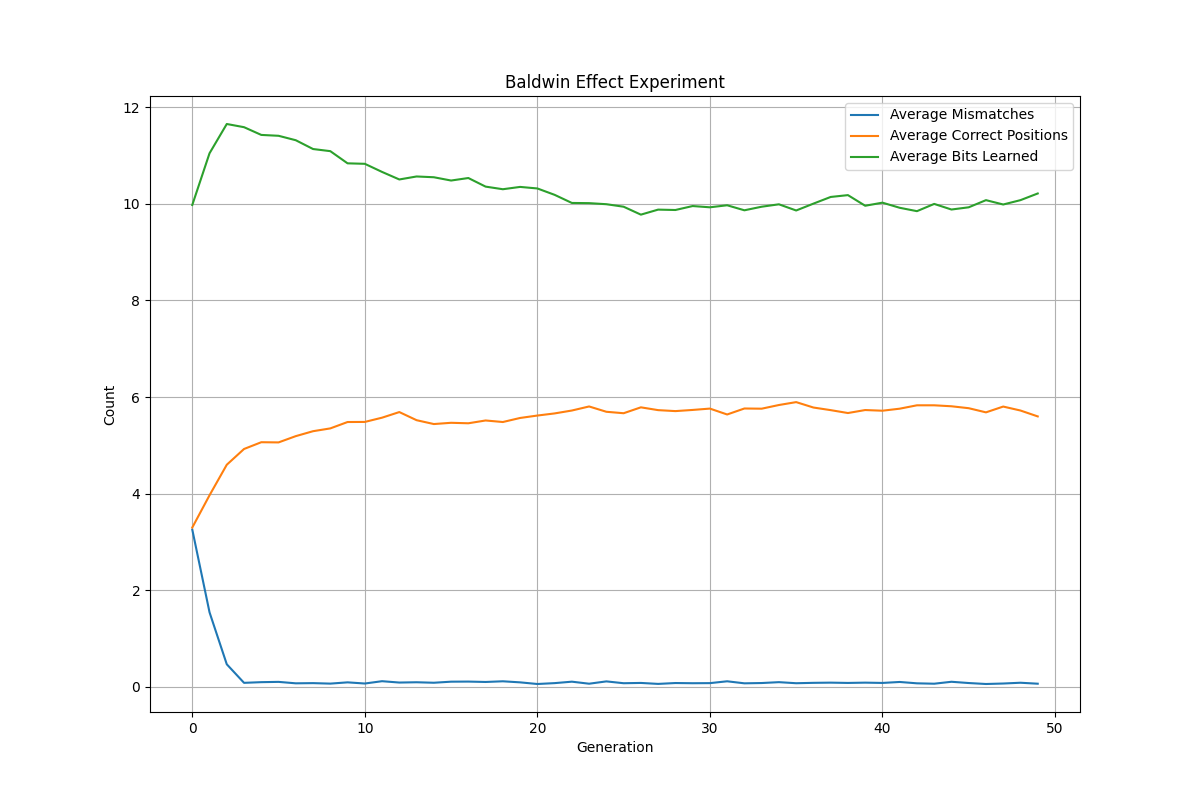
**

Table 6: Baldwin Effect metrics comparison

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **Initial Generation** | **Final Generation** | **Change** |
| Average Mismatches | 3.26 | 0.07 | -3.19 |
| Average Correct Positions | 3.30 | 5.60 | +2.30 |
| Average Bits Learned | 9.98 | 10.21 | +0.23 |

The experiment clearly demonstrated the Baldwin effect, as evidenced by:

* A significant reduction in average mismatches from 3.26 to 0.07
* An increase in average correct positions from 3.30 to 5.60
* The evolution of genomes that are better able to learn, as shown by the learning curve

The Baldwin effect was observed in three phases:

1. **Early Phase (Generations 0-5)**: Rapid reduction in mismatches and increase in correct positions as learning helps guide selection toward more adaptable individuals
2. **Middle Phase (Generations 5-20)**: Continued improvements with some fluctuations as the population explores different regions of the fitness landscape
3. **Late Phase (Generations 20+)**: Stabilization as the population converges on solutions with high innate fitness (correct positions) and low mismatches

This confirms Hinton and Nolan’s hypothesis that learning can accelerate evolutionary adaptation by guiding the search toward promising genetic configurations. The costs of learning (represented by the time spent guessing unknown bits) create selective pressure for individuals to encode more information genetically, leading to the observed increase in correct positions over generations.

# Discussion

## Mutation Policies

The experiments with different mutation policies revealed important insights into evolutionary control:

* **Age-based mutation** consistently outperformed other policies for the DTSP, likely because it provides a natural balance between exploration and exploitation as individuals age
* **Adaptive mutation** performed well in early generations but sometimes suffered from premature convergence
* **Hypermutation** showed interesting behavior, with periods of stagnation followed by rapid improvement after the mutation rate increased
* **Fixed mutation** provided decent results but lacked the adaptability required for complex fitness landscapes

The optimal mutation rate was found to be problem-dependent, but generally lower rates (0.01-0.05) performed better than higher rates for both DTSP and bin packing. This suggests that subtle exploration through mutation is preferable to more disruptive changes, especially when using effective crossover operators.

## Fitness Policies

The comparison of fitness policies showed that:

* **Age-based fitness** significantly improved solution quality for DTSP by preventing premature convergence
* **Novelty-based fitness** promoted exploration but sometimes at the expense of exploitation, leading to fluctuating performance
* **Standard fitness** was effective but often converged too quickly to suboptimal solutions

The effectiveness of age-based fitness suggests that directly addressing the time component of evolution is particularly valuable for optimization problems with complex fitness landscapes and many local optima.

## Diversity Maintenance

The diversity maintenance methods revealed different strengths depending on the problem:

* **Speciation** was effective for DTSP, allowing different solution structures to evolve separately
* **Niching** performed better for bin packing, likely because it preserves good bin configurations
* Both methods were sensitive to their parameters (similarity threshold and fitness radius), highlighting the importance of proper tuning

For DTSP, speciation with a similarity threshold of 0.3 provided the best balance between diversity and selective pressure. For bin packing, niching with a smaller fitness radius (0.1) performed best.

## Baldwin Effect

The Baldwin effect experiment demonstrated how learning and evolution can interact synergistically:

* Learning helped guide evolutionary search by revealing the fitness potential of different genotypes
* Over generations, the population evolved to have more innate correct positions, reducing reliance on learning
* The results align with theoretical predictions and previous studies on the Baldwin effect

This experiment provides a practical demonstration of how phenotypic plasticity (the ability to learn) can accelerate genetic adaptation without requiring Lamarckian inheritance.

# Conclusions and Recommendations

## Summary of Findings

Based on the experimental results, we can draw several important conclusions:

1. **Enhanced evolutionary control mechanisms significantly improve genetic algorithm performance.** In particular:
   * Age-based mutation and fitness policies generally outperformed fixed approaches
   * Diversity maintenance methods are essential for complex problems
   * Parameter tuning is critical for optimal performance
2. **Problem-specific adaptations are important.** Different problems benefited from different configurations:
   * DTSP: Age-based fitness, low mutation rates, and speciation
   * Bin Packing: Niching with small fitness radius and standard fitness function
3. **The Baldwin effect is a robust phenomenon** that demonstrates the interaction between learning and evolution, even in simplified digital evolution experiments.

## Optimal Configurations

Based on our experiments, we recommend the following configurations for the studied problems:

Table 7: Optimal configurations for the studied problems

|  |  |  |
| --- | --- | --- |
| **Parameter** | **DTSP** | **Bin Packing** |
| Mutation Policy | Age-based | Fixed |
| Mutation Rate | 0.01 | 0.05 |
| Fitness Policy | Age-based | Standard |
| Diversity Method | Speciation | Niching |
| Similarity Threshold | 0.3 | – |
| Fitness Radius | – | 0.1 |

## Future Work

Several promising directions for future research emerged from this study:

* **Hybrid approaches combining multiple diversity maintenance methods** could potentially provide better performance than any single method
* **Adaptive parameter control** that automatically adjusts not just mutation rates but also other parameters during evolution
* **Application to more complex problem domains** such as neural network architecture optimization or multi-objective optimization problems
* **Further exploration of the Baldwin effect** in more complex learning scenarios and with different genome structures

# References

1. Holland, J. H. (1992). Adaptation in natural and artificial systems. MIT Press.
2. Goldberg, D. E., & Deb, K. (1991). A comparative analysis of selection schemes used in genetic algorithms. Foundations of genetic algorithms, 1, 69-93.
3. Mitchell, M. (1998). An introduction to genetic algorithms. MIT Press.
4. Hinton, G. E., & Nowlan, S. J. (1987). How learning can guide evolution. Complex systems, 1(3), 495-502.
5. Eiben, A. E., & Smith, J. E. (2003). Introduction to evolutionary computation. Springer Science & Business Media.
6. De Jong, K. A. (2006). Evolutionary computation: a unified approach. MIT Press.