POLI TECH NIKA Czestochowa University of Technology

# EMPLOYEES ALLOCATION PROBLEM

Report project

Soufiane HMIAD

Evolutionary algorithms and search strategies

# 1 CONTENTS

# 2 DESCRIPTION OF THE PROBLEM

Imagine you are responsible for scheduling the shifts for the nurses in your hospital department for this week. There are three shifts in a day – morning, afternoon, and night – and for each shift, you need to assign one or more of the eight nurses that work in your department. If this sounds like a simple task, take a look at the list of relevant hospital rules:

- A nurse is not allowed to work two consecutive shifts.
- A nurse is not allowed to work more than five shifts per week.
- The number of nurses per shift in your department should fall within the following limits:
    - Morning shift: 2–3 nurses
    - Afternoon shift: 2–4 nurses
    - Night shift: 1–2 nurses

In addition, each nurse can have shift preferences. For example, one nurse prefers to only work morning shifts, another nurse prefers to not work afternoon shifts, and so on.

This task is an example of the nurse scheduling problem (NSP), which can have many variants. Possible variations may include different specialties for different nurses, the ability to work on cover shifts (overtime), or even different types of shifts – such as 8-hour shifts and 12-hour shifts.

By now, it probably looks like a good idea to write a program that will do the scheduling for you. Why not apply our knowledge of genetic algorithms to implement such a program?

# 3 SOLUTION REPRESENTATION

For solving the nurse scheduling problem, we decided to use genetic algorithms that can naturally handle this representation.

For each nurse, we can have a binary string representing the 21 shifts of the week. A value of 1 represents a shift that the nurse is scheduled to work on. For example, take a look at the following binary list:

(0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0)

This can be broken into the following groups of three values, representing the shifts this nurse will be working each day of the week:

| **Sunday** | **Monday** | **Tuesday** | **Wednesday** | **Thursday** | **Friday** | **Saturday** |
|---|---|---|---|---|---|---|
| (0, 1, 0) | (1, 0, 1) | (0, 1, 1) | (0, 0, 0) | (0, 0, 1) | (1, 0, 0) | (0, 1, 0) |
| afternoon | morning and night | afternoon and night | (none) | night | morning | afternoon |

The schedules of all nurses can be then concatenated together to create one long binary list representing the entire solution.

When evaluating a solution, this long list can be broken down into the schedules of the individual nurses, and violations of the constraints can be checked for. The preceding sample nurse schedule, for instance, contains two occurrences of consecutive 1 value that represent consecutive shifts being worked (afternoon followed by night, and night followed by morning). The number of weekly shifts for that same nurse can be calculated by totalling the binary values of the list, which results in 8 shifts. We can also easily check for adherence to the shift preferences by checking each day's shifts against the given preferred shifts of that nurse.

Finally, to check for the constraints of the number of nurses per shift, we can sum the weekly schedules of all nurses and look for entries that are larger than the maximum allowed or smaller than the minimum allowed.

In our case, we chose to penalize the violations of the hard constraints to a larger degree than those of the soft constraints. This was done by creating a cost function, where the cost of a hard constraint violation is greater than that of a soft constraint violation. The total cost is then used as the fitness function to be minimized.

# 4 GENETIC ALGORITHMS SOLUTION

The main parts of our solution are described in the following steps:

1. Our program starts by creating an instance of the NurseSchedulingProblem class with the desired value for hardConstraintPenalty, which is set by the HARD_CONSTRAINT_PENALTY constant:

```
nsp = NurseSchedulingProblem(HARD_CONSTRAINT_PENALTY)
```

2. Since our goal is to minimize the cost, we define a single objective, minimizing fitness strategy:

```
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
```

3. Since the solution is represented by a list of 0 or 1 values, we use the following toolbox definitions to create the initial population:

```
toolbox.register("zeroOrOne", random.randint, 0, 1)

toolbox.register("individualCreator", tools.initRepeat, creator.Individual,
toolbox.zeroOrOne, len(nsp))

toolbox.register("populationCreator", tools.initRepeat, list, toolbox.individualCreator)
```

4. The actual fitness function is set to calculate the cost of the various violations in the schedule, represented by each individual solution:

```
def getCost(individual):
    return nsp.getCost(individual),  # return a tuple


toolbox.register("evaluate", getCost)
```

5. As for the genetic operators, we use tournament selection with a tournament size of 2, along with two-point crossover and flip-bit mutation, since this is suitable for binary lists:

```
toolbox.register("select", tools.selTournament, tournsize=2)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=1.0/len(nsp))
```

6. We keep using the elitist approach, where HOF members – the current best individuals – are always passed untouched to the next generation:

```
population, logbook = eaSimpleWithElitism(
    population,
    toolbox,
    cxpb=P_CROSSOVER,
    mutpb=P_MUTATION,
    ngen=MAX_GENERATIONS,
    stats=stats,
    halloffame=hof,
    verbose=True
)
```

7. When the algorithm concludes, we print the details of the best solution that was found:

```
nsp.printScheduleInfo(best)
```

Before we run the program, let's set the algorithm constants, as follows:

```
P_CROSSOVER = 0.9  # probability for crossover
P_MUTATION = 0.1   # probability for mutating an individual
HALL_OF_FAME_SIZE = 30
POPULATION_SIZE = 300
MAX_GENERATIONS = 10
```

In addition, let's start by setting the penalty for violating hard constraints to a value of 10, which makes the cost of violating a hard constraint similar to that of violating a soft constraint:

```
HARD_CONSTRAINT_PENALTY = 10
```

# 5 RESULTS

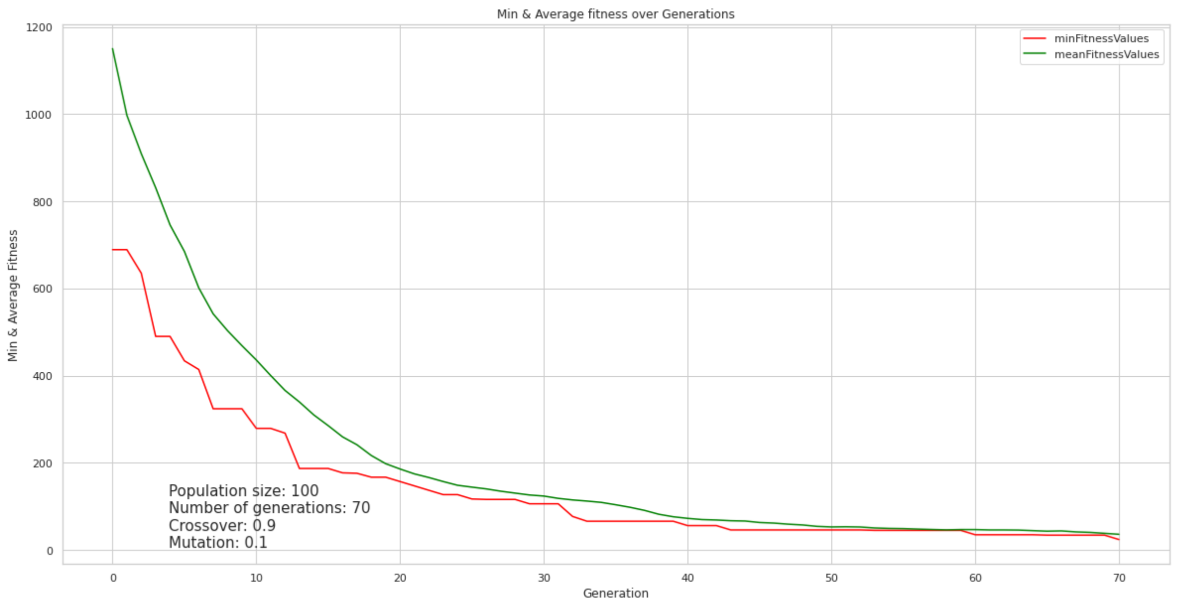

*Figure 1*

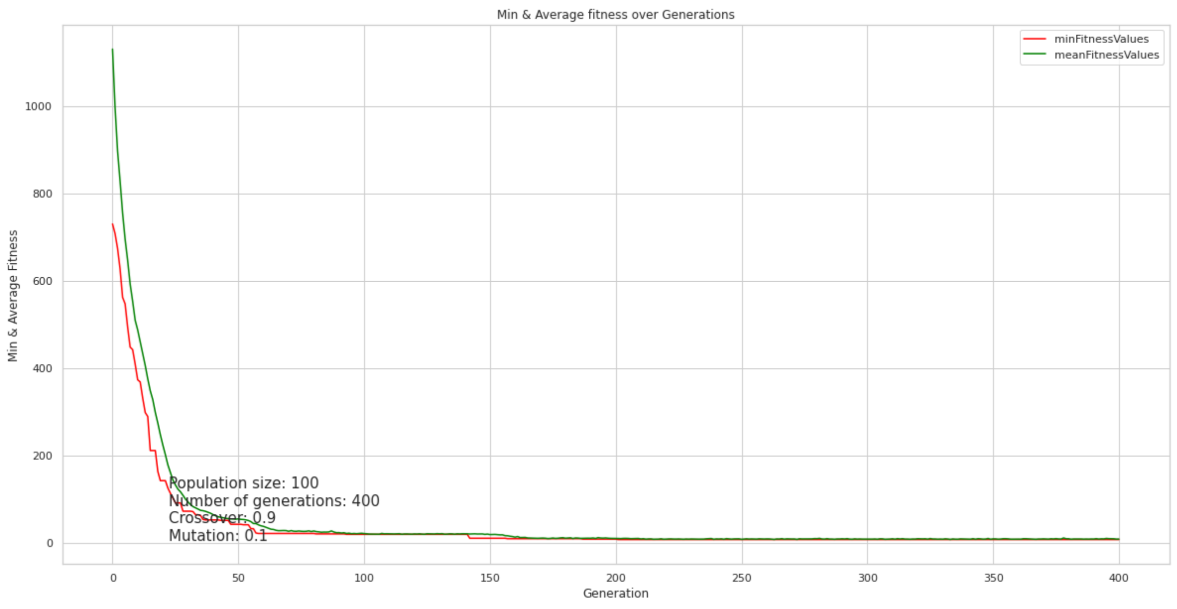

*Figure 2*

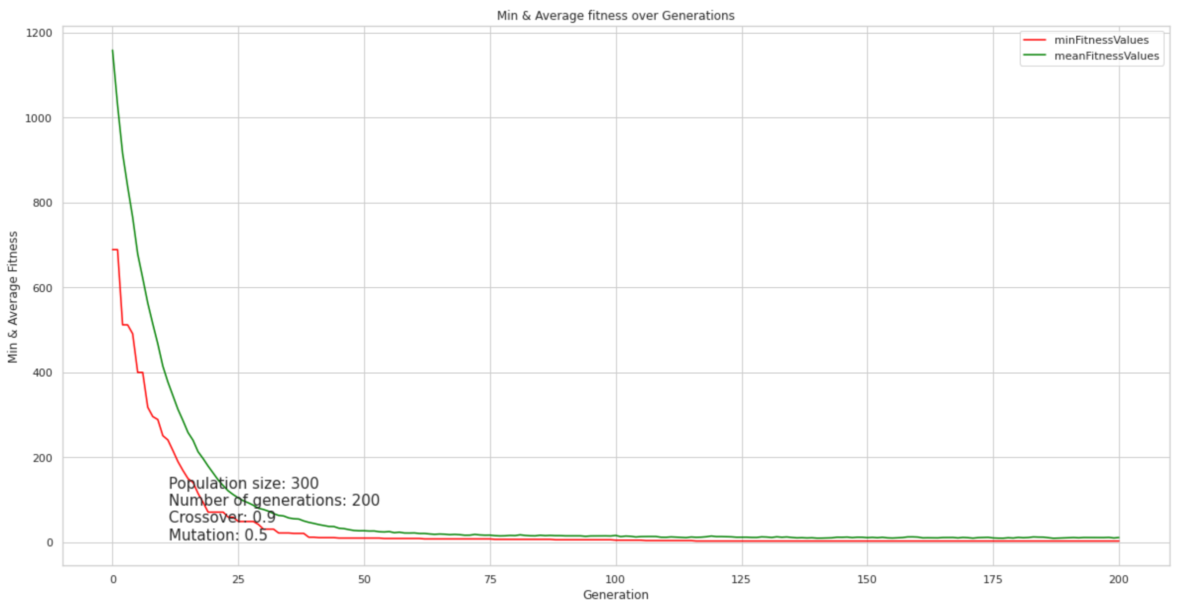

*Figure 3*

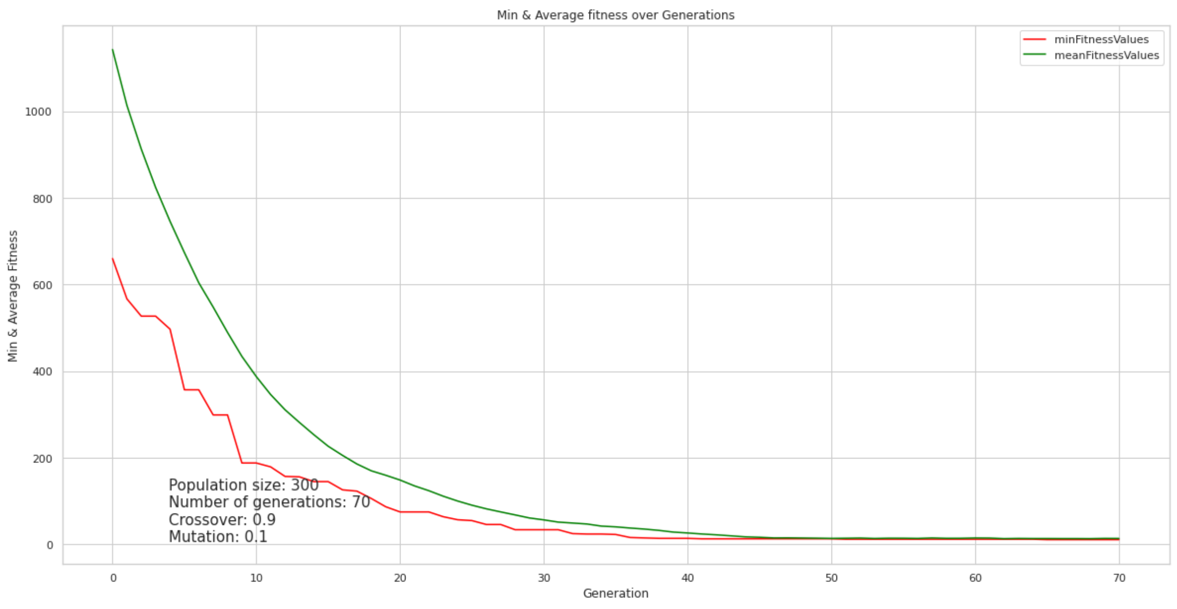

*Figure 4*

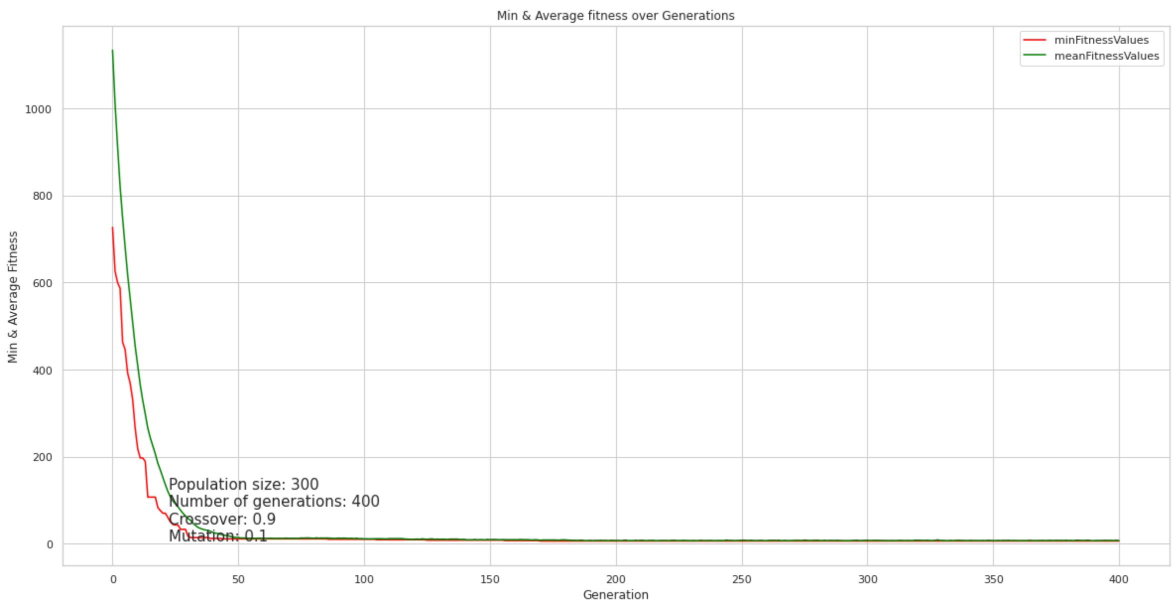

*Figure 5*

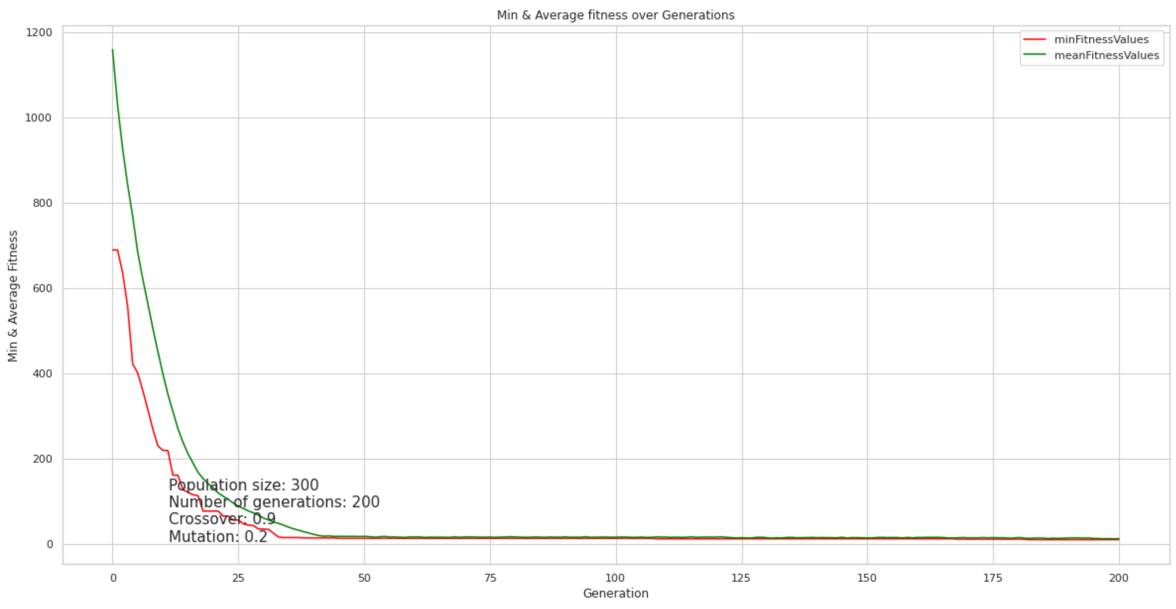

*Figure 6*

As we can tell from these graphs, the number of generations and population size are important for us. Looking at figures 1, 4 we see especially for the number of generations not enough to have better solutions even with a higher crosser-over probability.

Comparing the rest of figures, we always end up with good results but the second issue we have to consider is the execution time. From the experiments we noticed that cross-over probability between 0.7 and 0.9 almost same and lower or higher than that easily we can notice worst results, same goes for mutation, lower is better but has to be more than 0 since its needed for changing in new constant generations.

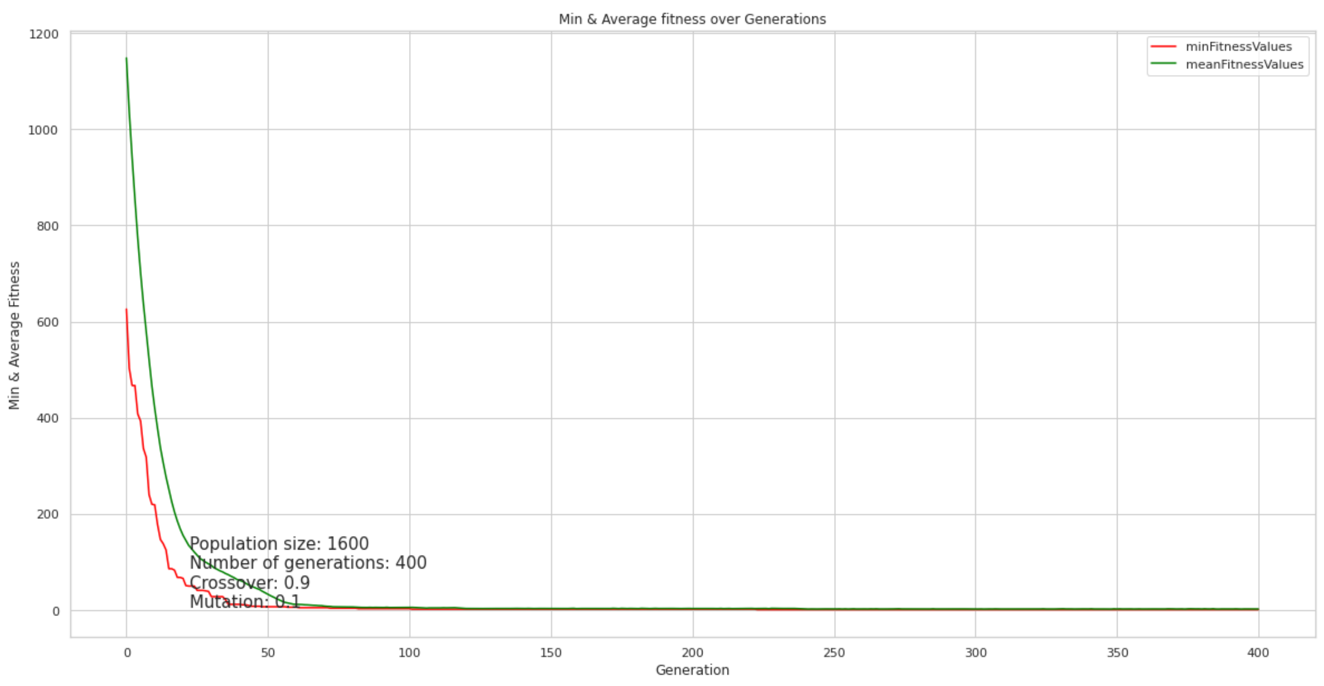

*Figure 7*

| Nurse | | Debra | Riann | Velia | Synne | Anona | Noell | Fadwa | Zyana | Nurses Per Shift |
|---|---|---|---|---|---|---|---|---|---|---|
| **Monday** | morning | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 2 |
| | evening | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 2 |
| | night | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 2 |
| **Tuesday** | morning | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 2 |
| | evening | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 2 |
| | night | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 2 |
| **Wednesday** | morning | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 2 |
| | evening | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 2 |
| | night | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| **Thursday** | morning | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 2 |
| | evening | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 2 |
| | night | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| **Friday** | morning | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 2 |
| | evening | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 2 |
| | night | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| **Saturday** | morning | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| | evening | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 2 |
| | night | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |
| **Sunday** | morning | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| | evening | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 2 |
| | night | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| **weekly Shifts** | | 5 | 5 | 3 | 5 | 5 | 5 | 5 | 4 | 37 |

-- Best Fitness = 1.0
-- Violations:
consecutive shift violations = 0
Shifts Per Week Violations = 0
Nurses Per Shift Violations = 0
Shift Preference Violations = 1

# 6 CONCLUSION

From the last figure 7 and the table of results, we had to sacrifice a small amount of time but still better than higher number of generations in this case, because more generations mean more checking and more time is needed, but with a very high population size and the best parameters for cross-over and mutation we got the best results with lowest violation = 1 for the softest constraint which was shift preference.

Generally, from these graphs we could confirm that population size and generation number were important elements to show big difference in the results, where mostly the algorithm starts to be stable after 40-50 generations even if we pick the worst parameters.

The findings showed that the genetic program created could manage the work schedule of nurses under given conditions and achieve the desired objective. It could create work schedules of nurses that were fair in overtime payment to all the nurses and could reduce the hospital expenses.

As we know, hospitals belong to the service business sector; continuous improvement along with response to customer satisfaction is the key success factor in this business. To reduce the waiting time in every hospital process, high standards in medical care and highly experienced medical doctors are needed.

Inpatient nurses may be allocated to assist when the management wishes to reduce the fluctuation in the number of outpatient nurses. Thus, modern management and highly efficient tools are necessary; this is the reason why GA is involved instead of an adoption of the conventional approach. However, regarding the implementation of simulation, even GA is not familiar to the hospital people. Optimization capacity building is another issue that the hospital management will have to consider.

# 7 References

Hands-On Genetic Algorithms with Python by Eyal Wirsansky

The book explains the logic used and how to approach the problem, using Deap library to optimize the solution.