Adaptive Evolutionary Algorithm for Solving Sudokus: AEASS

Emre Ertürk, Computer Engineering Department, Izmir Institute of Technology,

Izmir,Turkey, emreerturk@std.iyte.edu.tr

***Abstract*— Sudokus are logic based combinatorial number placement puzzles that became popular at the beginning of the 21st century. A popular site for sudoku enthusiasts sudokuwiki.org contain "unsolvable" sudokus which are sudokus that can't be solved by the site's own sudoku solver. This paper presents AEASS an adaptive EA combining concepts of Traditional EAs, LEAs and MAs to solving these "unsolvable" sudokus as well as other less complex sudokus by combining genetic algorithms with local search techniques. Our proposed method uses specialized genetic operators, and adaptive local search strategies with Lamarckian evolution, where local improvements are inherited by offspring. In addition to these our algorithm has population reset while in theory resetting a population is contradictory to the nature of MAs because they improve a population and population reset replaces this improved population with a new population instead, in practice this approach allows the algorithm to be more time efficient since population reset helps the algorithm to escape local optimas. Key findings are AEASS can solve both unsolvable and other less complex sudokus but getting stuck at a local optima greatly decreases our algorithms performance. When our algorithm doesn't get stuck at local optima it can solve a complex sudoku in mere seconds.**

Keywords- Evolutionary Algorithm(EA), Lamarckian Evolution (LE),Memetic Algorithm(MA)

I. INTRODUCTION

A. Sudoku

Sudoku is a popular logic-based number placement puzzle that has attracted significant research attention as a constraint satisfaction problem. A standard Sudoku grid consists of a 9×9 matrix divided into nine 3×3 sub-grids. The objective is to fill the grid such that each row, column, and 3×3 sub-grid contains the digits 1-9 exactly once while respecting any initially provided values (givens).

From Brittanica [1] "While the puzzle is based entirely on logic without arithmetic, it presents fascinating mathematical properties that have intrigued mathematicians worldwide. The total possible number of valid Sudoku grids, as proven in 2005, is 6,670,903,752,021,072,936,960 (approximately 6.67×10^{21})". However, this represents only is the beginning of its mathematical depth.

From a computational perspective, Sudoku belongs to the class of NP-complete problems, as proven by Yato and Seta in 2003[2]. This classification means that while verifying a solution is straightforward, finding one can be computationally intensive. The difficulty level of a Sudoku puzzle is determined by both the quantity and positions of the original fixed numbers, creating a spectrum of challenges that can range from simple logical deductions to complex mathematical reasoning.

3		2			6			1
8						7		
		6			4			9
		7						
				5				
4	6				2		9	
	4						1	
6	8				1			2
	2			9	5			6

Figure1: Unsolvable sudoku example from sudokuwiki.org

B. Classifying AEASS

When designing AEASS multiple types of EAs were tested for implementation and AEASS is a hybrid of these EAs but the EA that it resembles most can be said to be and Memetic EAs since local search used for repairing children's errors and for local optimization AEASS resembles Lamarckian evolution. But AEASS does not fully fit the classification of MAs because local search used in this algorithm is rather a weak and simple one is not fully systematic and the algorithm features combinations of different techniques not traditionally used in MAs such as population reset for global adaptation.

The main reason for this is AEASS was first designed to be a pure genetic algorithm at first but due to various problems like solutions getting stuck at local optimas frequently, AEASS was improved to include methods from MAs to escape local optimas. The methods used in AEASS are more specific for this problem rather than being more general like pure genetic algorithms for example mutation and crossover operations are specific for solving sudokus or other grid type representations.

II. Algorithm Design (Method)

A. Core Algorithm Architecture

The proposed algorithm combines evolutionary computation with local search strategies, implementing a memetic approach to solve Sudokus. The algorithm maintains fixed numbers from the original puzzle while evolving the solution through genetic operations and local refinement. original puzzle are maintained throughout the evolution

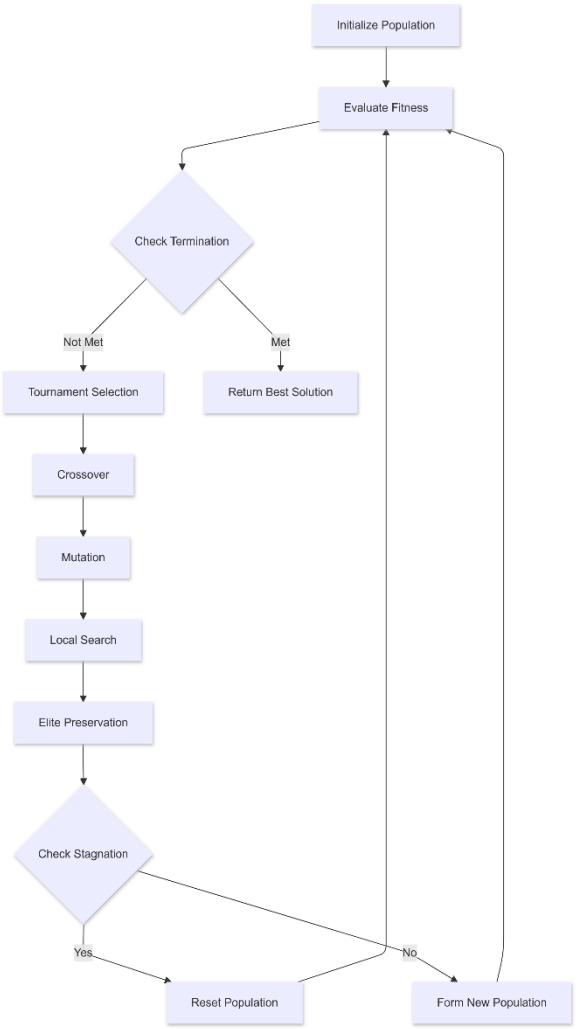


Figure 2: Flowchart of the whole AEASS algorithm

1) Problem Representation

The algorithm utilizes a population of solution candidates where each individual represents a complete 9×9 Sudoku grid. The representation ensures: gg

- Fixed numbers preservation through a binary mask array
- Valid 3×3 block constraints through specialized genetic operators
- Row-wise validity maintenance in initialization

The chromosome structure consists of 81 integers (9×9 grid), partitioned into nine 3×3 blocks for efficient manipulation.

.

	3		2			6		1
	8					7		
			6			4		9
			7					
					5			
	4	6				2	9	
		4					1	
	6	8				1		2
		2		9	5			6

5	3	9	2	7	8	6	4	1
9	8	1	4	2	3	7	5	6
1	7	5	6	1	2	4	8	9
2	2	3	7	8	4	9	6	5
8	9	7	1	6	5	3	2	8
3	4	6	8	3	7	2	9	3
7	5	4	3	5	6	8	1	7
4	6	8	5	4	9	1	3	2
6	1	2	9	9	5	5	7	6

Figures 3-4: Representation and individual initialization of sudokus in our algorithm (red numbers in Figure 4 are fixed values given in Figure 3)

B. Population initialization

1) Individual Initialization

.

The initialization procedure generates valid individuals for problem representation by:

1. Preserving all fixed numbers from the original puzzle
2. For each row, identifying empty cells that need to be filled
3. Creating a random permutation of the remaining numbers (1-9) that are not fixed in that row
4. Assigning these numbers to the empty cells

2) Population Initialization

The initialization process in AEASS is designed to create a diverse yet valid starting population while respecting the constraints of the Sudoku puzzle. Population is initialized in the solve function of our algorithm.

- Generates specified population size
- Each individual is a complete grid
- All individuals respect fixed numbers

C. Selection Method

The algorithm employs 2 selection methods with the following characteristics:

1. Tournament Selection
2. Elite Selection

1) Tournament Selection

Tournament selection identifies parent solutions through the following process:

- Randomly sample k (tournament size) individuals
- Evaluate fitness of sampled individuals
- Select the highest fitness individual
- Repeat for second parent if needed for crossover

2) Elite Selection

Elite population selection is crucial for maintaining solution quality while allowing exploration. This feature of our algorithm has features Of Lamarckian Evolution since local search is used to create elite offspring.

Process of elite selection:

- Evaluates all individuals' fitness scores
- Ranks population according to their fitness
- Selects top 5% of population or minimum of 5 elites
- Enhances these elites by performing local search on them to improve their quality

D. Population management

1) New population generation

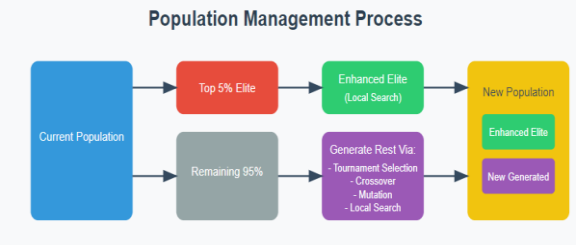


Figure 5

E. Fitness Function

The fitness function evaluates solution quality through constraint violation counting:
fitness = 1 / (1 + total_errors)
where: total_errors = row_errors + column_errors + box_errors
Each error type is weighted equally and represents violations of Sudoku rules:

- Duplicate numbers in rows
- Duplicate numbers in columns
- Duplicate numbers in 3×3 boxes

This formulation creates a maximization problem where a fitness of 1.0 indicates a valid solution.

F. Genetic Operations

1) Crossover Operation

Operation Procedure:

- Selects random 3-row block (top/middle/bottom)
- Copies entire parent1 structure initially
- Replaces selected block with parent2's block

Features:

- Maintains row-wise number uniqueness
- Preserves fixed position numbers
- Specific to Sudoku's 3x3 block structure
- Can be easily modified to work for problems that have a grid structure

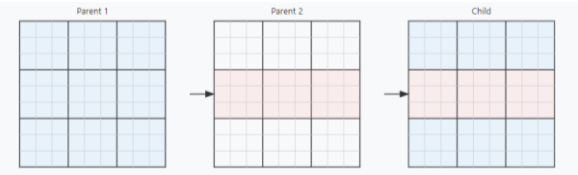


Figure 6 Crossover operation representation

2) Mutation Operators

Two specialized mutation operators are implemented:

a) Swap in Row Operation:

Operation Procedure:

- Selects two non-fixed numbers within the same row randomly
- Only swaps numbers that were not in the original puzzle (non-fixed positions)

Operation Features:

- Fine tuning
- Preserves row-wise number uniqueness since it only swaps existing numbers
- Maintains the validity of fixed position numbers
- Allows for local improvements within a single row
- Can be easily modified to work for problems that have a grid structure
-

b) Swap Rows Operation:

Operation procedure:

- Selects two rows within the same 3x3 block to swap
- Swaps entire rows rather than individual numbers

Operation features:

- Major change similar to the crossover operation so it has lower percentage than swap in row mutation since we need more fine tuning
- Preserves the block structure constraints of Sudoku
- Maintains row uniqueness since it swaps complete valid rows
- Can be easily modified to work for problems that have a grid structure

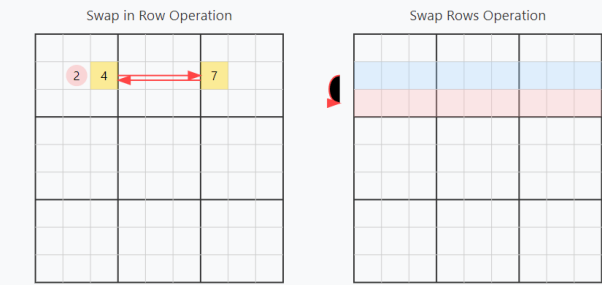


Figure 6 Crossover operation representation

G. Local Search

The local search mechanism in AEASS implements an adaptive neighborhood search strategy with multiple operators. It combines concepts from both traditional local search and variable neighborhood descent, making it particularly effective for Sudoku solving. But does not strictly adhere to traditional rules for local search since it is not fully systematic and neighbors are created using mutation operators. With these exceptions it can be classified as repair focused local search.

1) Neighborhood Structures:

The algorithm employs two primary neighborhood structures using our mutation operators swap row and swap in row.

2) Search Process:

Searches through these created neighborhood structures and tries to find a improved version with higher fitness of the offspring it is applied to.

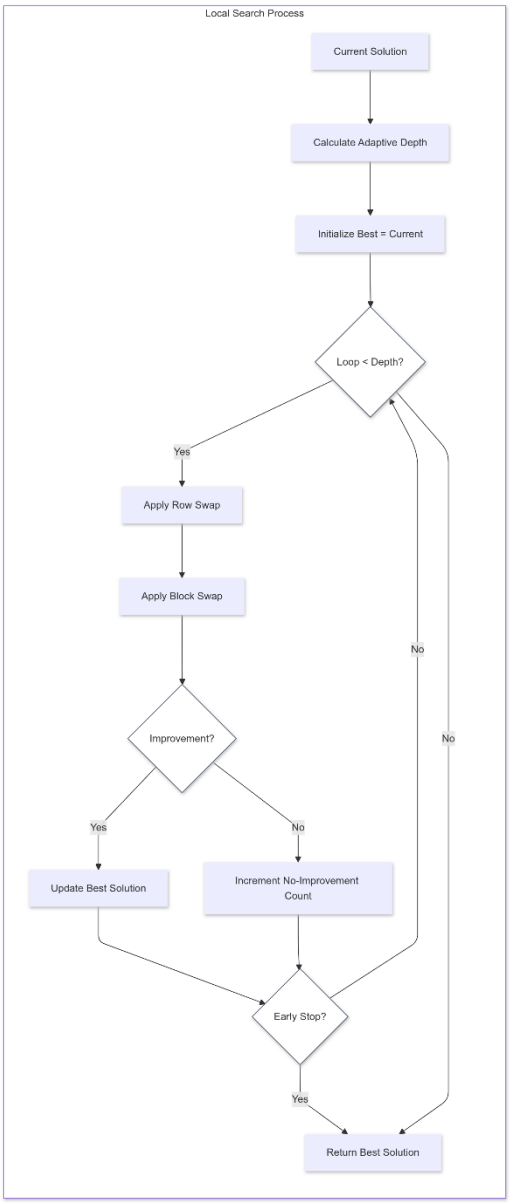


Figure 7 Local search flowchart

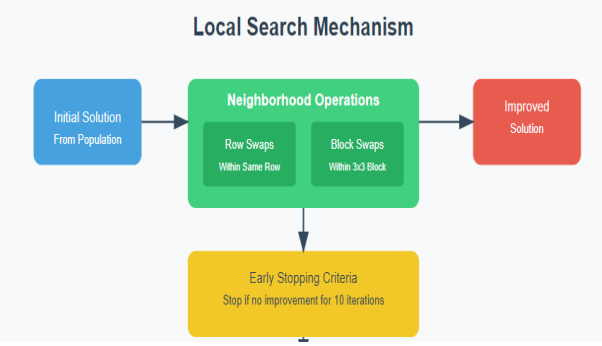


Figure 8

H. *Population Reset Mechanism*

The population reset mechanism in AEASS is a radical diversification strategy that completely regenerates the population when the algorithm stagnates. Unlike regular generational updates that preserve elites, population reset discards ALL current solutions, including elites, to escape local optima and it is used as last resort solution when the algorithm stagnates for too many generations. The best solution is preserved but not part of the new population it is used to keep track of fitness history

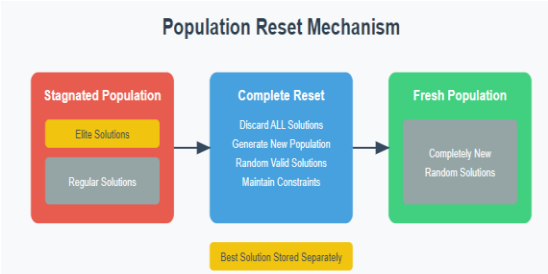


Figure 9

I. *Termination Criteria*

While there are two termination criteria in the algorithm generation count and time limit, the one used mostly is time due to time constraints when testing the implementation of the algorithm. The termination criteria for this algorithm is 200 seconds.

III. Experimental Work

A. *Hyperparameter Optimization*

Note: Worst case scenario takes 27 (combinations) X 10 (runs) X 200 seconds= 54,000seconds or 15 hours This limits testing due to time constraints

The evolutionary algorithm's performance was evaluated across three key parameters:

- Population Size (Pop_Size): 100, 200, and 350
- Local Search Frequency (LSF): 0.2, 0.5, and 0.8
- Crossover Rate (CR): 0.3, 0.5, and 0.7

This resulted in 27 different configurations (3×3×3), with each configuration tested for performance and computational efficiency. The primary metrics for evaluation were:

1. Mean Fitness: Indicating solution quality
2. Mean Time: Computational time in seconds

We want to minimize mean time while maximizing mean fitness. Also an important note for the mean fitness results are due to the constraints of our fitness function some mean values are given as 1 this would indicate a perfect solution but in reality it is just data lost to rounding but mean fitness score being 1 still indicates there were a lot of successful offspring generated

Figure 10 Hyperparameter configuration table

Population Size	LSF	CR	Mean Fitness	Mean Time (s)
100	0.2	0.3	0.8000	73.8
100	0.2	0.5	0.9333	52.7
100	0.2	0.7	1	39.3
100	0.5	0.3	0.9333	72.6
100	0.5	0.5	0.9333	72.8
100	0.5	0.7	0.9333	42.1
100	0.8	0.3	1	57.3
100	0.8	0.5	0.9333	51.9
100	0.8	0.7	1	53.3
200	0.2	0.3	0.9333	60.6
200	0.2	0.5	0.9333	84.5
200	0.2	0.7	0.9333	59.3
200	0.5	0.3	1	76.5
200	0.5	0.5	1	72.9
200	0.5	0.7	0.9333	59.7
200	0.8	0.3	1	57.8
200	0.8	0.5	0.9333	77.1
200	0.8	0.7	1	72.7
350	0.2	0.3	1	91.2
350	0.2	0.5	0.9333	91.4
350	0.2	0.7	1	50.2
350	0.5	0.3	0.8000	106.3
350	0.5	0.5	1	80.7
350	0.5	0.7	1	80.7
350	0.8	0.3	1	52.2
350	0.8	0.5	0.9333	66.3
350	0.8	0.7	0.9333	88.3

B. *Key Findings:*

1.Success Rate Analysis:

- 44.4% of configurations achieved best fitness (1.0)
- The remaining configurations achieved a minimum fitness of 0.8, indicating robust performance across different parameters

2.Parameter Impact Analysis:

a) Population Size:

- Population size 350 showed the highest success rate in achieving best fitness
- However, larger populations generally required more computational time
- Population size 100 showed more variance in performance

b) Local Search Frequency:

- LSF = 0.8 demonstrated the most consistent performance
- Higher LSF values generally led to better solution quality
- Mid-range LSF (0.5) showed balanced performance between quality and time

c) Crossover Rate:

- CR = 0.7 produced the most configurations with best fitness
- Lower CR values (0.3) showed more variance in performance
- Mid-range CR (0.5) demonstrated consistent but not optimal performance

3.Optimal Configuration: Based on the experimental results, the best configuration was:

- Population Size: 100
- Local Search Frequency: 0.2
- Crossover Rate: 0.7

However further testing showed this combination was too unreliable and its performance would vary since lower population and lower LSF made the algorithm less reliable so the third best scoring solution was selected since second and third best scores were close in mean time but having higher LSF makes the algorithm more reliable. The first and third best combinations were tested twice here are the results Note: time limit was 100 seconds instead 200 seconds this is because time constraints and shouldn't effect our results since we are trying to find the combination that finds the correct solution fastest.

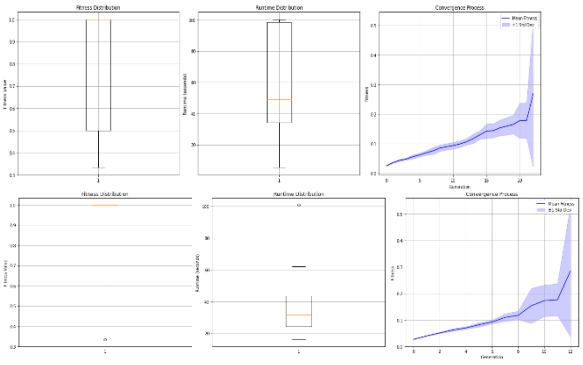
the 3rd best configuration was:

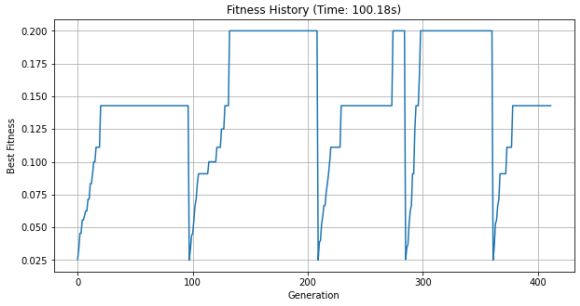
- Population Size: 350
- Local Search Frequency: 0.8
- Crossover Rate: 0.3

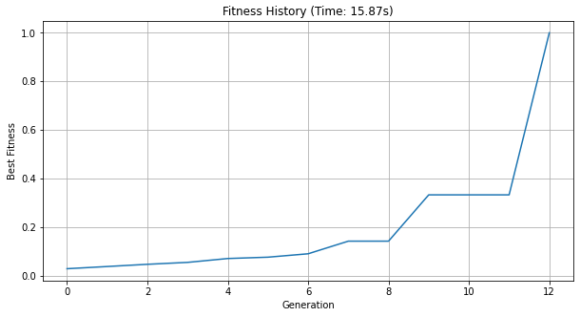
Test configuration:

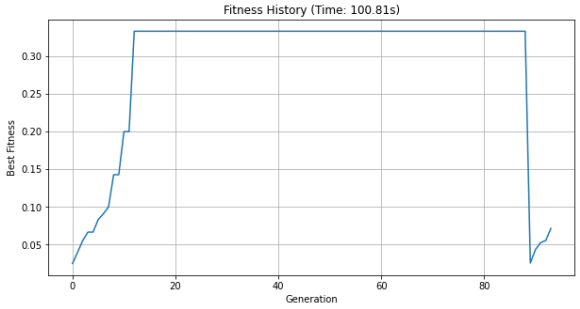
- Population Size: 350
- Local Search Frequency: 0.8
- Crossover Rate: 0.7

1) For the 1st best combination





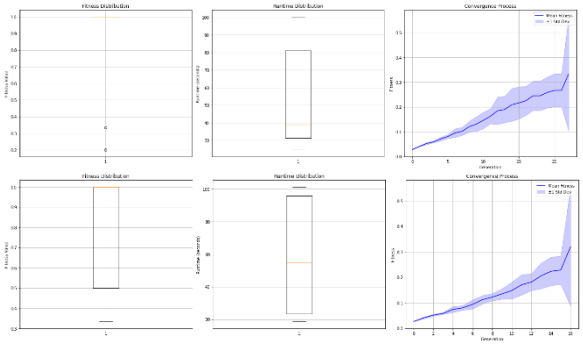


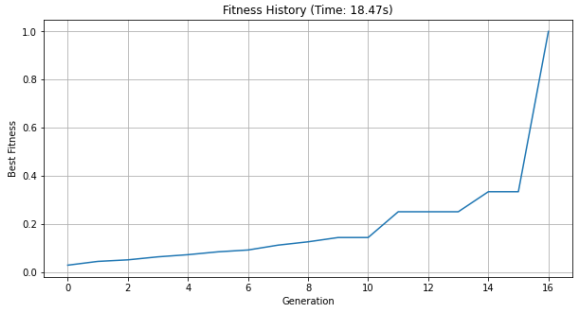


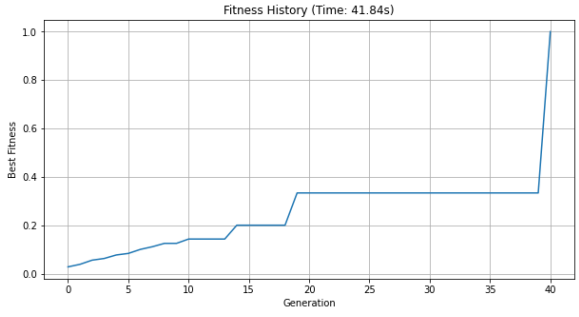

```
== Final Results ===
ime: 93.30s | Generations: 405
inal fitness: 1.0000
itness: 1.0000
untime: 93.39s

inal Statistics:
          Metric Fitness Runtime (s)
            Mean  0.8000       59.34
tandard Deviation  0.3220       36.19
            Best  1.0000           -
           Worst  0.3333           -
```

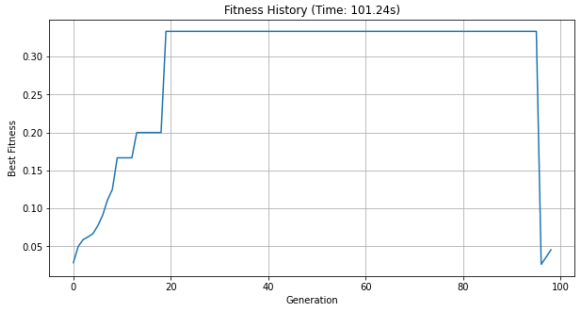
```
Time limit reached - stopping search

=== Final Results ===
Time: 100.04s | Generations: 624
Final fitness: 0.3333
Fitness: 0.3333
Runtime: 100.12s

Final Statistics:
            Metric Fitness Runtime (s)
              Mean  0.8000       64.05
Standard Deviation  0.3220       35.61
              Best  1.0000           -
             Worst  0.3333           -
```

2) For the 3rd best combination





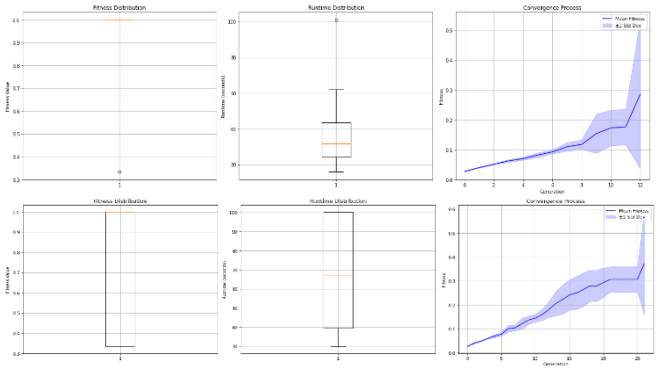


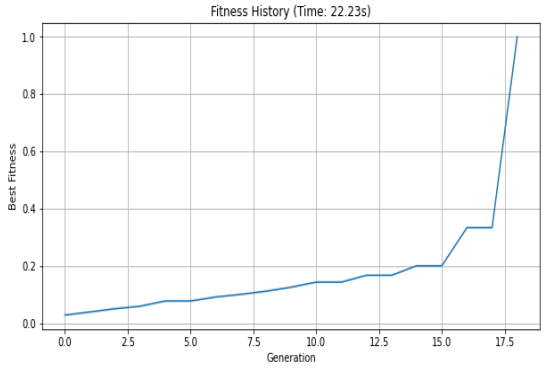


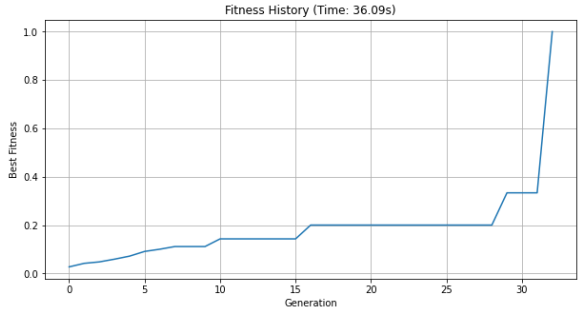

```
Final Statistics:
            Metric Fitness Runtime (s)
              Mean  0.8533       55.51
Standard Deviation  0.3108       30.68
              Best  1.0000           -
             Worst  0.2000           -
```

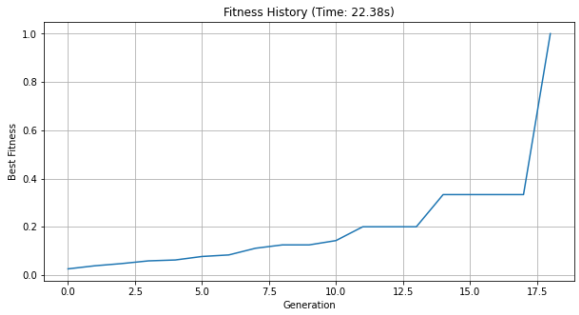
```
Final Statistics:
            Metric Fitness Runtime (s)
              Mean  0.8000       58.20
Standard Deviation  0.3220       36.09
              Best  1.0000           -
             Worst  0.3333           -
```

3) For the test combination










```
Final Statistics:
           Metric Fitness Runtime (s)
             Mean  0.6533       59.58
Standard Deviation  0.3676       42.96
             Best  1.0000           -
            Worst  0.2000           -
```

```
Final Statistics:
           Metric Fitness Runtime (s)
             Mean  0.9333       39.60
Standard Deviation  0.2108       25.32
             Best  1.0000           -
            Worst  0.3333           -
```

Figures 11-32 Test results and example runs for 3 configuration of the algorithm

```
Generation 17 | Time: 21.28s | Fitness: 0.3333
Errors - Row: 0, Col: 2, Box: 0, Total: 2

Current Best

| 3  7  2 | 8  9  6 | 5  4  1 |
| 8  9  4 | 2  1  5 | 7* 6  3 |
| 1  5  6 | 3  7  4 | 2  8* 9 |

| 5  3  7 | 4  6  9 | 1  2  8 |
| 4  6  8 | 1  3  2 | 7* 9  5 |
| 2  1  9 | 5  8  7 | 6  3  4 |

| 6  8  3 | 7  4  1 | 9  5  2 |
| 9  4  5 | 6  2  8 | 3  1  7 |
| 7  2  1 | 9  5  3 | 4  8* 6 |
```

Figure 33 Algorithm getting stuck local optima example

4) Results of these test:

These results show that our algorithm varies a lot regardless of the combination we used if we evaluate results by time and fitness but if we take generation into count account having high LSF is better. The difference between mean fitness can be accounted by time limit difference between the runs.

5) Overall results:

Our algorithm is robust since changing parameters does not effect our algorithms performance if don't care for generation count but gets stuck at local optimas and this greatly reduces our algorithms performance since if it doesn't get stuck at local optima it solves the sudoku in mere seconds and when it gets stuck it can escape eventually but takes most of the algorithms runtime to escape the local optima.

IV. CONCLUSION AND DISCUSSION

This paper presented AEASS, an adaptive evolutionary algorithm for solving Sudoku puzzles that combines concepts from traditional EAs, Lamarckian Evolution, and Memetic Algorithms. Through extensive experimentation and analysis, several key conclusions can be drawn about the algorithm's performance and characteristics:

A. Key Findings:

1) Algorithm Performance:

- AEASS successfully solves both standard and "unsolvable" Sudoku puzzles, demonstrating its effectiveness as a solution method
- The algorithm shows robust performance across different parameter configurations, with 44.4% of tested configurations achieving near optimal fitness
- When not trapped in a local optima, AEASS can solve complex Sudokus in seconds, showing its potential for efficient puzzle solving

2) Parameter Sensitivity:

- High Local Search Frequency (LSF = 0.8) produces more consistent results with less generation count
- Larger population sizes (350) improve solution reliability but increase computational time
- The algorithm demonstrates resilience to parameter changes, suggesting a robust underlying design

B. Limitations and Challenges

1) Local Optima

- The primary limitation of AEASS is its susceptibility to getting trapped in local optima
- While the algorithm can eventually escape local optima through population reset, this process consumes significant computational resources
- The time needed to escape local optima accounts for the majority of solving time in affected runs

2) Performance Variability

- Solution times show considerable variation even with identical parameter settings

C. Contributions

1) Algorithmic Innovations

- Successfully integrated concepts from multiple evolutionary computation paradigms
- Developed specialized genetic operators for Sudoku grid structures
- Implemented an adaptive local search mechanism with variable intensity

2) Practical Applications

- Demonstrated viability for solving puzzles deemed "unsolvable" by traditional methods
- Provided insights into parameter selection for evolutionary Sudoku solvers
- Created a framework adaptable to other grid-based constraint satisfaction problems

D. Future Work

Several promising directions for future research emerge from this work:

1. Algorithm Enhancements
   - Development of more sophisticated local optima escape mechanisms
   - Investigation of alternative local search strategies
   - Implementation of adaptive parameter control during runtime
2. Performance Optimization
   - Exploration of parallel implementation possibilities
   - Research into more efficient local search patterns
   - Testing all hyperparameter optimizations :
     a. population_size
     b. local_search_frequency
     c. tournament_size
     d. crossover_rate
     e. mutation_rate
     f. stagnation_threshold
     g. local_search_depth
     h. early_ls_stop
     i. Elite percentage

   Testing for all would be optimal but due to time constraints can't be tested now
3. Application Extensions
   - Adaptation of AEASS to other grid-based puzzles and constraint satisfaction problems
   - Investigation of scalability to larger grid sizes
   - Development of hybrid approaches combining AEASS with traditional solving techniques

E. Conclusion

In conclusion, while AEASS demonstrates promising capabilities in solving complex Sudoku puzzles, its primary challenge lies in local optima management. The algorithm's robust performance across different parameter configurations suggests its potential as a foundation for future evolutionary Sudoku solvers, while its limitations provide clear directions for future improvements. The success of AEASS in solving "unsolvable" puzzles validates the effectiveness of combining multiple evolutionary computation paradigms in tackling complex constraint satisfaction problems.

V. References

[1] https://www.britannica.com/topic/sudoku

[2] T. Yato and T. Seta, Complexity and Completeness of Finding Another Solution and Its Application to Puzzles, Graduate School of Information Science and Technology The University of Tokyo,2003

[3] Pablo Moscato and Carlos Cotta, A Modern Introduction to Memetic Algorithms, University of Newcastle Australia, University of Malaga,2010

[4] https://www.geeksforgeeks.org/lamarckian-evolution-and-baldwin-effect-in-evolutionary/

[5] Rhyd Lewis, Metaheuristics can Solve Sudoku Puzzles, Centre for Emergent Computing, Napier University, Scotland.2007

[6] S. K. Jones, P. A. Roach, S. Perkins, Construction of Heuristics for a Search-Based Approach to Solving Sudoku, Department of Computing and Mathematical Sciences, University of Glamorgan, Pontypridd,2007

[7] Timo Mantere and Janne Koljonen, Solving, Rating and Generating Sudoku Puzzles with GA,