# Optimal Scheduling

Anastasia Ciaica
*Optimization Class Project. MIPT*

## Introduction

The employee scheduling, originally called the nurse scheduling problem (NSP) is the operations research problem of finding an optimal way to assign employees to shifts. It necessitates the assignment of suitably qualified personnel to particular shifts in order to meet an organization's service demands while observing workplace rules and trying to accommodate individual work preferences. Finding a schedule that satisfies all constraints manually can be much more difficult compared to computer calculations. This problem has been studied since before 1969, and is known to have NP-hard complexity[1].

## Simulated annealing algorithm

One approach is the *simulated annealing*:

$$\text{minimize } \sum_{n=0}^{k} solution\ [n] * matrixPayJob\ [[solution]\ [n]],$$

where:

- $k$ is the scheduling period in shifts.
- $solution = (a_0 \dots a_n)$ (sample solution generated randomly before SA algorithm is applied), $n$ is the shift number and $a_n$ is the employee number.
- $solution\ [n]$ is the employee number on the $n$th shift.

## Genetic algorithm

Another approach is the *genetic algorithm*. The population is comprised of solutions to the problem. The parents are randomly selected solutions, the crossover is a constructed operator on a solution, the mutation is a random change in the solution, and we apply a fitness evaluator to determine which solutions should move on between generations. The basic outline of the GA is given below:

```
Initialize a Population of Individuals
While Stop Criterion not met
    Selection of Individuals to Combine
    Application of Crossover Operator
    Application of Mutation Operator
    Application of Local Search Heuristics
    Evaluation of Fitness of the Newly Created Individuals
    Update Population
Endwhile
```

## Algorithm choice justification

Most studies show that Simulated Annealing is more efficient than Genetic Algorithm as it solves more problems on average, regardless of the random initialization. It was found that there are cases where all the GA weight functions fail to find global minimum (which exists per Simulated Annealing). SA yields a more efficient result than GA since the first one discourages bad neighbors while the other accepts all results to be in next generation.

## Algorithm used for optimization

We solve this problem using Simulated Annealing implementation:

1. Compute an initial solution $X$ and choose an initial temprature $T > 0$ and a repetition factor $L$.
2. As long as the stopping criterion is not satisfied perform the following steps:

**A.** Do the following $L_k$ times
   a. Generate a random solution in the neighborhood $x' \in N(x)$.
   b. Compute $\Delta = f(x') - f(x) + unfit(x')$
   c. Generate a random number $0 < u < 1$
   d. If $\Delta < 0$ or $u < e^{-\frac{\Delta}{T_k}}$ then set $x \leftarrow x'$
   e. If $x$ is feasible and $f(x) < f(x^*)$ then set $x^* \leftarrow x$

**B.** Update $T_k$ and $L_k$ a. $T_{k+1} \leftarrow \alpha * T_k$ b. $L_{k+1} \leftarrow \gamma * L_k$

3. Report the current solution

## Linear algorithm vs. Randomization

There are two approaches to solving the problem:

- Generating a random solution (some kind of work schedule) and then applying the optimization algorithm that will filter out inappropriate solutions.
- Otherwise, a linear algorithm is applied first, which builds a certain valid solution, which is then fed to the input to the Simulated Annealing algorithm. In this case, the linear algorithm looks like this[2]:

```
def canBeFitted(daycount, manNumber,matrixFit,matrixQualityFit):
  if matrixFit[daycount] != 0:
    if (matrixProb[manNumber][daycount] != 0):
      if matrixQualityFit[daycount] == 0:
        if matrixQual[manNumber] == 1:
          return True
        else:
          return False
      else:
        return True
  return False

def checkwillOverwork(manNumber, value,matrixJobFit):
  if (matrixJobFit[manNumber] - value < 0):
    return True
  else:
    return False
```

- matrixFit - the remaining number of employees required for a shift.
- matrixProb - the employee's ability to go to work shift.
- matrixQualityFit - check if an employee with the necessary qualifications has already been assigned to the shift (for example, a head nurse).
- matrixQual - the degree of qualification of the employees.
- matrixJobFit - the number of working hours remaining for each employee before overtime.
- matrixCount - the number of people required per shift.
- matrixMaxJob - the number of maximum working hours of an employee for the allocated period.
- matrixPayJob - the cost of the work shift for each employee.

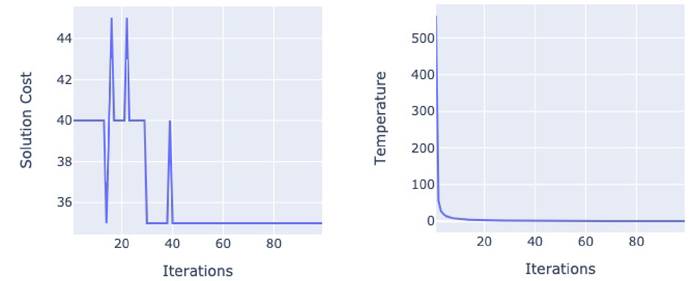
## Sample data set

Consider a sample data set (generated artificially looking back at real data):

```
0 1 1 0 1 1 0 0 0 1 1 1 1 1 1
0 0 1 1 1 1 0 0 0 1 1 1 1 0 1      matrixProb
1 0 1 0 1 0 1 1 0 1 0 1 0 0 0

2 1 2                              matrixCount

0 0 1                              matrixQual

48 32 40                           matrixMaxJob

10 15 20                           matrixPayJob
```

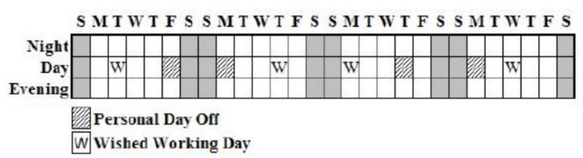
## Results

Over 100 iterations in five cases in a row, the algorithm has converged.




## Conclusion and Further Work

As of now, the linear algorithm already gives a pretty good solution and there is almost no difference after applying the Simulated Annealing algorithm. In this approach, it is easier to add new constraints, and nevertheless, not all options are enumerated (due to the lack of randomization) and all constraints are considered strict. It is planned to implement the random generation of the solution and the subsequent application of the SA algorithm. In the future it is planned to collect initial data in a more user-friendly way, namely from annotated schedule with preferences of the following type:




## References

[1] Tianhao Lu Sijia Zhu Fanying Chen, Shuyao Lu. Nurse scheduling.
[2] Anastasia Ciaica. Optimal scheduling, 2021.