Genetic Algorithms for Workforce Shift Scheduling

Sara Carolina Gómez Delgado
Universidad Panamericana
Campus Aguascalientes, México
0226594@up.edu.mx

Luis Eduardo Robles Jiménez
Universidad Panamericana
Campus Aguascalientes, México
0224969@up.edu.mx

***Abstract*— The shift scheduling problem is one of the most important activities of managers in companies and it is usually a burden since still nowadays, it is hand-made. The objective of this problem is to assign work hours and breaks to the employees of a certain company. This problem's complexity increases as we add restrictions, workers, skills, needs and options to it. To solve it, we have developed in this work four different approaches based on genetic algorithms (GA). The purpose of this paper is to present the best solution capable of solving this problem through evolutionary computation, following the well-known strategy of having several individuals where each one of those propose a solution and given a fitness function, through the pass of generations, the population is intended to get better results since they start embracing the change. Reducing the search space has proven to be an excellent strategy as well in order to improve the probability of getting better results, another important point discussed in the paper is the importance of tuning the algorithm, because even after finding the right strategy, some changes can be made to the reproduction or mutation techniques, or even getting to know better the hyperparameters can lead to improve the exploration of the search space.**

***Keywords*— Genetic Algorithms, Scheduler, Python, Evolutionary Computation, Shift problem, Fitness function, Optimization,**

I. Introduction

The importance of good scheduling in a company is critical, since this determines the working hours of each employee, their days off, lunch hours, etc.

As organizing schedules by hand might take too much time, it is better to create an algorithm that creates it by itself given certain constraints depending on each company.

There are several constraints in this specific company on which we are testing our algorithm such as:

- The week starts on Monday and ends on Sunday.
- Lunchtime lasts two hours on weekdays, on Saturdays an hour and on Sundays half an hour.
- People who study have the Saturday off, so they can attend their classes.
- Preferably, give constant lunch times to people, so the hour doesn't become a burden or leads to disorders.
- Stick to the already defined schedule of the workers' Sundays.
- Breaks:
    - People from the same area must not have the same day or half day off.
    - They have a day and a half off per week.
    - If the area has inventory people belonging to it should not have Wednesday or Thursday off.
    - Whoever has Sunday-off enters at 1 on any other day and still has 1 day-off left.
    - Each employee works two Sundays and has the third one off.

In recent years, various studies have been published focused on trying to solve problems in relation to schedules, such as the job shop scheduling problem which is a difficult problem in combinatorial optimization that consists of assigning several tasks to multiple machines with certain restrictions.

In an earlier work published by Wahyu Sardino and others, the performance of Genetic Algorithms (GA) to solve shift scheduling problems is proven to be better than the others.

II. Methodology

In order to select an algorithm to solve the problem, we first analyzed several Evolutionary Computation and Metaheuristic algorithms.

A. Genetic Algorithms versus others

Evolutionary Programming and Evolution Strategies, both use step size, which defines how much a variable is modified (Figure 1). And as our problem does not require this step size, one and the other were discarded.

$$\underbrace{< x_1, x_2, \dots, x_d,}_{\vec{x}} \quad \underbrace{\sigma_1, \sigma_2, \dots, \sigma_d >}_{\vec{\sigma}}$$

Fig. 1 Step Size (Evolutionary Programming and EvolutionStrategies)

On the other hand, Genetic Programming works with syntax trees and generates formulas. This does not solve our problem since we are generating schedules and not equations or similar.

Talking about Differential evolution, it takes multiple individuals in order to create new ones based on mutations that rely on the nature of the problems it was designed for (optimization of continuous problems) and Particle Swarm Optimization (PSO) does it as well. As this is a discrete optimization problem, these algorithms don't fit our needs.

Finally, Ant Colony would not be the best option in view of the fact that it is represented with graphs and we can not represent our solution this way.

Genetic algorithms are commonly used for search-based optimization problems wherein the idea is to maximize or minimize a given objective function value under several restrictions. This exactly fits in our problem.

B. Input Data Modelling

Genetic Algorithms can be represented in numerous ways such as binary arrays, integer arrays, real arrays and even in a permutation of a set.

Our input data modelling is composed by:

I. *Breaks: An array where each element represents the specific hour or a point of the day (morning, afternoon) when the person's break will occur.*
II. *People: An array where each element represents an employee.*
III. *Areas: An array of arrays (a) where each* a_i *represents the areas where the ith person in the previous described array belongs.*
IV. *Inventory: A value that contains what inventory will be audited in the current week.*

C. Fitness Function

The fitness function describes how good or bad the solution is. In order to know this, what our fitness function does is subtract points when something goes wrong depending on the severity of the "mistake". Otherwise, it adds points when something that's preferable is met.

It is worth mentioning that we are aware that the conditions in these problems are not always met since sometimes the workforce is not enough and there's nothing that can be done about it, so a pseudo-logger has been implemented within the objective function that notifies when a condition is not met. So, when the solution is ideal (which, again, is not always achievable), no comments show up.

D. Applying Genetic Algorithms

A generic implementation of genetic algorithms used in this research works with individuals that are matrix-shaped filled up with discrete data of dimension *[height, width]* and values between *minVal* and *maxVal*.

The algorithm receives the objective function, the arguments of the function, population size, number of generations, reproduction probability, mutation probability and the constraints of the solution that will be generated *(height, width, minVal, maxVal)*.

I. *Initial Population: It's provided to the algorithm and it is previously built based on the already assigned sundays-off, and their fixed time-off people have per week.*
II. *Parent Selection: As recommended, a Roulette selection is used. For this research an inverse Roulette selection was used as well and its objective is to find the worst individual in order to replace it with the elite.*
III. *Crossover: A random set of rows is taken from the second parent and inserted into the first one (uniform crossover).*
IV. *Mutation: A random set of rows is chosen and a random permutation is performed to each one of those. This operation was chosen since that's the one that preserves the amount of days off of each person*

without requiring extra effort to count days or modifying the proposed solution.

Multiple approaches were applied in order to test their performance solving the problem.

A. *First Approach*

The first approach gives the whole problem to the algorithm and checks if it succeeds. The search space was: A 20x7 matrix (people vs days) where each cell could have 15 different values (3 lunch hours for weekdays, 5 for saturdays, 3 for sundays and other 4 different states: day-off, morning-off, afternoon-off, morning-off as an extra for their day off is Sunday). So the resulting search space is 15^(20x7).

B. *Second Approach*

The second approach is a preliminary version of the first one. In this case, the specific lunch hours depending on the day are ignored and only three options are given. The search space was a 20x7 matrix (people vs days) where each cell could have 7 different values (3 lunch hours and other 4 different states: day-off, morning-off, afternoon-off, morning-off as an extra for those whose day off is Sunday). Thus, its search space is 7^(20x7).

C. *Third Approach*

The third approach collapses all the lunch times into a single value. This is intended to reduce the search space even more and help the algorithm to improve its solutions. In this case, we have a search space of a 20x7 matrix (people vs days) where each cell could have five different values (specifies a lunch hour and other 4 different states: day-off, morning-off, afternoon-off, morning-off as an extra for those whose their day off is Sunday). In this case, its search space is 5^(20x7).   .

D. *Fourth Approach*

The fourth approach is divided into three different stages:

I. *Assign days based on whether it's their sunday-off or not, give them their vacation.*
II. *Find different permutations in order to keep as many people in the areas as possible at all times.*
III. *Assign them their lunch time and entry time.*

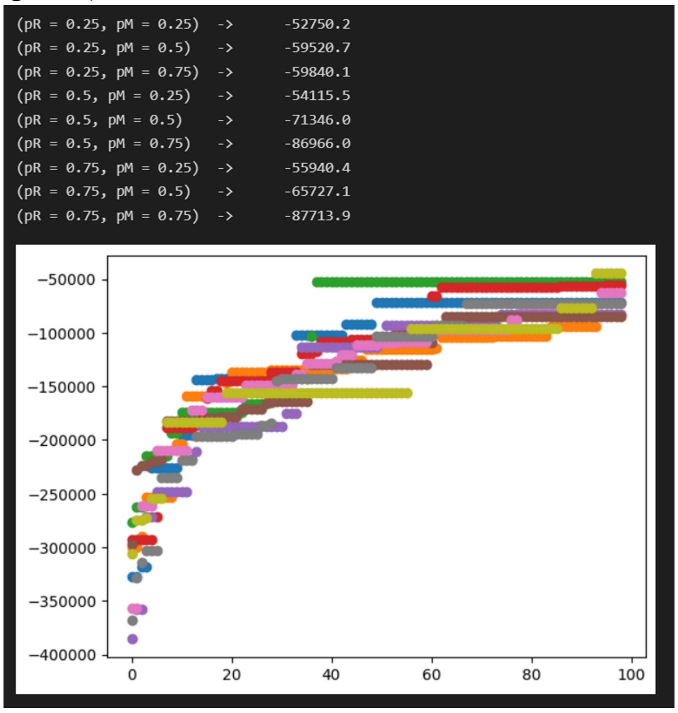
As it's possible to see, the previous approaches were intuitive, but the search spaces were incredibly huge, and the bigger they get, the harder it is to explore them. This is why a new approach was used. Our fourth solution pretends to be a game changer since the size is still enormous but it is not even a 1% of what the smallest of the previous approaches was. This approach takes into consideration that everyone has a fixed amount of days off a week and a fixed schedule of Sundays is already defined, so it makes the task easier for the algorithm. Also, dividing the problem into several sections will lead to better convergences on each stage. Since each row will be already given and we only want to see the permutations of each of them, the search space is defined by *n!^a*, where *n = days of the schedule - 1* and *a = number of people.* In this case, its search space is 6!^20.

III. Results

The last three approaches were tested. The first one was omitted since this solution was the most intuitive, however, the search space is extremely huge (15^140). This makes it very difficult to program in addition to reducing the probability of success of the algorithm.

A. Second approach

In the second approach, we obtained that, by getting the average fitness value of several iterations changing the probability of reproduction and of mutation, it's possible to see that some combination of values give similar results, but in a more general vision, we can notice there are some combinations that can actually be seen as bad (Figure 2).

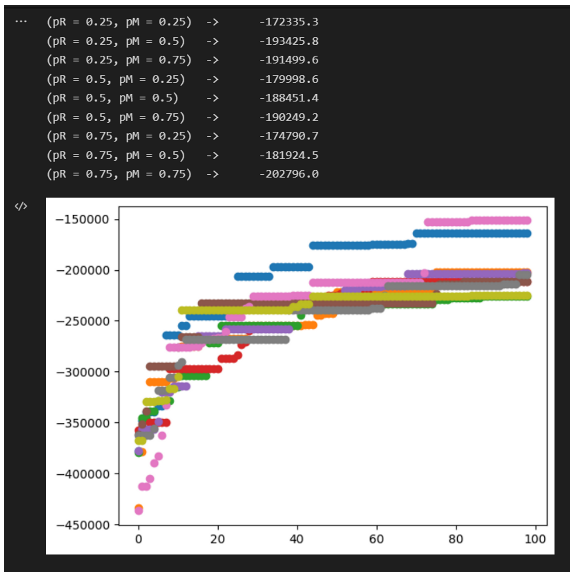


```
(pR = 0.25, pM = 0.25)  ->      -52750.2
(pR = 0.25, pM = 0.5)   ->      -59520.7
(pR = 0.25, pM = 0.75)  ->      -59840.1
(pR = 0.5, pM = 0.25)   ->      -54115.5
(pR = 0.5, pM = 0.5)    ->      -71346.0
(pR = 0.5, pM = 0.75)   ->      -86966.0
(pR = 0.75, pM = 0.25)  ->      -55940.4
(pR = 0.75, pM = 0.5)   ->      -65727.1
(pR = 0.75, pM = 0.75)  ->      -87713.9
```

Fig. 2 Exploring results of the second approach

B. Third approach

In our third approach, the results seem benefitted when choosing a small probability of reproduction and mutation. As we reduced the possibilities of choosing the lunch time, the fitness function is different and it is not affected by overlapped lunch time within the same employee's areas (Figure 3).



```
(pR = 0.25, pM = 0.25)  ->      -172335.3
(pR = 0.25, pM = 0.5)   ->      -193425.8
(pR = 0.25, pM = 0.75)  ->      -191499.6
(pR = 0.5, pM = 0.25)   ->      -179998.6
(pR = 0.5, pM = 0.5)    ->      -188451.4
(pR = 0.5, pM = 0.75)   ->      -190249.2
(pR = 0.75, pM = 0.25)  ->      -174790.7
(pR = 0.75, pM = 0.5)   ->      -181924.5
(pR = 0.75, pM = 0.75)  ->      -202796.0
```

Fig. 3 Result of the third approach

A. Fourth approach

In the fourth (and last) approach, we obtained even better results.

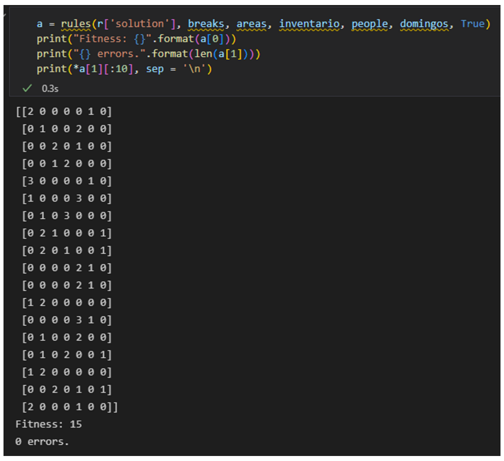
As mentioned previously, this solution was divided in three stages:

I. **First Stage:** This stage consists of assigning the correct set of days that a worker has:
   - If it's their turn to have Sunday off, they still have a day-off and will have a late entry time one day of the week.
   - If it is not their turn to have Sunday off, they have half-day off and a whole day-off.

   As we already have that information, building the set is pretty straightforward. The only stochastic element is assigning a random morning or afternoon to the ones who have half-days off.

II. **Second Stage:** In this stage is where the search space has been abruptly reduced. Also, the fitness function has much less restrictions since it has less things to check now. The way it currently works is, it gives and removes points to the restrictions with different priorities. The restrictions taken into account are:
   - *High Priority:* If the area of a specific person has inventory, a person can neither have Wednesday nor Thursday off.
   - *Medium Priority:* Requests of employees to have a specific day off.
   - *Low Priority:* The most "empty" day of each week by area. This small modification means that, when the above conditions are already fulfilled, if the algorithm has to choose between two solutions, it goes for the one with the best distribution of people and the area that has the largest number of people possible.

The results have been amazing. The objective function has the possibility to show up which conditions are not being met and, in this approach, no errors were shown, this means every condition was met (Figure 4).

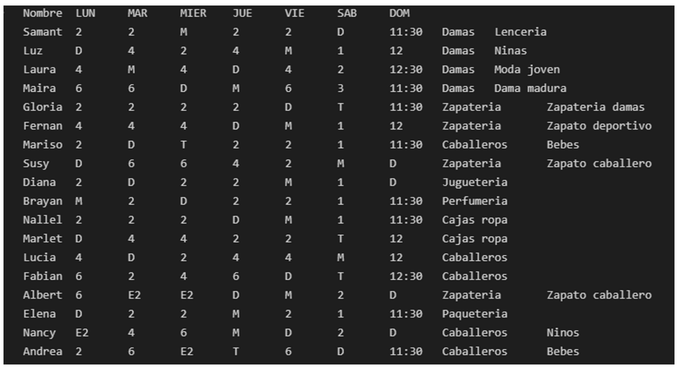


```
a = rules(r['solution'], breaks, areas, inventario, people, domingos, True)
print("Fitness: {}".format(a[0]))
print("{} errors.".format(len(a[1])))
print(*a[1][:10], sep = '\n')
✓ 0.3s

[[2 0 0 0 0 1 0]
 [0 1 0 0 2 0 0]
 [0 0 2 0 1 0 0]
 [0 0 1 2 0 0 0]
 [3 0 0 0 0 1 0]
 [1 0 0 0 3 0 0]
 [0 1 0 3 0 0 0]
 [0 2 1 0 0 0 1]
 [0 2 0 1 0 0 1]
 [0 0 0 0 2 1 0]
 [0 0 0 0 2 1 0]
 [1 2 0 0 0 0 0]
 [0 0 0 0 3 1 0]
 [0 1 0 0 2 0 0]
 [0 1 0 2 0 0 1]
 [1 2 0 0 0 0 0]
 [0 0 2 0 1 0 1]
 [2 0 0 0 1 0 0]]
Fitness: 15
0 errors.
```

Fig. 4 No errors shown up in approach 4.

III. **Third Stage:** Assigning hours is a task that can be achieved in a deterministic way, which is good since it allows us personalizing results as needed. A finished schedule, with meal times assigned looks as follows:



Nombre	LUN	MAR	MIER	JUE	VIE	SAB	DOM		
Samant	2	2	M	2	2	D	11:30	Damas	Lenceria
Luz	D	4	2	4	M	1	12	Damas	Ninas
Laura	4	M	4	D	4	2	12:30	Damas	Moda joven
Maira	6	6	D	M	6	3	11:30	Damas	Dama madura
Gloria	2	2	2	2	D	T	11:30	Zapateria	Zapateria damas
Fernan	4	4	4	D	M	1	12	Zapateria	Zapato deportivo
Mariso	2	D	T	2	2	1	11:30	Caballeros	Bebes
Susy	D	6	6	4	2	M	D	Zapateria	Zapato caballero
Diana	2	D	2	2	M	1	D	Jugueteria	
Brayan	M	2	D	2	2	1	11:30	Perfumeria	
Nallel	2	2	2	D	M	1	11:30	Cajas ropa	
Marlet	D	4	4	2	2	T	12	Cajas ropa	
Lucia	4	D	2	4	4	M	12	Caballeros	
Fabian	6	2	4	6	D	T	12:30	Caballeros	
Albert	6	E2	E2	D	M	2	D	Zapateria	Zapato caballero
Elena	D	2	2	M	2	1	11:30	Paqueteria	
Nancy	E2	4	6	M	D	2	D	Caballeros	Ninos
Andrea	2	6	E2	T	6	D	11:30	Caballeros	Bebes

Fig. 5 Finished Schedule.

Where the numbers are the hours of the employee's meal times. T represents an afternoon off, M a morning off, D day off and E2 is for those who have to come in late. Please note that lunch hours change depending on whether it is Saturday, Sunday or a weekday.

IV. Conclusions

This paper proposed four different approaches to solve the workforce shift scheduling problem that handles numerous constraints.

To evaluate the success of each approach, we applied them to a real company week scheduling. And come to an end that the last approach, in which the search space was reduced considerably, is the best one since the results were abysmally better.

We can finally conclude that, the more the search space is reduced and the more we correctly prioritize and define the objective function, the better solutions we should have.

Taking that into account, we can expect solutions close to or achieving the ideal solutions with a few iterations only, just as it happened with the last approach.

References

[1] Wahyu Sardino *et al*, Genetic Algorithm Implementation for application of shifting work scheduling system, ICIC Express Letters, 2021.

[2] Ning Xue *et al*, A Genetic Algorithm With Composite Chromosome for Shift Assignment of Part-time Employees, 2018 IEEE Congress on Evolutionary Computation (CEC), 2018,

[3] Nysret Musliu, Heuristic Methods for Automatic Rotating Workforce Scheduling, International Journal of Computational Intelligence Research, 2006.

[4] Neeraj Kumar and Abhishek Mishra, Comparative study of different heuristics algorithms in solving classical job shop scheduling problem, Materials Today: Proceedings, 2020.

[5] José Fernando Gonçalves, Jorge José de Magalhães Mendes and Maurício G. C. Resend, A Hybrid Genetic Algorithm for the Job Shop Scheduling Problem, European Journal of Operational Research, 2004, pp.77-95

[6] GeeksforGeeks. (2022, 29 septiembre). Genetic Algorithms. https://www.geeksforgeeks.org/genetic-algorithm

[7] Julio Tanomaru, Solution of a Difficult Workforce Scheduling Problem by a Genetic Algorithm,vol 37 No 8, Faculty of Engineering The University of Tokushima, 1996.

[8] Amr Arisha, Paul Young and Mochie El Baradie, Job Shop Scheduling Problem: an Overview, Dublin Institute of Technology, 2007.

[9] Nagraj Balakrishnan and Richard T. Wong, A network model for the rotating workforce scheduling problem, Networks an International Journal, 1990.

[10] Jonas Volland, Andreas Fügener and Jens O. Brunner, A column generation approach for the integrated shift and task scheduling problem of logistics assistants in hospitals, European Journal of Operational Research, 2017, pp 316-334.