SUDOKU

Originally called Number Place is a logic-based, combinatorial number-placement puzzle. In classic Sudoku, the objective is to fill a 9 × 9 grid with digits so that each column, each row, and each of the nine 3 × 3 subgrids that compose the grid (also called "boxes", "blocks", or "regions") contain all of the digits from 1 to 9. The puzzle setter provides a partially completed grid, which for a well-posed puzzle has a single solution.

A genetic algorithm was used to solve the problem.

In [115]:

```
import pygad
import random
import time
```

In [116]:

```
# solved sudoku boards

solved_sudoku_1 = [[4, 6, 9, 3, 8, 2, 1, 5, 7],
                   [2, 8, 1, 4, 7, 5, 3, 9, 6],
                   [7, 3, 5, 9, 1, 6, 2, 8, 4],
                   [5, 1, 3, 8, 9, 4, 6, 7, 2],
                   [6, 9, 4, 5, 2, 7, 8, 3, 1],
                   [8, 7, 2, 1, 6, 3, 5, 4, 9],
                   [1, 5, 8, 2, 4, 9, 7, 6, 3],
                   [9, 2, 6, 7, 3, 8, 4, 1, 5],
                   [3, 4, 7, 6, 5, 1, 9, 2, 8]]

solved_sudoku_2 = [[2, 9, 5, 4, 8, 1, 7, 3, 6],
                   [1, 4, 7, 6, 3, 9, 2, 8, 5],
                   [8, 3, 6, 7, 5, 2, 9, 4, 1],
                   [6, 8, 4, 5, 1, 7, 3, 9, 2],
                   [7, 5, 3, 9, 2, 8, 1, 6, 4],
                   [9, 1, 2, 3, 6, 4, 8, 5, 7],
                   [5, 6, 1, 2, 9, 3, 4, 7, 8],
                   [3, 7, 8, 1, 4, 6, 5, 2, 9],
                   [4, 2, 9, 8, 7, 5, 6, 1, 3]]

solved_sudoku_3 = [[6, 7, 1, 3, 5, 8, 2, 4, 9],
                   [8, 9, 3, 7, 4, 2, 6, 5, 1],
                   [2, 4, 5, 9, 6, 1, 8, 7, 3],
                   [4, 5, 9, 6, 7, 3, 1, 2, 8],
                   [3, 6, 8, 1, 2, 5, 7, 9, 4],
                   [7, 1, 2, 8, 9, 4, 3, 6, 5],
                   [9, 3, 4, 2, 1, 7, 5, 8, 6],
                   [1, 2, 6, 5, 8, 9, 4, 3, 7],
                   [5, 8, 7, 4, 3, 6, 9, 1, 2]]

solved_sudoku_list = [solved_sudoku_1, solved_sudoku_2, solved_sudoku_3]
```

In [117]:

```python
# generating sudoku with a given number of empty fields

def generate_sudoku(solved, gaps):
    sudoku = [[0 for i in range(9)] for j in range(9)]
    for i in range(9):
        for j in range(9):
            sudoku[i][j] = solved[i][j]

    while gaps > 0:
        i = random.randint(0, 8)
        j = random.randint(0, 8)
        if sudoku[i][j] != 0:
            sudoku[i][j] = 0
            gaps -= 1

    return sudoku


# chosen sudoku to solve
selected_sudoku = random.choice(solved_sudoku_list)
generated_sudoku = generate_sudoku(selected_sudoku, 30)

for i in generated_sudoku:
    print(i)
```

```
[4, 6, 9, 3, 0, 0, 0, 5, 0]
[0, 8, 0, 4, 0, 5, 0, 9, 0]
[0, 3, 0, 9, 1, 6, 2, 8, 4]
[5, 0, 3, 8, 0, 4, 6, 7, 0]
[6, 0, 4, 0, 2, 0, 8, 3, 0]
[8, 0, 0, 1, 6, 3, 5, 0, 9]
[0, 0, 8, 2, 4, 9, 0, 6, 0]
[9, 2, 0, 7, 3, 8, 4, 0, 5]
[3, 4, 7, 0, 5, 0, 0, 2, 8]
```

In [118]:

```
# generating empty board
def empty_board():
    return [[0 for i in range(9)] for j in range(9)]

# checking the number of unique values in a row
def check_row(board, row):
    return len(set(board[row]))

# checking the number of unique values in a column
def check_col(board, col):
    column = [board[i][col] for i in range(9)]
    return len(set(column))

# checking the number of unique values in a 3x3 square
def check_box(board, row, col):
    box = [board[i][j] for i in range(row, row+3) for j in range(col, col+3)]
    return len(set(box))

# filling sudoku with values from the solution
def fill_in(solution):
    filled_sudoku = empty_board()
    sol_index = 0
    for i in range(9):
        for j in range(9):
            if generated_sudoku[i][j] != 0:
                filled_sudoku[i][j] = int(generated_sudoku[i][j])
            else:
                filled_sudoku[i][j] = int(solution[sol_index])
                sol_index += 1
    return filled_sudoku
```

In [119]:

```
# defining chromosome parameters
gene_space = [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [120]:

```
# defining fitness function
def fitness_func(solution, solution_idx):
    filled_sudoku = fill_in(solution)
    fitness = 0
    for i in range(9):
        fitness += check_row(filled_sudoku, i)
        fitness += check_col(filled_sudoku, i)
    for i in range(0, 9, 3):
        for j in range(0, 9, 3):
            fitness += check_box(filled_sudoku, i, j)
    return fitness

fitness_function = fitness_func
```

In [121]:

```
# defining function counting the number of genes
def genes_number(sudoku):
    genes = 0
    for i in range(9):
        for j in range(9):
            if sudoku[i][j] == 0:
                genes += 1
    return genes
```

In [122]:

```
# amount of chromosomes in population
# amount of genes in chromosome
sol_per_pop = 400
num_genes = genes_number(generated_sudoku)

num_parents_mating = sol_per_pop // 2 # about 50% of population
num_generations = 5000
keep_parents = sol_per_pop // 13 # about 7% of population

parent_selection_type = "sss"

crossover_type = "single_point"

mutation_type = "random"
mutation_percent_genes = 2
```

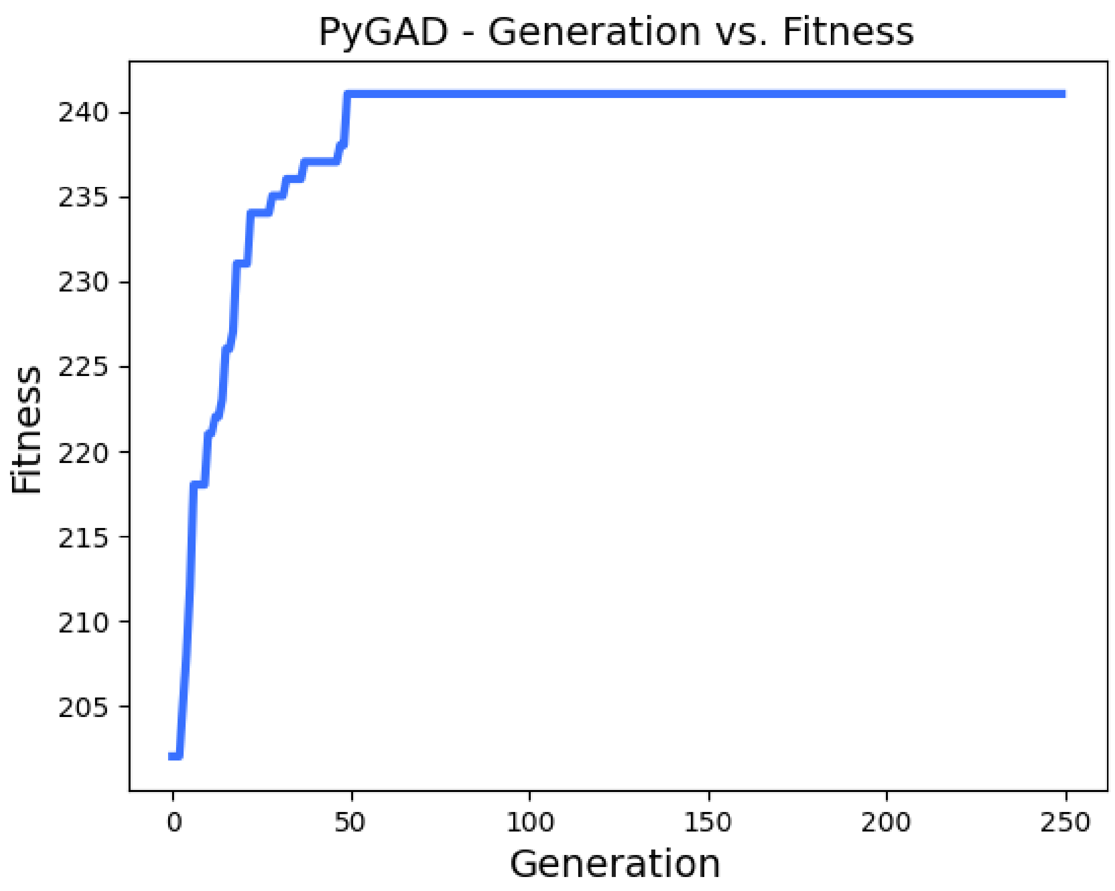
In [123]:

```
# algorithm initialization with the above parameters written in attributes
ga_instance = pygad.GA(gene_space=gene_space,
                       num_generations=num_generations,
                       num_parents_mating=num_parents_mating,
                       fitness_func=fitness_function,
                       sol_per_pop=sol_per_pop,
                       num_genes=num_genes,
                       parent_selection_type=parent_selection_type,
                       keep_parents=keep_parents,
                       crossover_type=crossover_type,
                       mutation_type=mutation_type,
                       mutation_percent_genes=mutation_percent_genes,
                       stop_criteria=["reach_243", "saturate_200"]
                       )

# algorithm run
start_time = time.time()
ga_instance.run()
end_time = time.time()
time_passed = end_time - start_time
```

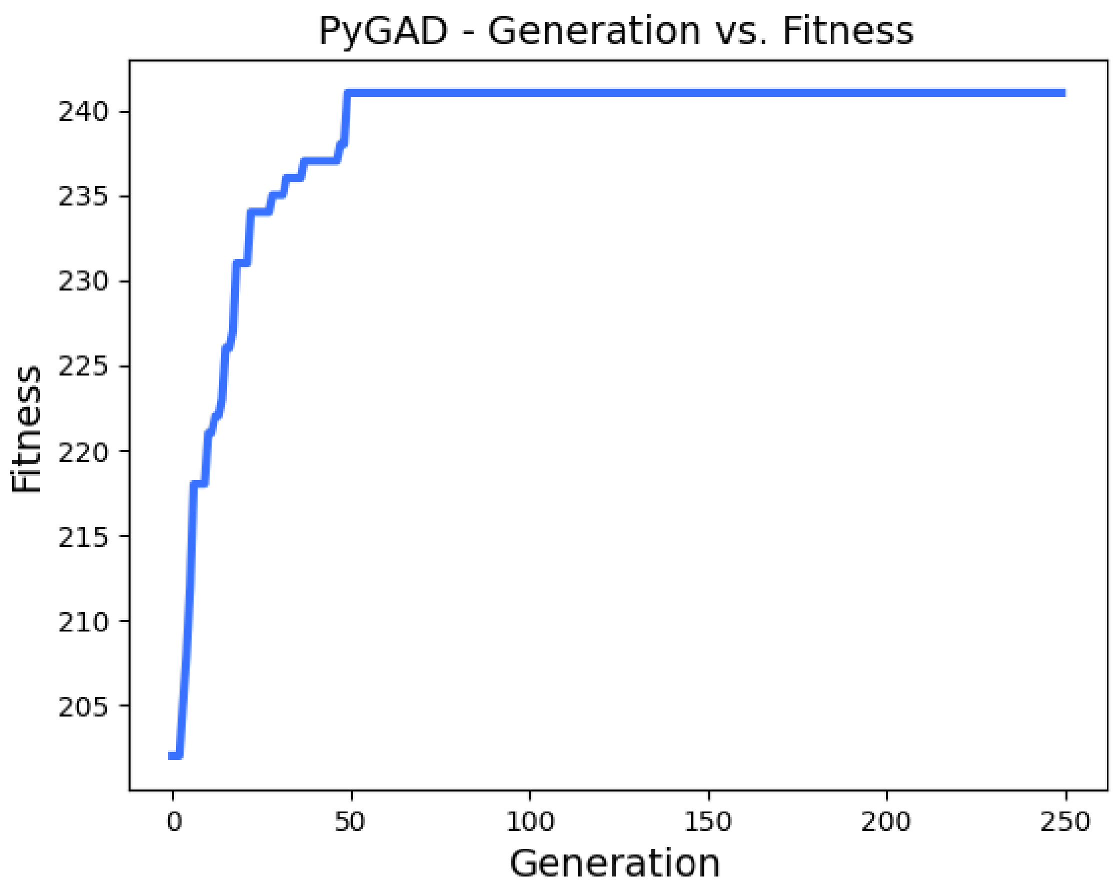
In [124]:

```
# summary: best found solution (chromosome + evaluation)
solution, solution_fitness, solution_idx = ga_instance.best_solution()
print("Parameters of the best solution: {solution}".format(solution=solution))
print("Fitness value of the best solution (solved=243) = {solution_fitness}".format(solu

# amount of generations passed
print("Number of generations passed is {generations_completed}".format(generations_compl

# algorithm run time
print("Time passed: {time_passed}".format(time_passed=time_passed) + "\n")

# display the best solution found
print("Best solution found is:")
for i in fill_in(solution):
    print(i)
print("")

# display the correct solution
print("Correct solution is:")
for i in selected_sudoku:
    print(i)

# display plot: how the evaluation changed over the generations
ga_instance.plot_fitness()
```

```
Parameters of the best solution: [8. 2. 1. 7. 2. 1. 7. 3. 6. 7. 5. 9. 9.
1. 1. 5. 7. 2. 7. 2. 4. 1. 5. 7.
 3. 6. 1. 6. 1. 9.]
Fitness value of the best solution (solved=243) = 241
Number of generations passed is 249
Time passed: 19.840336322784424

Best solution found is:
[4, 6, 9, 3, 8, 2, 1, 5, 7]
[2, 8, 1, 4, 7, 5, 3, 9, 6]
[7, 3, 5, 9, 1, 6, 2, 8, 4]
[5, 9, 3, 8, 9, 4, 6, 7, 1]
[6, 1, 4, 5, 2, 7, 8, 3, 2]
[8, 7, 2, 1, 6, 3, 5, 4, 9]
[1, 5, 8, 2, 4, 9, 7, 6, 3]
[9, 2, 6, 7, 3, 8, 4, 1, 5]
[3, 4, 7, 6, 5, 1, 9, 2, 8]

Correct solution is:
[4, 6, 9, 3, 8, 2, 1, 5, 7]
[2, 8, 1, 4, 7, 5, 3, 9, 6]
[7, 3, 5, 9, 1, 6, 2, 8, 4]
[5, 1, 3, 8, 9, 4, 6, 7, 2]
[6, 9, 4, 5, 2, 7, 8, 3, 1]
[8, 7, 2, 1, 6, 3, 5, 4, 9]
[1, 5, 8, 2, 4, 9, 7, 6, 3]
[9, 2, 6, 7, 3, 8, 4, 1, 5]
[3, 4, 7, 6, 5, 1, 9, 2, 8]
```


PyGAD - Generation vs. Fitness
Fitness
Generation
240
235
230
225
220
215
210
205
0
50
100
150
200
250

Out[124]:


PyGAD - Generation vs. Fitness
Fitness
Generation
240
235
230
225
220
215
210
205
0
50
100
150
200
250

In [125]:

```python
# checking the efficiency of the algorithm for small sudoku
small_solved_cases = 0
small_avg_time = 0
for i in range(100):
    selected_sudoku = random.choice(solved_sudoku_list)
    generated_sudoku = generate_sudoku(selected_sudoku, 10)
    num_genes = genes_number(generated_sudoku)
    # algorithm initialization with the above parameters written in attributes
    ga_instance = pygad.GA(gene_space=gene_space,
                        num_generations=num_generations,
                        num_parents_mating=num_parents_mating,
                        fitness_func=fitness_function,
                        sol_per_pop=sol_per_pop,
                        num_genes=num_genes,
                        parent_selection_type=parent_selection_type,
                        keep_parents=keep_parents,
                        crossover_type=crossover_type,
                        mutation_type=mutation_type,
                        mutation_percent_genes=mutation_percent_genes,
                        stop_criteria=["reach_243", "saturate_200"]
                        )
    # algorithm run
    start_time = time.time()
    ga_instance.run()
    end_time = time.time()
    solution, solution_fitness, solution_idx = ga_instance.best_solution()
    if solution_fitness == 243:
        small_solved_cases += 1
        small_avg_time += end_time - start_time

print("Small solved cases: ", small_solved_cases)
if small_solved_cases != 0:
    print("Small avg time: ", small_avg_time / small_solved_cases)
```

```
Small solved cases:  100
Small avg time:  0.9638094091415406
```

Small solved cases: 100

Small avg time: 0.9638094091415406

In [126]:

```
# checking the efficiency of the algorithm for medium sudoku
medium_solved_cases = 0
medium_avg_time = 0
for i in range(100):
    selected_sudoku = random.choice(solved_sudoku_list)
    generated_sudoku = generate_sudoku(selected_sudoku, 20)
    num_genes = genes_number(generated_sudoku)
    # algorithm initialization with the above parameters written in attributes
    ga_instance = pygad.GA(gene_space=gene_space,
                        num_generations=num_generations,
                        num_parents_mating=num_parents_mating,
                        fitness_func=fitness_function,
                        sol_per_pop=sol_per_pop,
                        num_genes=num_genes,
                        parent_selection_type=parent_selection_type,
                        keep_parents=keep_parents,
                        crossover_type=crossover_type,
                        mutation_type=mutation_type,
                        mutation_percent_genes=mutation_percent_genes,
                        stop_criteria=["reach_243", "saturate_200"]
                        )
    # algorithm run
    start_time = time.time()
    ga_instance.run()
    end_time = time.time()
    solution, solution_fitness, solution_idx = ga_instance.best_solution()
    if solution_fitness == 243:
        medium_solved_cases += 1
        medium_avg_time += end_time - start_time

print("Medium solved cases: ", medium_solved_cases)
if medium_solved_cases != 0:
    print("Medium avg time: ", medium_avg_time / medium_solved_cases)
```

```
Medium solved cases:  99
Medium avg time:  2.701251234671082
```

Medium solved cases: 99

Medium avg time: 2.701251234671082

In [127]:

```
# checking the efficiency of the algorithm for big sudoku
big_solved_cases = 0
big_avg_time = 0
for i in range(100):
    selected_sudoku = random.choice(solved_sudoku_list)
    generated_sudoku = generate_sudoku(selected_sudoku, 30)
    num_genes = genes_number(generated_sudoku)
    # algorithm initialization with the above parameters written in attributes
    ga_instance = pygad.GA(gene_space=gene_space,
                        num_generations=num_generations,
                        num_parents_mating=num_parents_mating,
                        fitness_func=fitness_function,
                        sol_per_pop=sol_per_pop,
                        num_genes=num_genes,
                        parent_selection_type=parent_selection_type,
                        keep_parents=keep_parents,
                        crossover_type=crossover_type,
                        mutation_type=mutation_type,
                        mutation_percent_genes=mutation_percent_genes,
                        stop_criteria=["reach_243", "saturate_200"]
                        )
    # algorithm run
    start_time = time.time()
    ga_instance.run()
    end_time = time.time()
    solution, solution_fitness, solution_idx = ga_instance.best_solution()
    if solution_fitness == 243:
        big_solved_cases += 1
        big_avg_time += end_time - start_time

print("Big solved cases: ", big_solved_cases)
if big_solved_cases != 0:
    print("Big avg time: ", big_avg_time / big_solved_cases)
```

```
Big solved cases:  86
Big avg time:  5.717752026957135
```

Big solved cases: 86

Big avg time: 5.717752026957135

Second solution of the problem using genetic algorithm with the use of another fitness function

In [131]:

```python
# amount of unique rows in sudoku
def unique_rows(sudoku):
    counter = 0
    for i in range(9):
        if len(set(sudoku[i])) == 9:
            counter += 1
    return counter

# amount of unique columns in sudoku
def unique_columns(sudoku):
    counter = 0
    for i in range(9):
        column = []
        for j in range(9):
            column.append(sudoku[j][i])
        if len(set(column)) == 9:
            counter += 1
    return counter

# amount of unique 3x3 squares in sudoku
def unique_squares(sudoku):
    counter = 0
    for i in range(0, 9, 3):
        for j in range(0, 9, 3):
            square = []
            for k in range(3):
                for l in range(3):
                    square.append(sudoku[i + k][j + l])
            if len(set(square)) == 9:
                counter += 1
    return counter
```

In [132]:

```python
# new fitness function
def new_fitness_function(solution, solution_idx):
    sudoku = fill_in(solution)
    return unique_rows(sudoku) + unique_columns(sudoku) + unique_squares(sudoku)

fitness_function = new_fitness_function
```

In [133]:

```
# sudoku selection
selected_sudoku = random.choice(solved_sudoku_list)
generated_sudoku = generate_sudoku(selected_sudoku, 30)
num_genes = genes_number(generated_sudoku)

for i in generated_sudoku:
    print(i)
```

```
[6, 0, 0, 3, 5, 8, 2, 4, 9]
[8, 9, 3, 7, 4, 2, 6, 0, 1]
[2, 4, 0, 0, 6, 0, 8, 0, 0]
[4, 5, 9, 0, 7, 0, 1, 2, 0]
[3, 0, 8, 0, 2, 5, 7, 0, 4]
[7, 1, 2, 8, 0, 4, 0, 0, 5]
[0, 3, 4, 0, 1, 0, 5, 0, 6]
[1, 2, 0, 0, 8, 0, 4, 0, 7]
[0, 8, 0, 4, 0, 6, 0, 1, 0]
```

In [134]:

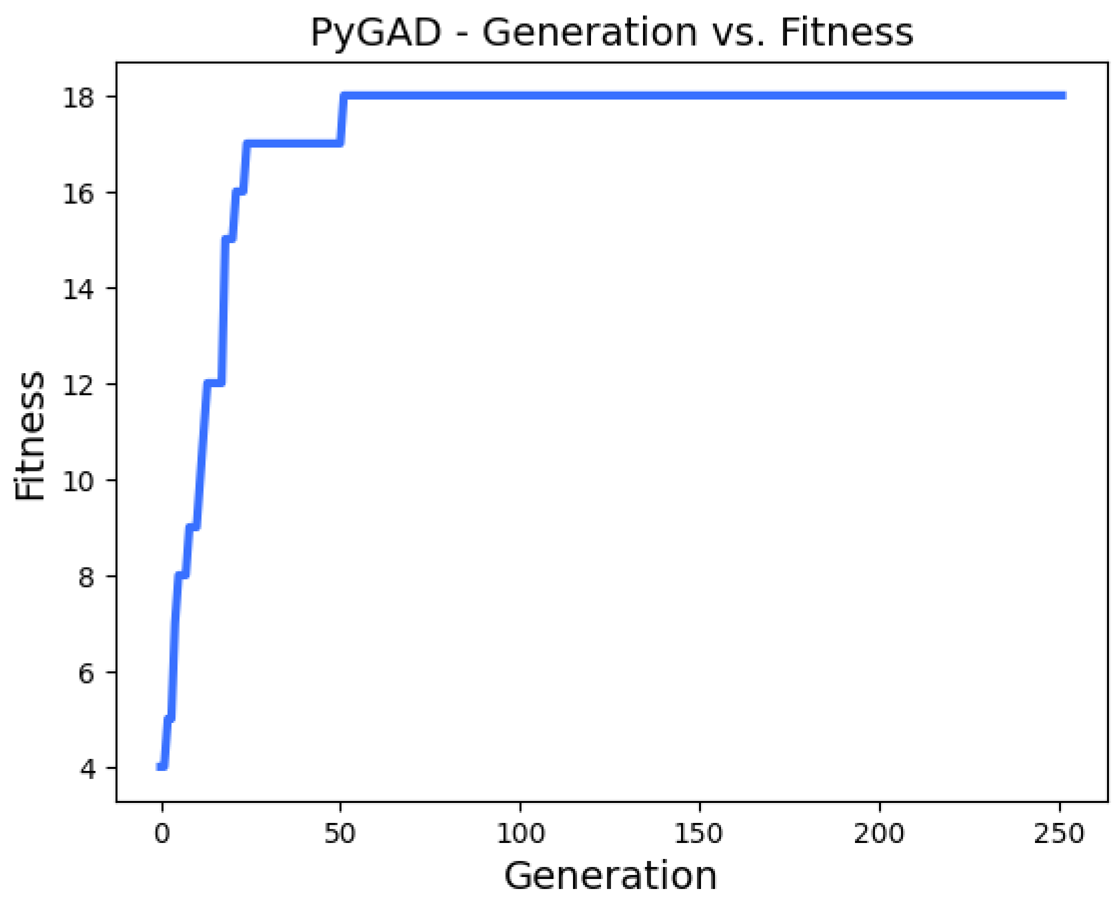
```
# initialization of the algorithm with the above parameters written in attributes
ga_instance = pygad.GA(gene_space=gene_space,
                       num_generations=num_generations,
                       num_parents_mating=num_parents_mating,
                       fitness_func=fitness_function,
                       sol_per_pop=sol_per_pop,
                       num_genes=num_genes,
                       parent_selection_type=parent_selection_type,
                       keep_parents=keep_parents,
                       crossover_type=crossover_type,
                       mutation_type=mutation_type,
                       mutation_percent_genes=mutation_percent_genes,
                       stop_criteria=["reach_27", "saturate_200"]
                       )

# algorithm run
start_time = time.time()
ga_instance.run()
end_time = time.time()
time_passed = end_time - start_time
```

```
C:\Users\szymo\AppData\Roaming\Python\Python311\site-packages\pygad\pyga
d.py:522: UserWarning: The percentage of genes to mutate (mutation_percen
t_genes=2) resutled in selecting (0) genes. The number of genes to mutate
is set to 1 (mutation_num_genes=1).
If you do not want to mutate any gene, please set mutation_type=None.
  if not self.suppress_warnings: warnings.warn("The percentage of genes t
o mutate (mutation_percent_genes={mutation_percent}) resutled in selectin
g ({mutation_num}) genes. The number of genes to mutate is set to 1 (muta
tion_num_genes=1).\nIf you do not want to mutate any gene, please set mut
ation_type=None.".format(mutation_percent=mutation_percent_genes, mutatio
n_num=mutation_num_genes))
```

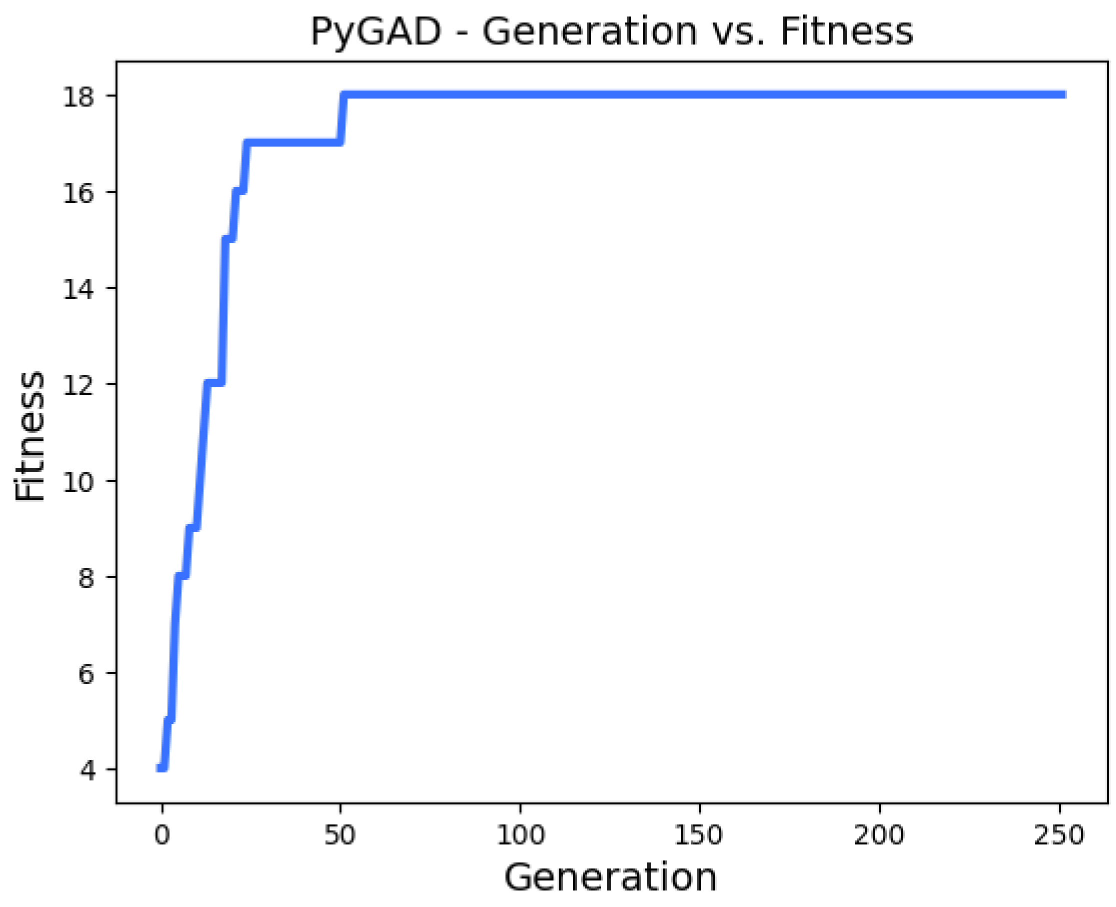
In [135]:

```
# summary: the best found solution (chromosome + evaluation)
solution, solution_fitness, solution_idx = ga_instance.best_solution()
print("Parameters of the best solution: {solution}".format(solution=solution))
print("Fitness value of the best solution (solved=27) = {solution_fitness}".format(solut

# amount of generations passed
print("Number of generations passed is {generations_completed}".format(generations_compl

# algorithm run time
print("Time passed: {time_passed}".format(time_passed=time_passed) + "\n")

# best solution found
print("Best solution found is:")
for i in fill_in(solution):
    print(i)
print("")

# correct solution
print("Correct solution is:")
for i in selected_sudoku:
    print(i)

# display plot: how the evaluation changed over the generations
ga_instance.plot_fitness()
```

```
Parameters of the best solution: [7. 1. 5. 5. 1. 9. 3. 7. 9. 6. 4. 6. 1.
9. 3. 9. 6. 9. 2. 8. 7. 7. 3. 5.
 8. 5. 6. 9. 3. 3.]
Fitness value of the best solution (solved=243) = 18
Number of generations passed is 251
Time passed: 25.665995836257935

Best solution found is:
[6, 7, 1, 3, 5, 8, 2, 4, 9]
[8, 9, 3, 7, 4, 2, 6, 5, 1]
[2, 4, 5, 1, 6, 9, 8, 3, 7]
[4, 5, 9, 9, 7, 6, 1, 2, 4]
[3, 6, 8, 1, 2, 5, 7, 9, 4]
[7, 1, 2, 8, 3, 4, 9, 6, 5]
[9, 3, 4, 2, 1, 8, 5, 7, 6]
[1, 2, 7, 3, 8, 5, 4, 8, 7]
[5, 8, 6, 4, 9, 6, 3, 1, 3]

Correct solution is:
[6, 7, 1, 3, 5, 8, 2, 4, 9]
[8, 9, 3, 7, 4, 2, 6, 5, 1]
[2, 4, 5, 9, 6, 1, 8, 7, 3]
[4, 5, 9, 6, 7, 3, 1, 2, 8]
[3, 6, 8, 1, 2, 5, 7, 9, 4]
[7, 1, 2, 8, 9, 4, 3, 6, 5]
[9, 3, 4, 2, 1, 7, 5, 8, 6]
[1, 2, 6, 5, 8, 9, 4, 3, 7]
[5, 8, 7, 4, 3, 6, 9, 1, 2]
```


PyGAD - Generation vs. Fitness
Fitness
Generation
18
16
14
12
10
8
6
4
0
50
100
150
200
250

```
Out[135]:
```


PyGAD - Generation vs. Fitness
Fitness
Generation
18
16
14
12
10
8
6
4
0
50
100
150
200
250

In [136]:

```
# checking the efficiency of the algorithm for small sudoku
small_solved_cases = 0
small_avg_time = 0
for i in range(100):
    selected_sudoku = random.choice(solved_sudoku_list)
    generated_sudoku = generate_sudoku(selected_sudoku, 10)
    num_genes = genes_number(generated_sudoku)
    # algorithm initialization with the above parameters written in attributes
    ga_instance = pygad.GA(gene_space=gene_space,
                           num_generations=num_generations,
                           num_parents_mating=num_parents_mating,
                           fitness_func=fitness_function,
                           sol_per_pop=sol_per_pop,
                           num_genes=num_genes,
                           parent_selection_type=parent_selection_type,
                           keep_parents=keep_parents,
                           crossover_type=crossover_type,
                           mutation_type=mutation_type,
                           mutation_percent_genes=mutation_percent_genes,
                           stop_criteria=["reach_27", "saturate_200"]
                           )
    # alghorithm run
    start_time = time.time()
    ga_instance.run()
    end_time = time.time()
    solution, solution_fitness, solution_idx = ga_instance.best_solution()
    if solution_fitness == 27:
        small_solved_cases += 1
        small_avg_time += end_time - start_time

print("Small solved cases: ", small_solved_cases)
if small_solved_cases != 0:
    print("Small avg time: ", small_avg_time / small_solved_cases)
```

```
C:\Users\szymo\AppData\Roaming\Python\Python311\site-packages\pygad\pyga
d.py:522: UserWarning: The percentage of genes to mutate (mutation_percen
t_genes=2) resutled in selecting (0) genes. The number of genes to mutate
is set to 1 (mutation_num_genes=1).
If you do not want to mutate any gene, please set mutation_type=None.
  if not self.suppress_warnings: warnings.warn("The percentage of genes t
o mutate (mutation_percent_genes={mutation_percent}) resutled in selectin
g ({mutation_num}) genes. The number of genes to mutate is set to 1 (muta
tion_num_genes=1).\nIf you do not want to mutate any gene, please set mut
ation_type=None.".format(mutation_percent=mutation_percent_genes, mutatio
n_num=mutation_num_genes))

Small solved cases:  100
Small avg time:  1.023431544303894
```

Small solved cases: 100

Small avg time: 1.023431544303894

```
In [137]:
```

```
# checking the efficiency of the algorithm for medium sudoku
medium_solved_cases = 0
medium_avg_time = 0
for i in range(100):
    selected_sudoku = random.choice(solved_sudoku_list)
    generated_sudoku = generate_sudoku(selected_sudoku, 20)
    num_genes = genes_number(generated_sudoku)
    # algorithm initialization with the above parameters written in attributes
    ga_instance = pygad.GA(gene_space=gene_space,
                          num_generations=num_generations,
                          num_parents_mating=num_parents_mating,
                          fitness_func=fitness_function,
                          sol_per_pop=sol_per_pop,
                          num_genes=num_genes,
                          parent_selection_type=parent_selection_type,
                          keep_parents=keep_parents,
                          crossover_type=crossover_type,
                          mutation_type=mutation_type,
                          mutation_percent_genes=mutation_percent_genes,
                          stop_criteria=["reach_27", "saturate_200"]
                          )
    # alghorithm run
    start_time = time.time()
    ga_instance.run()
    end_time = time.time()
    solution, solution_fitness, solution_idx = ga_instance.best_solution()
    if solution_fitness == 27:
        medium_solved_cases += 1
        medium_avg_time += end_time - start_time

print("Medium solved cases: ", medium_solved_cases)
if medium_solved_cases != 0:
    print("Medium avg time: ", medium_avg_time / medium_solved_cases)
```

```
Medium solved cases:  96
Medium avg time:  4.133039007584254
```

Medium solved cases: 96

Medium avg time: 4.133039007584254

In [138]:

```
# checking the efficiency of the algorithm for big sudoku
big_solved_cases = 0
big_avg_time = 0
for i in range(100):
    selected_sudoku = random.choice(solved_sudoku_list)
    generated_sudoku = generate_sudoku(selected_sudoku, 30)
    num_genes = genes_number(generated_sudoku)
    # algorithm initialization with the above parameters written in attributes
    ga_instance = pygad.GA(gene_space=gene_space,
                          num_generations=num_generations,
                          num_parents_mating=num_parents_mating,
                          fitness_func=fitness_function,
                          sol_per_pop=sol_per_pop,
                          num_genes=num_genes,
                          parent_selection_type=parent_selection_type,
                          keep_parents=keep_parents,
                          crossover_type=crossover_type,
                          mutation_type=mutation_type,
                          mutation_percent_genes=mutation_percent_genes,
                          stop_criteria=["reach_27", "saturate_200"]
                          )
    # alghorithm run
    start_time = time.time()
    ga_instance.run()
    end_time = time.time()
    solution, solution_fitness, solution_idx = ga_instance.best_solution()
    if solution_fitness == 27:
        big_solved_cases += 1
        big_avg_time += end_time - start_time

print("Big solved cases: ", big_solved_cases)
if big_solved_cases != 0:
    print("Big avg time: ", big_avg_time / big_solved_cases)
```

```
Big solved cases:  40
Big avg time:  11.377614229917526
```

Big solved cases: 40

Big avg time: 11.377614229917526

SUMMARY

Genetic algorithm works quite well in the case of sudoku. Unfortunately, this problem is very complex, because we are dealing with a large number of possible solutions. For each empty field, there are as many as 9 digit combinations that can be in it. With each subsequent field, the number of possible solutions increases significantly, which causes the time of the algorithm to be extended, and for very large puzzles, the solutions are often wrong.

First fitness function checks the sum of unique digits in each row, column and square. The most favorable value in this case is 243, which is equivalent to the solution of the sudoku.

Second fitness function checks the sum of unique rows, columns and squares. The most favorable value in this case is 27, which is equivalent to the solution of the sudoku.

Results of the first function are much more precise and solutions are found faster.

LINKS

https://en.wikipedia.org/wiki/Sudoku (https://en.wikipedia.org/wiki/Sudoku)

https://pygad.readthedocs.io/en/latest/ (https://pygad.readthedocs.io/en/latest/)