Genetic Algorithm for Solving Magic Squares

Course: Computational Biology (80-512 / 89-512) - 2025 **Exercise 2:** Genetic Algorithms

Submitter:

- **Name:** Elad Cohen
- **ID:** 207252461
- **Department:** Computer Science

Declaration: אני מודעים לדרישת הנוכחות בקורס כפי שפורטו במכתבים ובשיעור הראשון ולכך שמי שלא עומד בדרישה זו לא יוכל לעבור את הקורס.

Code Operating Instructions:

- The Python code consists of two main files: magic_square_genetic.py (containing the GA logic) and magic_square_menu.py (providing a user interface to run experiments).
- **Dependencies:** The code requires ***numpy*** for numerical operations and ***matplotlib*** for plotting graphs. These are standard Python libraries.
- **To Run:**
    i. Ensure Python 3 is installed along with numpy and matplotlib. And I not install run those commands: `pip install numpy matplotlib`
    ii. Place both magic_square_genetic.py and magic_square_menu.py in the same directory.
    iii. Open a terminal or command prompt in that directory.
    iv. Execute the menu script using the command: `python magic_square_menu.py`
    v. Follow the on-screen menu to:
        - Run a single algorithm with specific parameters.
        - Compare all three algorithms (Classical, Darwinian, and Lamarckian) for a given square size.
        - Run a quick demo for a 3x3 square.
- **Code Location:**
    - The code can be accessed at the following cloud service link: https://github.com/elad425/bio_ex2.git

1. Introduction

A Magic Square is an N x N matrix containing a distinct set of integers, typically from 1 to N^2, arranged such that the sum of the numbers in each row, each column, and both main diagonals is the same. This constant sum is known as the "magic sum" or "magic constant." For a square of size N, the magic sum is calculated as M=N (N2+1)/2.

This project focuses on developing a Genetic Algorithm (GA) to find valid magic squares. The GA aims to evolve an initial population of random N x N matrices (permutations of numbers from 1 to N^2) into a valid magic square. The study also investigates the impact of different evolutionary strategies: a classical GA, a Darwinian GA (where fitness is evaluated after local optimization, but unoptimized individuals are used for breeding), and a Lamarckian GA (where optimized individuals and their acquired traits are passed to the next generation). The performance of these algorithms is compared for various square sizes (N=3, N=4, N=5, N=6, N=7), including the more constrained "Most Perfect Magic Square" for N=4.

2. Genetic Algorithm Implementation

The genetic algorithm was implemented in Python using the numpy library for efficient matrix operations. The core components of the GA are detailed below.

2.1. Solution Representation (Individuals)

Each individual in the population represents a candidate magic square.

- **Structure:** An individual is represented as an N x N Numpy array.
- **Content:** The array contains a permutation of integers from 1 to N^2, ensuring that each number appears exactly once, a fundamental requirement for a magic square.
- **Initialization:** The initial population is generated by creating population_size individuals, each filled with a random permutation of the numbers from 1 to N^2.

2.2. Fitness Function

The fitness function evaluates how close a candidate square is to being a valid magic square. A lower fitness score indicates a better solution, with a fitness of 0 signifying a perfect magic square.

- **Calculation:** The fitness is the sum of the absolute differences between the actual sums and the target magic_sum for:
    i. All N row sums.
    ii. All N column sums.
    iii. The two main diagonals (for standard magic squares).
- **Most-Perfect Magic Squares (MPMS):** For MPMS (applicable when N is a multiple of 4), the fitness function includes additional constraints:

- **All Broken Diagonals:** The sum of all N broken diagonals (pandiagonals) in both directions must equal the magic_sum.
- **2x2 Sub-squares:** The sum of numbers in every 2x2 sub-square (with wrap-around) must equal 2(N2+1).
- **Complementary Pairs:** For any two numbers a and b that are N/2 cells apart along a diagonal (i.e., square [i, j] and square [I + N/2, j + N/2] with indices taken modulo N), their sum a+b must equal N2+1.

- The number of calls to this evaluation function is tracked (evaluation_calls) to measure computational effort.

2.3. Crossover Operation

Crossover combines genetic material from two parent individuals to create offspring.

- **Method:** Order Crossover is used.
  i. Two parent squares are flattened into 1D arrays.
  ii. A random contiguous segment is selected from the first parent (Parent 1) and copied directly to the child.
  iii. The remaining positions in the child are filled with numbers from the second parent (Parent 2), in the order they appear in Parent 2, provided they are not already present in the child (from Parent 1's segment).
- **Rate:** The crossover_rate parameter (default 0.8) determines the probability that two selected parents will undergo crossover. If no crossover occurs, the first parent is typically cloned.

2.4. Mutation Operation

Mutation introduces small, random changes into an individual's genome, helping to maintain diversity and explore new areas of the search space.

- **Method:** Swap mutation is implemented.
  i. A copy of the individual's square is made.
  ii. A number of swaps, determined by max(1, int(N^2 * mutation_rate)), are performed.
  iii. For each swap, two distinct random positions (cells) in the square are chosen, and the numbers in these cells are exchanged.
- **Rate:** The mutation_rate parameter (default 0.1) influences the number of swaps.

2.5. Handling Premature Convergence

Premature convergence occurs when the population loses diversity and converges to a suboptimal solution too early. While no single, explicit mechanism is labeled "premature convergence handling," the following aspects contribute to mitigating it:

- **Tournament Selection:** By selecting parents based on tournaments rather than purely on fitness proportion, there's a chance for less fit individuals to be selected, helping maintain diversity.
- **Mutation:** The mutation operator, as described above, continuously introduces new genetic material.
- **Stagnation Detection:** The algorithm includes a stagnation_limit. If the best fitness score in the population does not improve for a specified number of generations, the evolution process is halted. This acts more as a stopping criterion when progress stalls, which can be a symptom of premature convergence.
- The problem did not require advanced techniques like adaptive mutation rates or niching, but the combination of tournament selection and mutation provides a basic level of diversity maintenance.

2.6. Selection of the Next Generation

The selection process determines which individuals from the current generation will contribute to creating the next generation.

- **Elitism:** A fixed number of the best-performing individuals (elite_size, default 10) from the current generation are directly copied to the next generation, ensuring that the best solutions found so far are preserved.
- **Parent Selection:** For the remaining spots in the new population, parents are selected using **Tournament Selection**.
    i. A small group of individuals (tournament_size, default 5) is randomly chosen from the current breeding population.
    ii. The individual with the best fitness (lowest score) within this tournament group is selected as a parent.
    iii. This process is repeated to select two parents for crossover.
- The offspring generated from crossover and mutation fill the rest of the next generation's population.

2.7. Stopping Criteria

The GA run terminates when one of the following conditions is met:

1. **Solution Found:** The best fitness in the population reaches the target_fitness (which is 0 for a perfect magic square).
2. **Maximum Generations:** The algorithm completes a predefined max_generations (default 1000).
3. **Stagnation:** The best fitness score in the population does not improve for stagnation_limit consecutive generations (default is dynamically set based on max_generations or N*10).

2.8. Lamarckian and Darwinian Evolution

To study the effects of acquired traits, Darwinian and Lamarckian evolution strategies were implemented alongside the classical GA.

- **Local Optimization:** A local optimization function attempts to improve an individual square. It iteratively tries a number of random swaps and applies the swap that yields the greatest fitness improvement. For standard magic squares, an incremental fitness calculation is used during optimization to speed up the process.
- **Classical GA:** No local optimization is applied. Evolution proceeds based on the original fitness of individuals.
- **Darwinian GA:**
    i. Each individual undergoes local optimization.
    ii. Its fitness is evaluated after this optimization.
    iii. However, for breeding (selection, crossover, mutation), the *original, unoptimized* version of the individual is used. Acquired traits are not directly inherited.
- **Lamarckian GA:**
    i. Each individual undergoes local optimization.
    ii. Its fitness is evaluated after this optimization.
    iii. The *optimized* version of the individual (with its acquired traits) is used for breeding and passed to the next generation.

3. Experiments and Results

Experiments were conducted to compare the performance of the Classical, Darwinian, and Lamarckian genetic algorithms across different magic square sizes (N) and types (standard vs. Most Perfect). The default parameters used were: population size = 100, max generations = 1000, mutation rate = 0.1, elite size = 10, tournament size = 5, crossover rate = 0.8, and optimization steps = N.

The primary metrics for comparison were:

- **Solution Found:** Whether a valid magic square (Fitness = 0) was found.
- **Generations:** The number of generations taken to find a solution.
- **Evaluation Calls:** The total number of calls to the fitness function (a proxy for computational effort).
- **Runtime (s):** The wall-clock time taken for the algorithm to complete.
- **Best Fitness:** The fitness of the best individual at the end of the run.

3.1. Summary of Results

The following table summarizes the key findings from the results.txt file:

N	Type	Algorithm	Solution Found	Generations	Evaluation Calls	Runtime (s)	Best Fitness
3	Standard	Classical	True	24	2,401	0.18	0
3	Standard	Darwinian	True	1	993	0.28	0
3	Standard	Lamarckian	True	1	989	0.28	0
4	Standard	Classical	True	43	4,301	0.40	0
4	Standard	Darwinian	True	1	1,701	0.79	0
4	Standard	Lamarckian	True	1	1,701	0.80	0
4	Most Perfect	Classical	True	73	7,301	0.80	0
4	Most Perfect	Darwinian	True	1	27,184	1.39	0
4	Most Perfect	Lamarckian	True	2	53,287	2.73	0
5	Standard	Classical	**False**	1000	100,001	10.92	5
5	Standard	Darwinian	True	26	67,601	46.36	0
5	Standard	Lamarckian	True	40	103,992	94.16	0
6	Standard	Classical	**False**	1000	100,001	13.74	24
6	Standard	Darwinian	True	328	1,213,600	1192.55	0
6	Standard	Lamarckian	True	23	85,099	107.38	0
7	Standard	Classical	**False**	1000	100,001	16.17	58
7	Standard	Darwinian	True	165	824,999	1102.59	0
7	Standard	Lamarckian	True	29	144,996	204.93	0

(Note: Runtimes are approximate and can vary based on the system.)

3.2. Analysis and Insights

Performance on Smaller Squares (N=3, N=4 Standard):

- All three algorithms successfully found solutions.
- Darwinian and Lamarckian algorithms consistently found solutions in significantly fewer generations (often in the first generation after initial optimization) compared to the Classical GA.
- While Darwinian and Lamarckian had slightly higher runtimes per generation due to the local optimization step, their drastically reduced generation count often led to comparable or even lower total evaluation calls for these smaller, easier problems.

Performance on Most Perfect Magic Square (N=4):

- This problem is notably harder due to additional constraints.
- The Classical GA still found a solution but took more generations (73) than for the standard 4x4 square.
- Darwinian GA found a solution in 1 generation, but the number of evaluation calls was very high (27,184), reflecting the intensive local optimization process needed to satisfy the complex MPMS fitness criteria.
- Lamarckian GA took 2 generations and an even higher number of evaluation calls (53,287). This suggests that while local optimization helps, inheriting these highly specific optimized traits might sometimes lead the search into regions that require more effort to escape or refine further for MPMS. The optimization itself is more costly due to the complex fitness function for MPMS (no incremental evaluation was used for MPMS optimization in the code).

Performance on Larger Squares (N=5, N=6, N=7 Standard):

- **Classical GA:** Failed to find solutions for N=5, N=6, and N=7 within the 1000-generation limit, terminating with non-zero fitness values. This indicates its inefficiency for more complex search spaces.
- **Darwinian vs. Lamarckian:**
    - For **N=5**, Darwinian GA performed better than Lamarckian, finding a solution in fewer generations (26 vs. 40), with fewer evaluation calls, and in less runtime. This is an interesting case where the Lamarckian approach of inheriting optimized traits was not superior. It's possible that the local optimization for N=5 sometimes led individuals into local optima that were harder for the Lamarckian GA to escape from, whereas the Darwinian approach, by breeding with unoptimized (potentially more diverse or differently structured) individuals, had a slight advantage.
    - For **N=6 and N=7**, Lamarckian GA significantly outperformed Darwinian GA. It found solutions in dramatically fewer generations (N=6: 23 vs. 328; N=7: 29 vs. 165), with substantially fewer evaluation calls, and in much shorter runtimes. This strongly suggests that for these larger and more complex standard magic squares, the ability to inherit acquired traits (Lamarckian) provided a

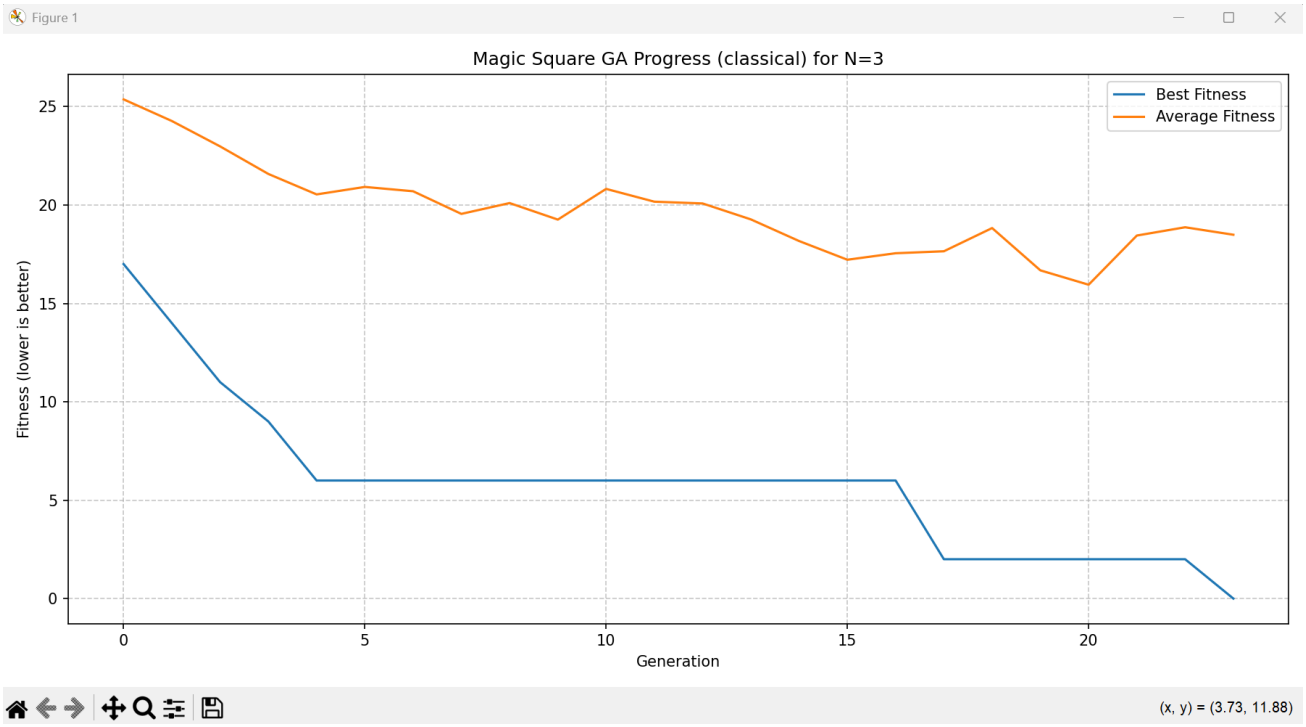
significant advantage in navigating the search space efficiently. The Darwinian approach, while eventually finding solutions, required extensive local search in each generation, leading to very high evaluation counts and runtimes.

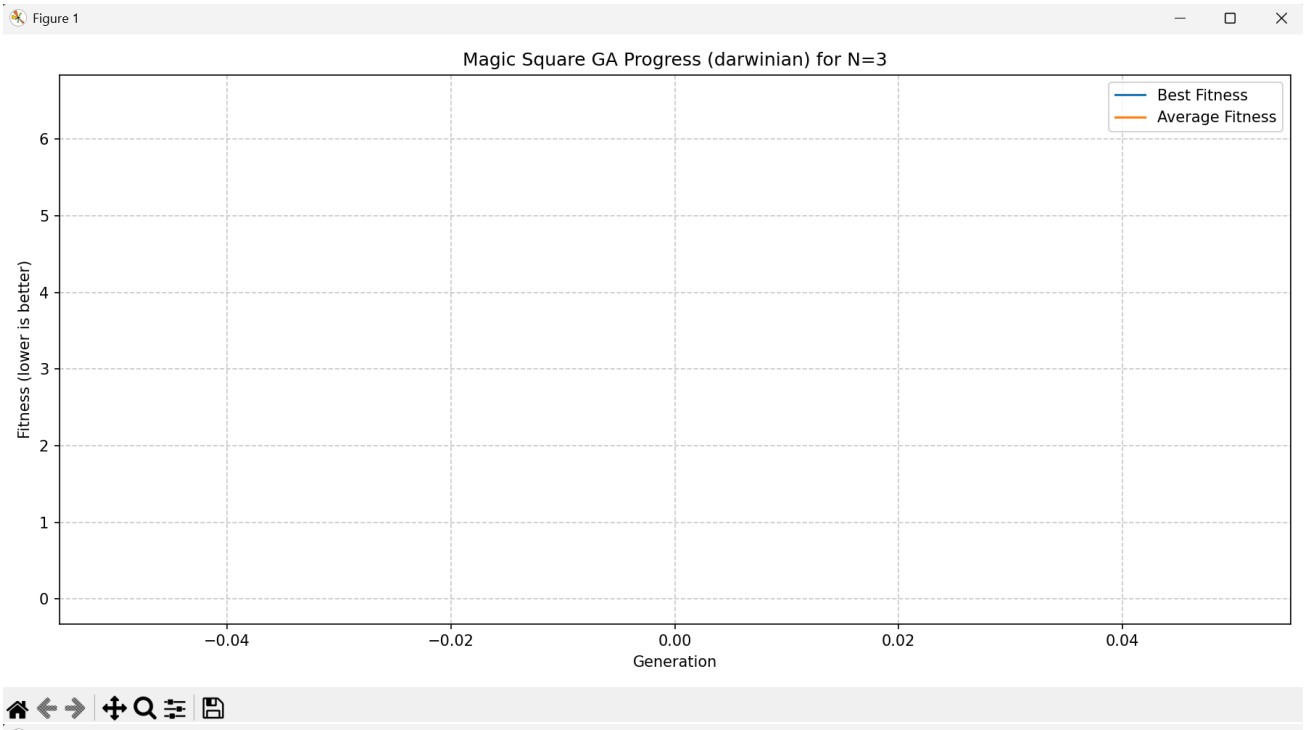
General Observations:

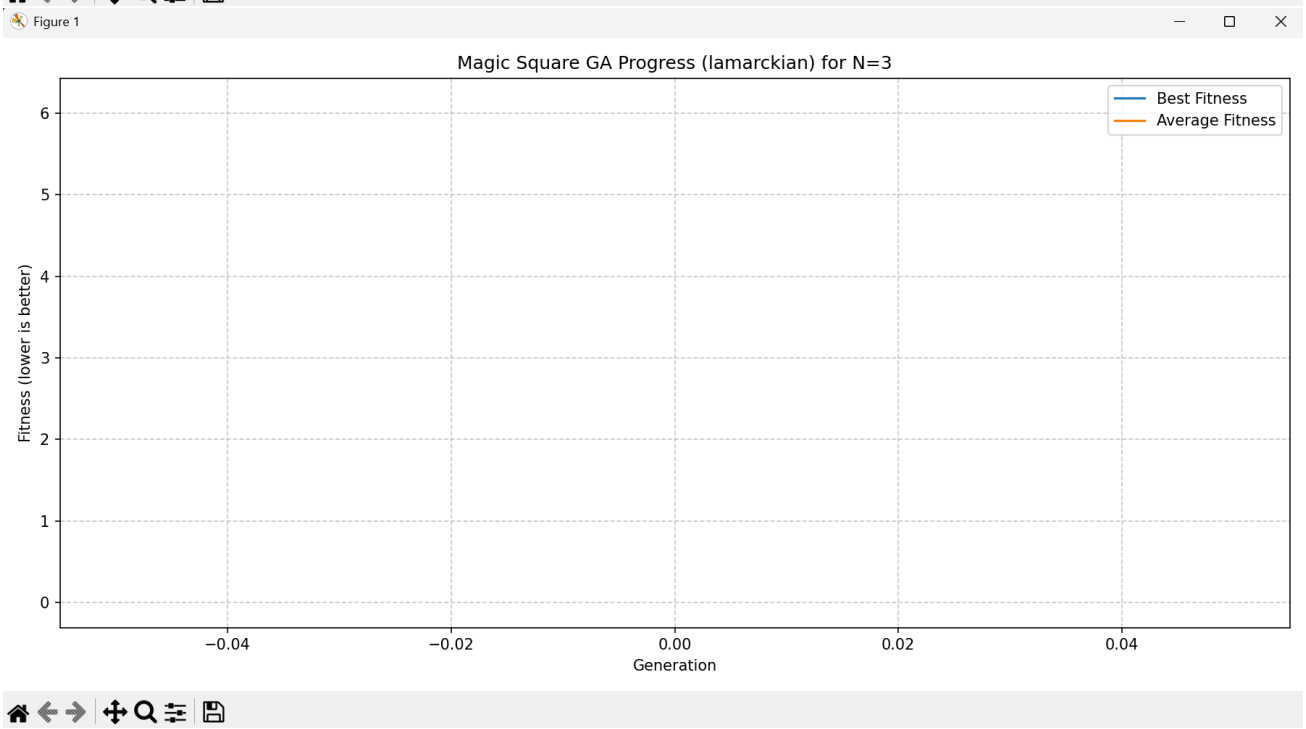
- **Effectiveness of Local Optimization:** Both Darwinian and Lamarckian approaches, which incorporate local optimization, were generally more effective at finding solutions, especially for larger squares where the Classical GA failed.
- **Lamarckian Evolution's Advantage:** For the more challenging standard squares (N=6, N=7), Lamarckian evolution demonstrated a clear benefit. The inheritance of traits improved by local optimization seemed to guide the search more effectively towards solutions.
- **Computational Cost of Optimization:** The local optimization step adds computational overhead per generation. This is evident in the higher number of evaluation calls per generation for Darwinian/Lamarckian types, especially for MPMS where fitness evaluation is more complex and optimization is more intensive. However, if it drastically reduces the number of generations needed, it can still be more efficient overall.
- **Convergence Rate:** Darwinian and Lamarckian algorithms generally exhibited faster convergence towards low fitness values due to the local optimization pushing individuals towards better solutions quickly. The Classical GA showed a more gradual decrease in fitness.

3.3. Graphical Representation

For n=3:

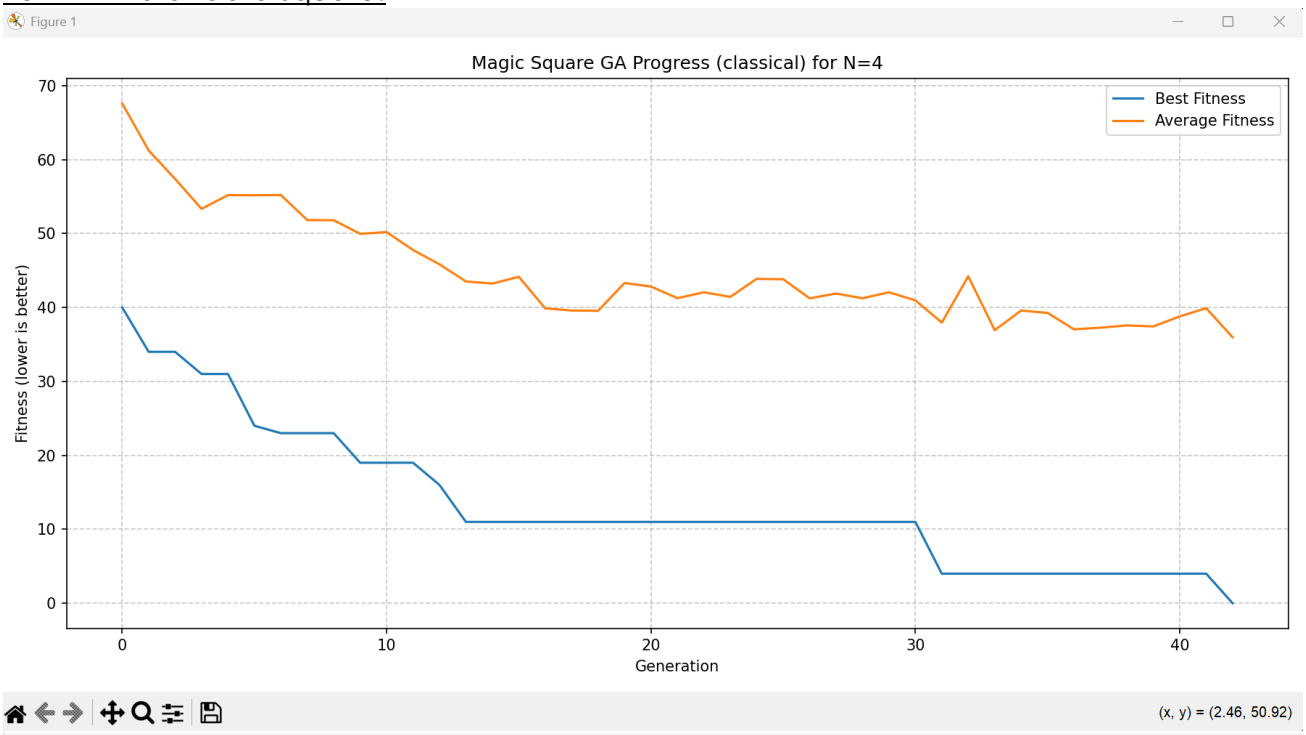


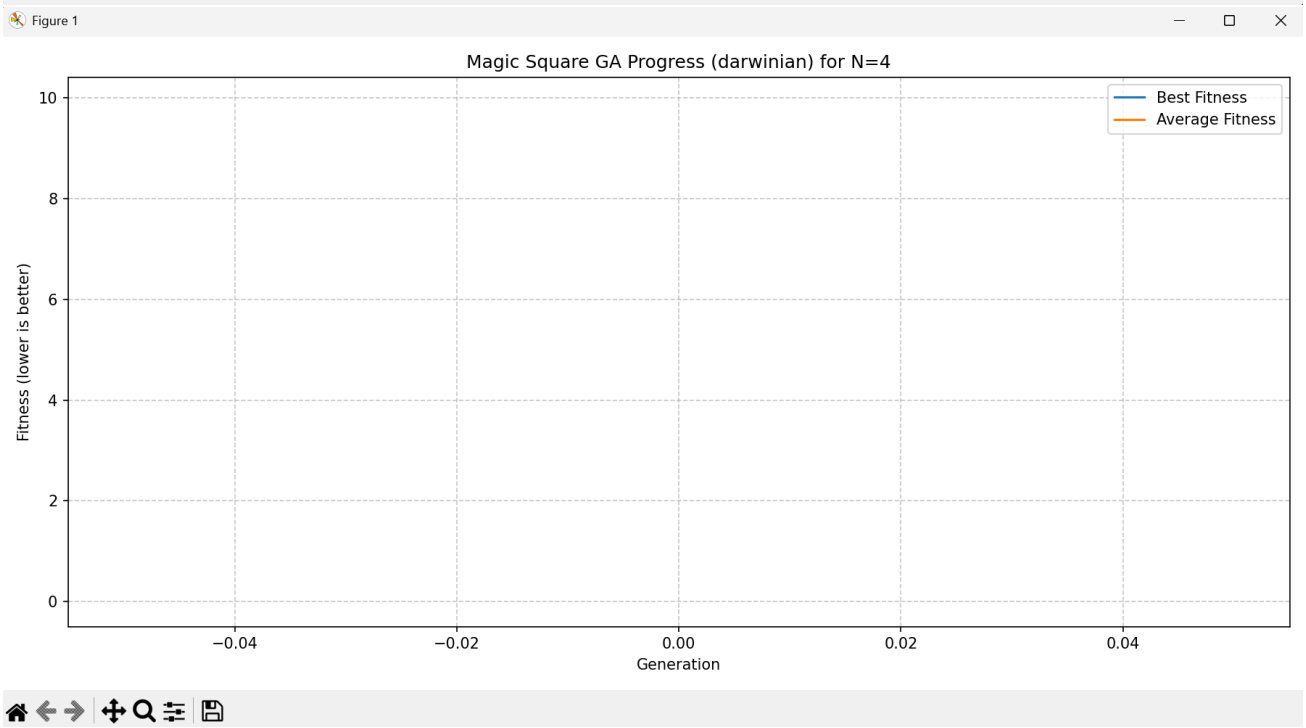


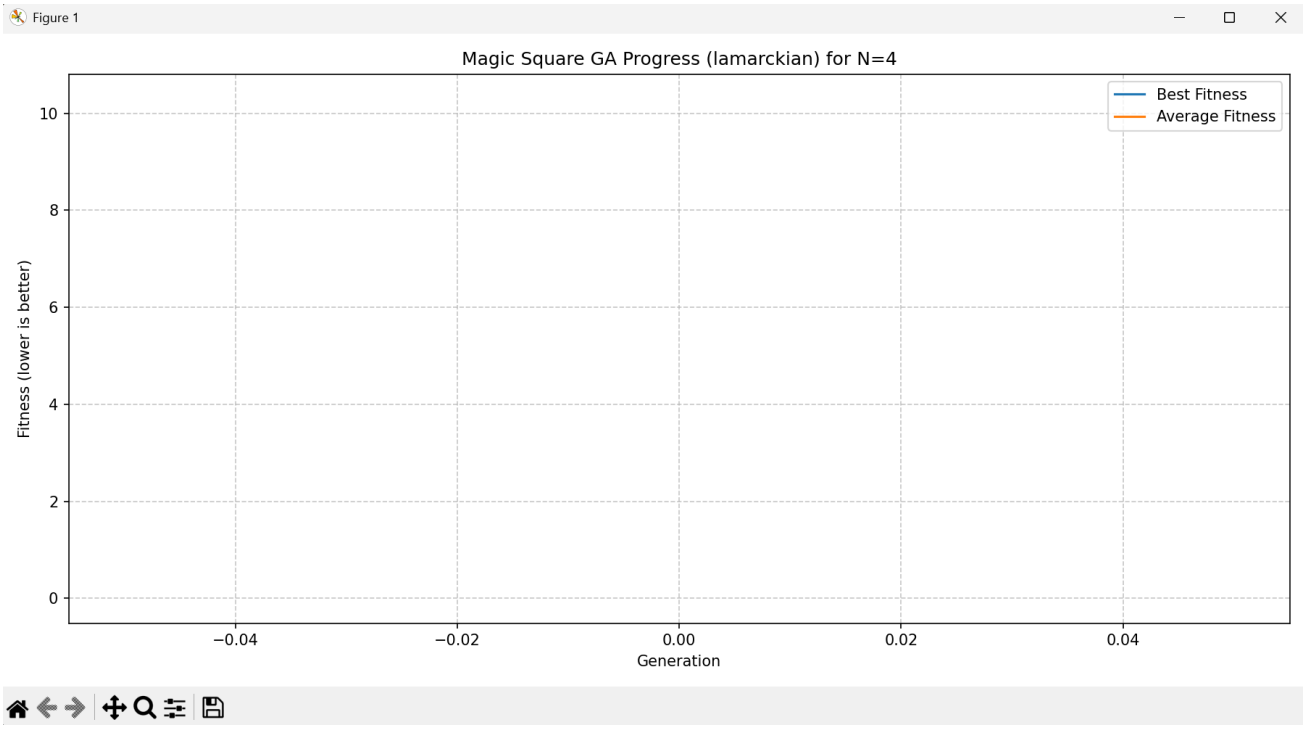


We can see that for the classic GA it took 24 generation to reach results. But for both Darwinian and Lamarckian it took only one generation

For N=4 – Standard square:

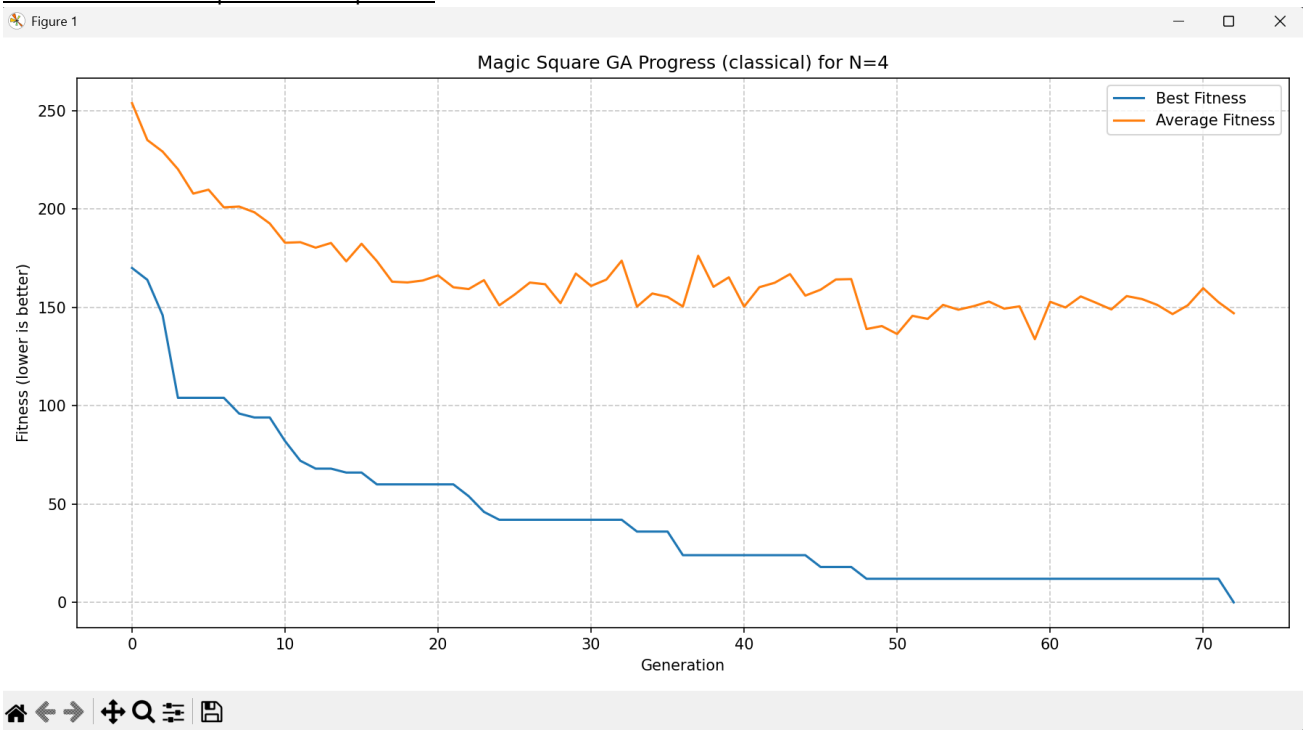


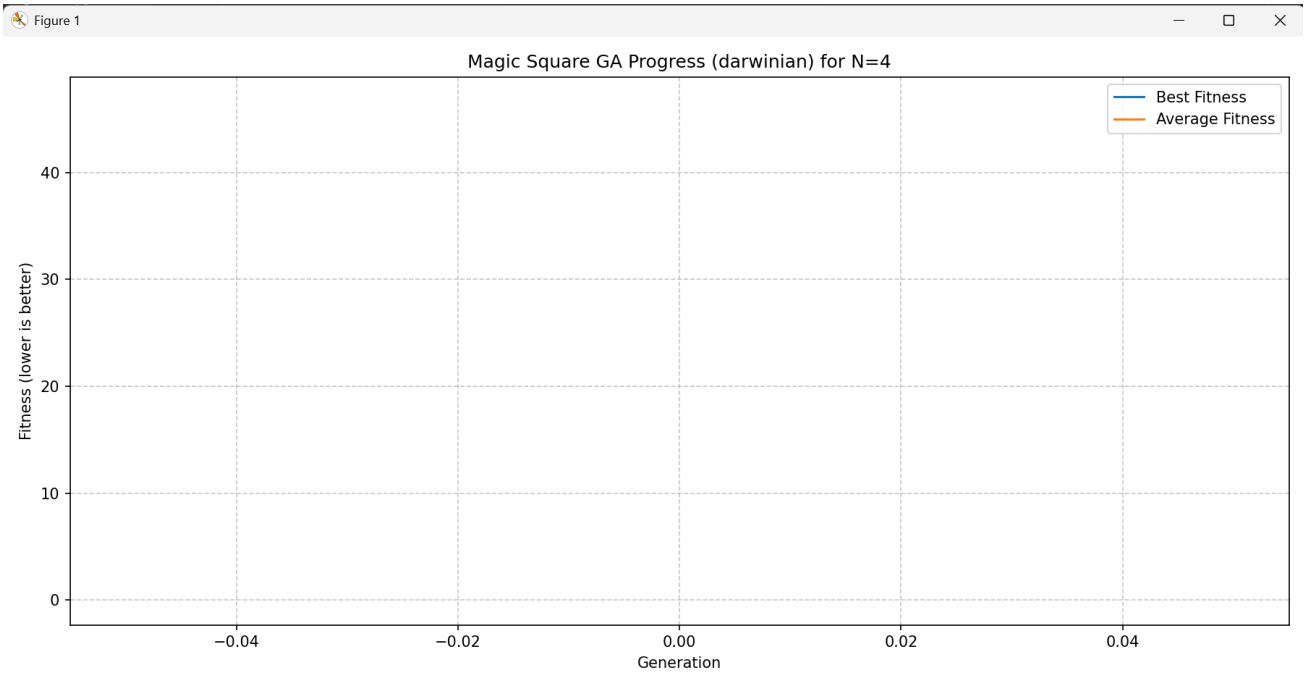


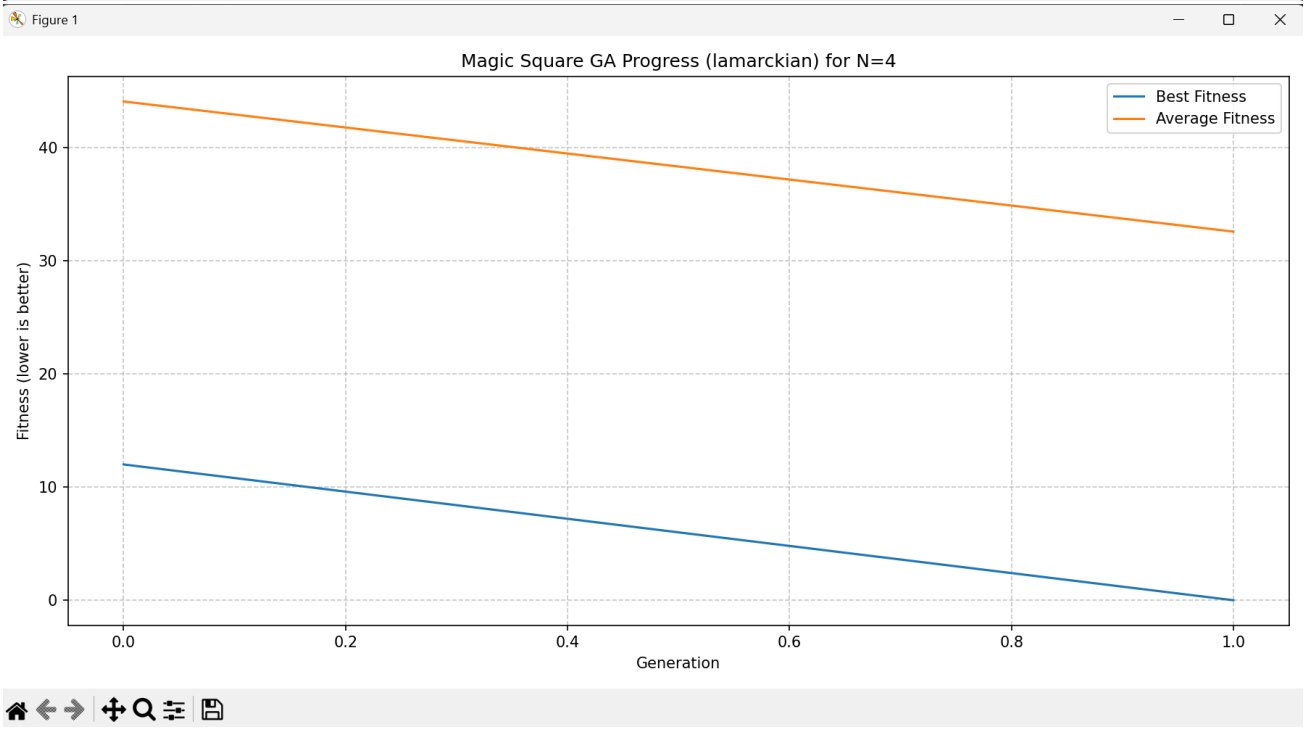


We can see that for the classic GA it took 43 generation to reach results. But for both Darwinian and Lamarckian it took only one generation

For N=4 – most perfect square:







We can see that for the classic GA it took 73 generation to reach results. For the Darwinian it took one generation and for the Lamarckian it took two round

For N=5:

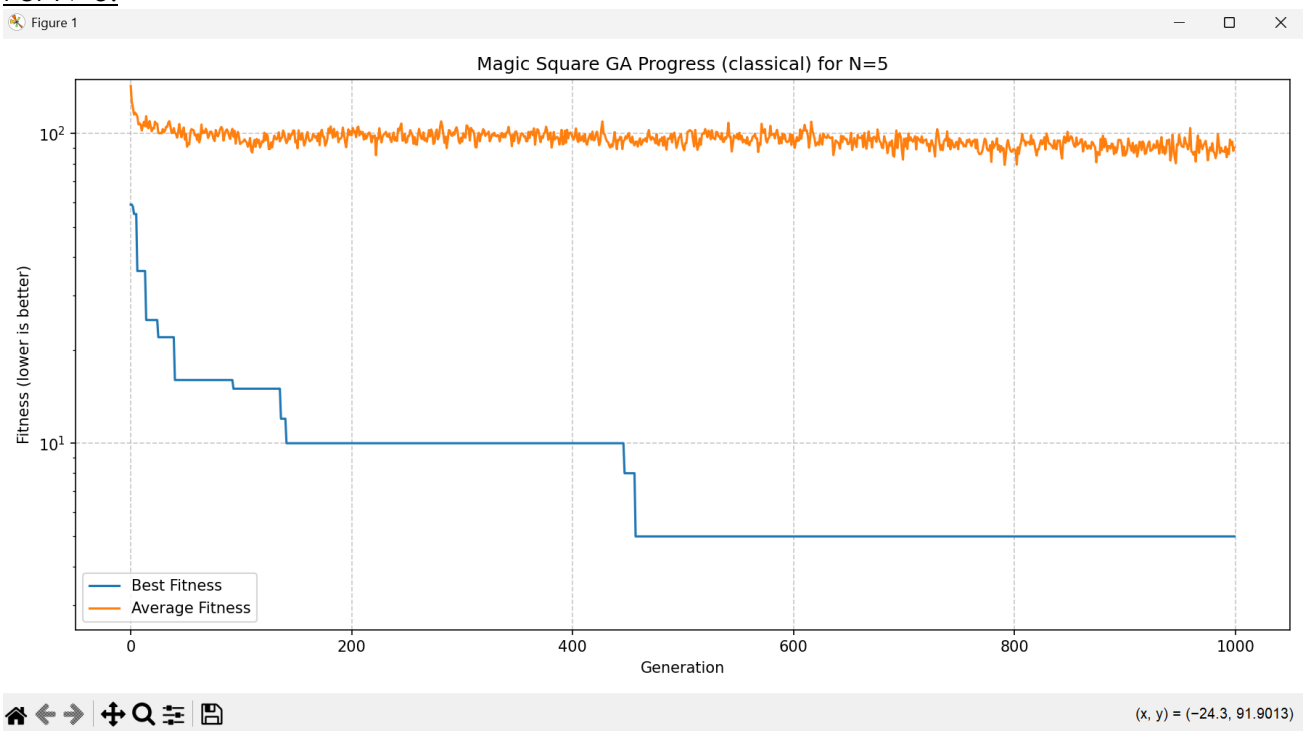
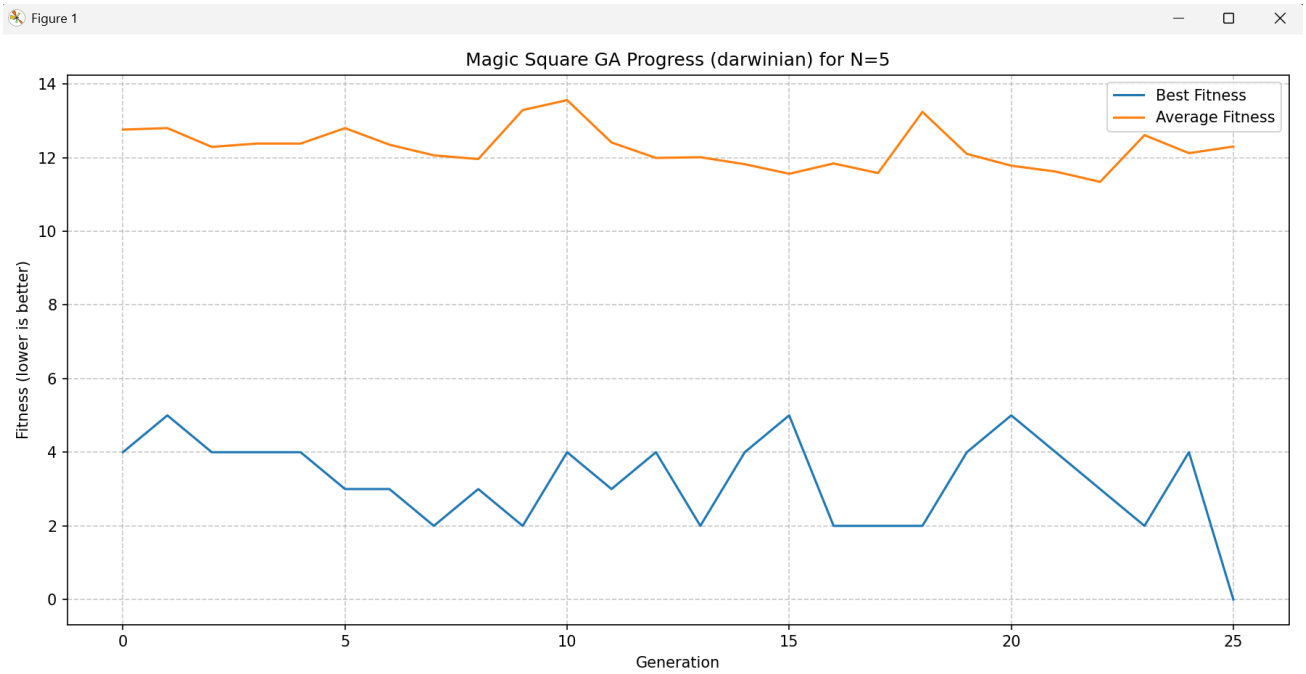
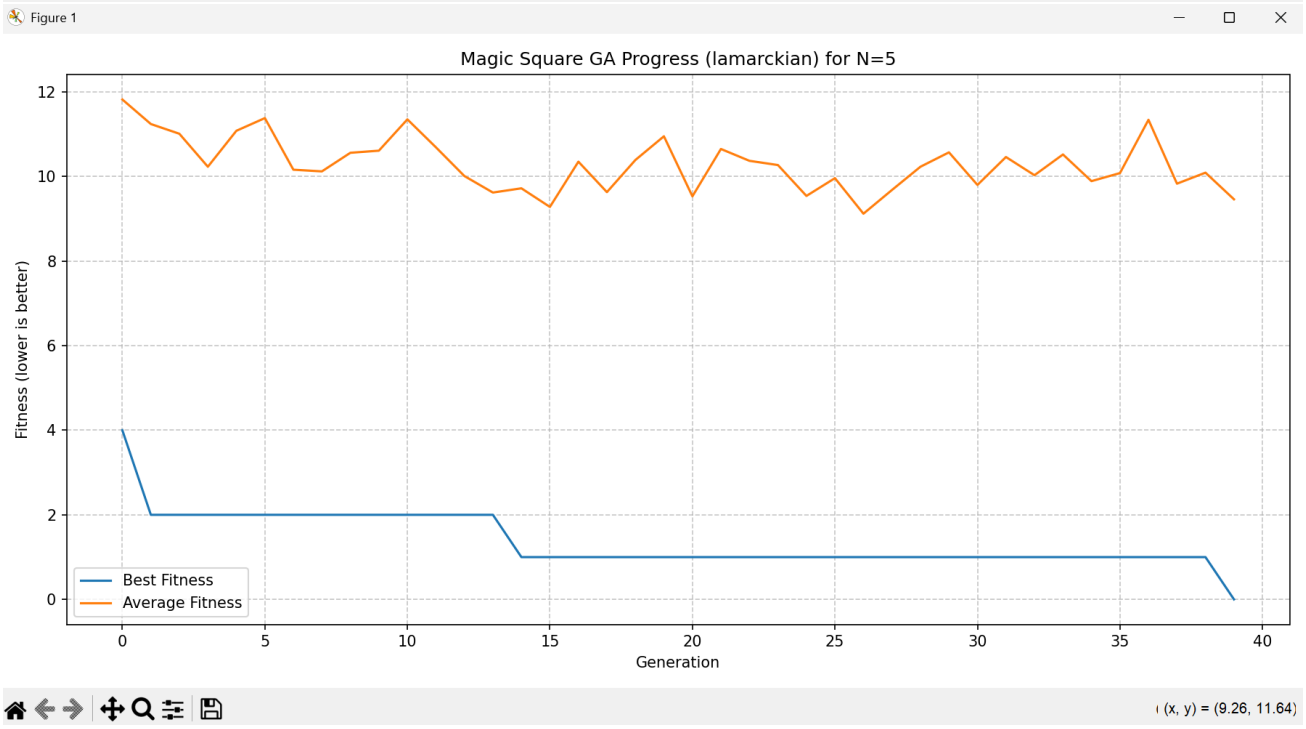


Figure 1
Magic Square GA Progress (classical) for N=5
10^2
10^1
Fitness (lower is better)
Best Fitness
Average Fitness
0
200
400
600
800
1000
Generation
(x, y) = (−24.3, 91.9013)





We can see that the classic GA did not reach results and reach the max generation (1000). But for the Darwinian it took 26 generations and for the Lamarckian it took 39 rounds.

For N=6:

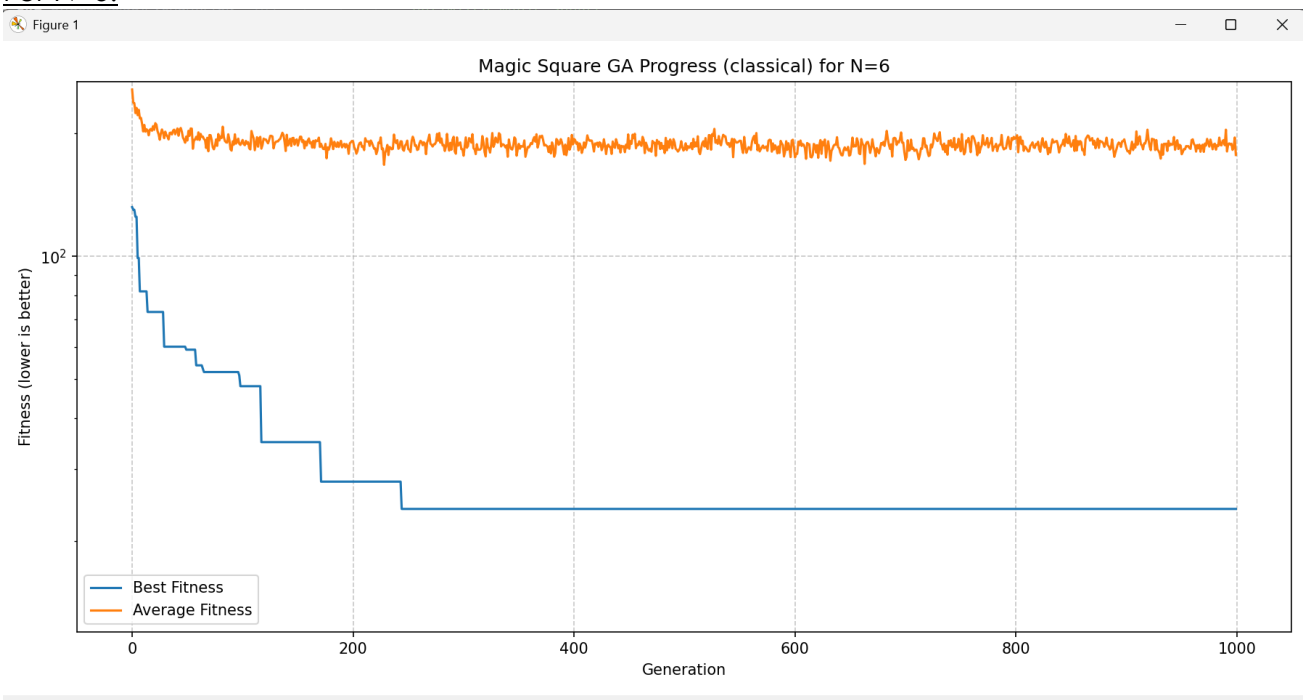


Figure 1
Magic Square GA Progress (classical) for N=6
Fitness (lower is better)
10^2
Best Fitness
Average Fitness
0
200
400
600
800
1000
Generation

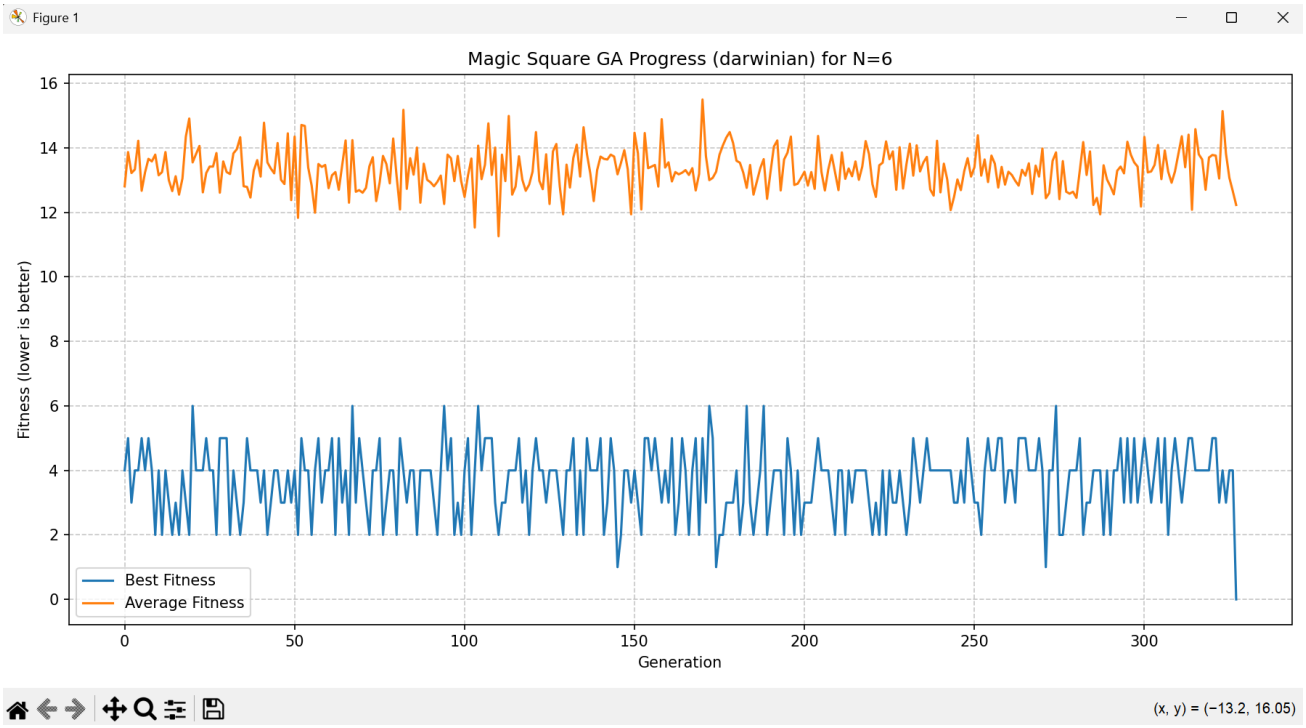
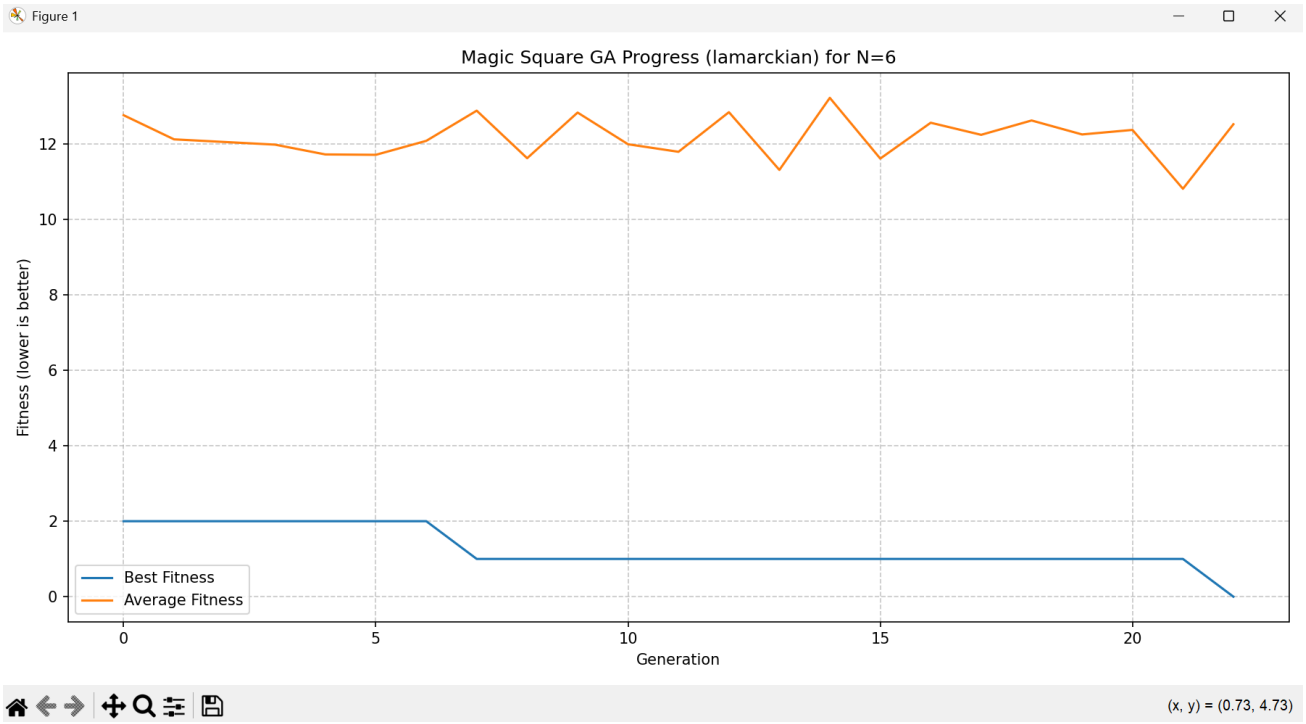
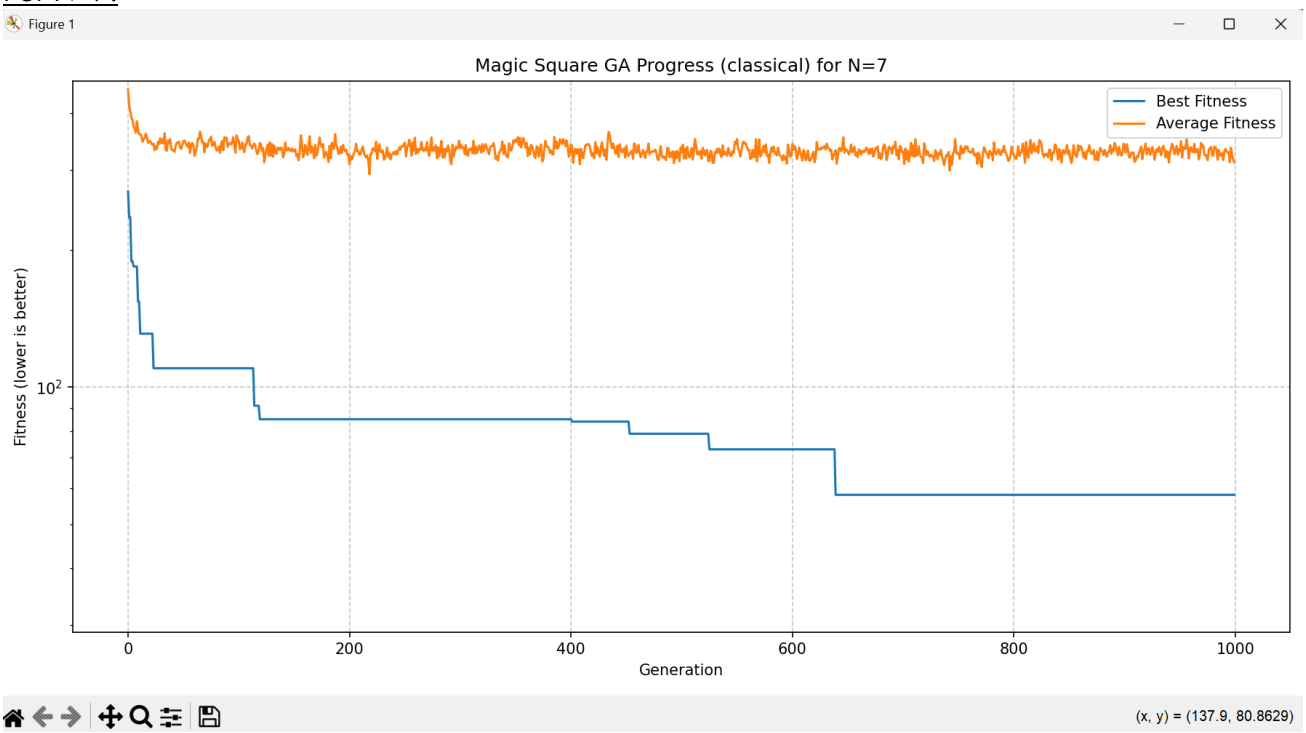


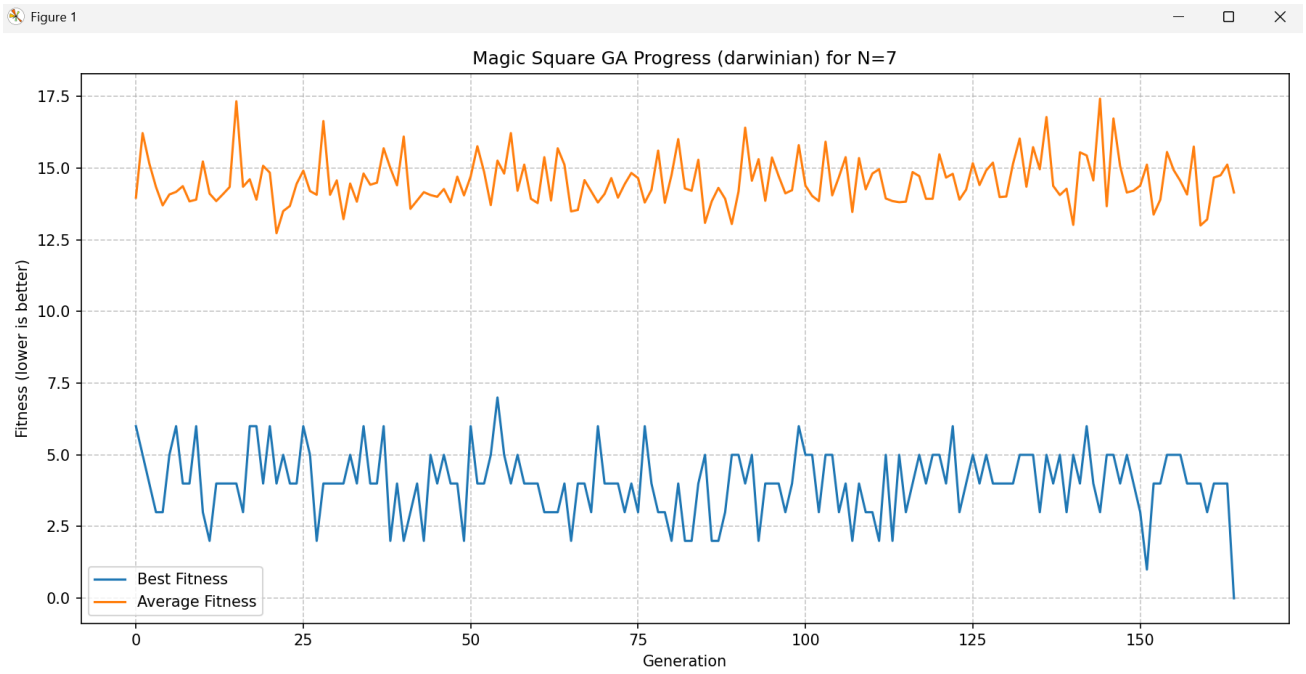
Figure 1
Magic Square GA Progress (darwinian) for N=6
Fitness (lower is better)
16
14
12
10
8
6
4
2
0
Best Fitness
Average Fitness
0
50
100
150
200
250
300
Generation
(x, y) = (−13.2, 16.05)

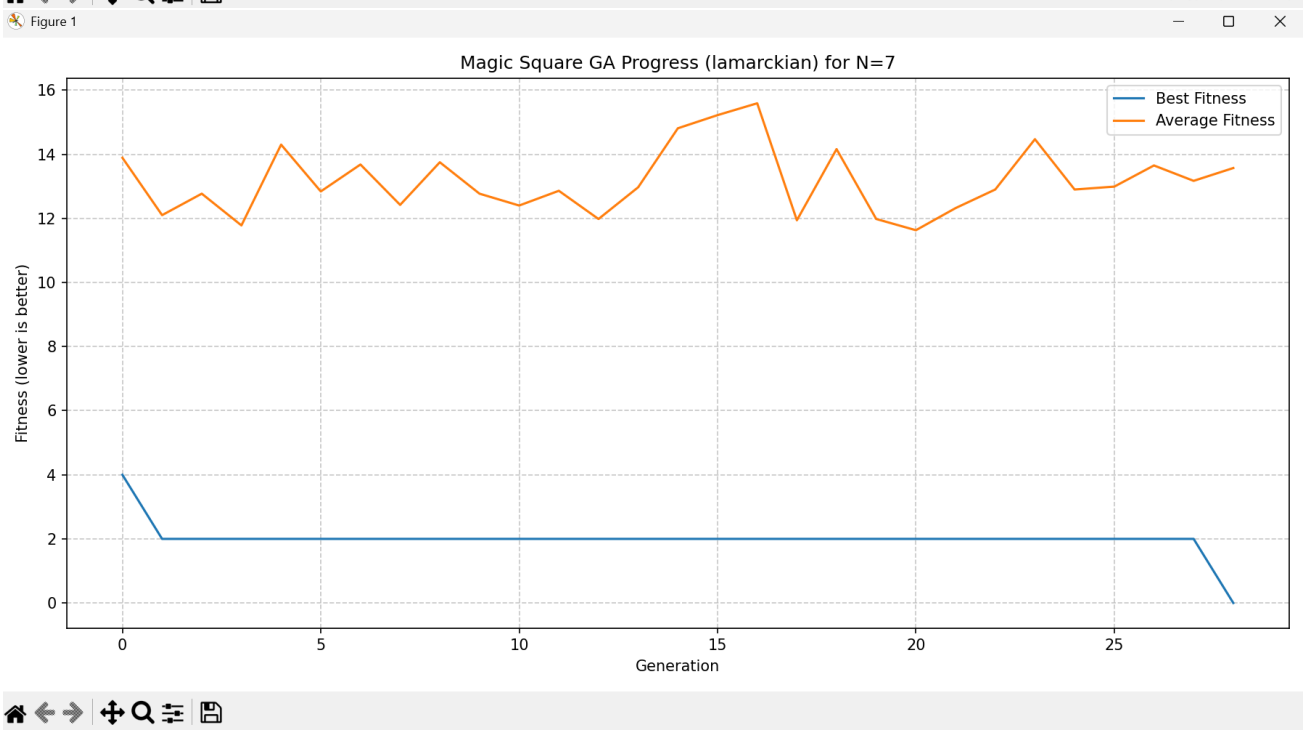


We can see that the classic GA did not reach results and reach the max generation (1000). The Darwinian took some times and the best fitness was unstable but eventually it reached results after 328 generations while for the Lamarckian it took only 23 rounds, which is a big different. In here we can see the power of the lamarckian algorithem.

For N=7:







We can see that the classic GA did not reach results and reach the max generation (1000). The Darwinian took some times and the best fitness was unstable but eventually it reached results after 165 generations while for the Lamarckian it took only 29 rounds, In here we can again see the power of the lamarckian algorithem compare to the other algorithms.

3.4. Example Solutions

Classical Solution for N=3 (Magic Sum 15)

| 2 | 7 | 6 |

| 9 | 5 | 1 |

| 4 | 3 | 8 |

Darwinian Solution for N=4 (Magic Sum 34) - Most Perfect Magic Square

| 9 | 7 | 14 | 4 |

| 6 | 12 | 1 | 15 |

| 3 | 13 | 8 | 10 |

| 16 | 2 | 11 | 5 |

Darwinian Solution for N=5 (Magic Sum 65)

| 3 | 20 | 19 | 6 | 17 |

| 16 | 13 | 2 | 9 | 25 |

| 14 | 4 | 21 | 15 | 11 |

| 24 | 10 | 1 | 23 | 7 |

| 8 | 18 | 22 | 12 | 5 |

Lamarckian Solution for N=7 (Magic Sum 175)

| 25 | 38 | 8 | 28 | 26 | 30 | 20 |

| 37 | 15 | 14 | 29 | 31 | 9 | 40 |

| 41 | 12 | 34 | 27 | 10 | 45 | 6 |

| 42 | 13 | 16 | 49 | 2 | 32 | 21 |

| 1 | 7 | 23 | 17 | 43 | 36 | 48 |

| 11 | 46 | 33 | 22 | 24 | 4 | 35 |

| 18 | 44 | 47 | 3 | 39 | 19 | 5 |

4. Conclusion

The genetic algorithm developed in this exercise successfully solved the magic square problem for various sizes, including the challenging Most Perfect Magic Square. The comparison between Classical, Darwinian, and Lamarckian evolutionary strategies provided valuable insights into their respective strengths and weaknesses.

The Classical GA, while simple to implement, struggled significantly with larger problem instances (N >= 5), failing to find solutions within the given computational limits. This highlights the limitations of pure random search combined with basic genetic operators for complex combinatorial problems.

The introduction of local optimization in the Darwinian and Lamarckian algorithms drastically improved performance. Both methods were able to find solutions for larger squares where the Classical GA failed.

- For smaller, simpler problems (N=3, N=4 standard), both Darwinian and Lamarckian GAs found solutions very quickly, often in the first generation, demonstrating the power of even a limited amount of local search.
- The Most Perfect Magic Square (N=4) proved more challenging. While solutions were found quickly in terms of generations by D/L methods, the number of fitness evaluations was high, indicating the computational cost of the local optimization and the complex fitness landscape.
- A key finding was the varying performance of Darwinian versus Lamarckian GAs for larger standard squares. For N=5, the Darwinian approach was more efficient. This could be attributed to the possibility that Lamarckian inheritance, by strongly guiding the population based on locally optimized traits, might occasionally lead to premature convergence on certain types of fitness landscapes if the local optima found are not on the path to global optima. The Darwinian approach, by separating the traits used for survival (optimized) from those used for reproduction (original), might maintain more diversity in the gene pool, allowing it to escape such traps.
- However, for even larger squares (N=6, N=7), the Lamarckian GA demonstrated superior performance, finding solutions much faster and with fewer evaluations than the Darwinian GA. This suggests that for highly complex problems, the benefit of directly inheriting beneficial acquired traits outweighs the potential risks of premature convergence, leading to a more directed and efficient search.

In conclusion, this exercise effectively demonstrated the principles of genetic algorithms and the significant impact that incorporating local search and different inheritance strategies (like Lamarckian evolution) can have on their efficacy, particularly for complex optimization problems. The choice between Darwinian and Lamarckian approaches may depend on the specific problem characteristics and the nature of the fitness landscape.