A Multi-objective Evolutionary Algorithms Approach to Optimize a Task Scheduling Problem

Nicolás Cobos, Ixtli Barbosa, Germán A. Montoya, and Carlos Lozano-Garzon(✉)

Systems and Computer Engineering Department,
Universidad de los Andes, Bogotá, Colombia
{n.cobos,iy.barbosa,ga.montoya44,calozanog}@uniandes.edu.co

Abstract. Nowadays, the size of the problems to be solved in the business world has increased largely; since companies have more resources and more demand for products and services from customers. As a result, different meta-heuristics have been developed in the computing world with the aim of finding an optimal solution in a shorter runtime. Involving a real-life case, this paper will present the approach of a multi-objective task scheduling model, solved with evolutionary algorithms; specifically, NSGA-II and SPEA2. In addition, a mathematical model was proposed and its solution was calculated in order to obtain results that allow us to compare the accuracy of the results obtained by the proposed algorithms. The running time and total cost of the task scheduling were the metrics for the evaluation of the results. Between the evolutionary algorithms, NSGA-II obtained the best results in both metrics.

Keywords: Multi-Objective Evolutionary Algorithms · Multi-objective optimization · NSGA-II · SPEA2 · Task scheduling

1 Introduction

The problem of resource allocation is referred to as: "the problem that seeks to find the optimal allocation of a fixed amount of resources to a certain number of activities in such a way that it minimizes the cost generated by the allocation" [1]. This problem is often implemented with a *minimax* in the objective function; this kind of function has been studied a lot for its simplicity and versatility since it can be adapted to be used to represent a large number of problems. An example of its extension and generalization is the relationship with the high-tech industries; in these companies, this problem has been connected with the production planning to maximize their efficiency [1].

The most basic formulation of the resource allocation problem according to Handbook of Combinatorial Optimization [1] is presented in the set of Eqs. 1.

$$\begin{aligned}
&minimize(maximize_{1 \leq j \leq n}(f_j(x_j))\\
&\text{subject to:}\\
&\sum_{j=1}^{n} x_j = N,\\
&x_j \geq 0, \forall j
\end{aligned} \tag{1}$$

Regarding the use of evolutionary algorithms to solve the resource allocation problem, it can be said that it is a topic from the 2010s. One of the most relevant paper was written by Xia and Shen [2], in this paper the authors addressed the allocation of resources in a cloud computing system in order to maximize the utility that a company receives from each user it serves and to satisfy the required level of service (latency, availability, etc.). To solve this, they used three types of algorithms: a genetic algorithm (GA), an ant colony algorithm with a genetic algorithm (ACO-GA), and a quantum genetic algorithm (QGA). Furthermore, a mapping process was used between the resource allocation matrix and the chromosomes of each algorithm, searching for pairs of resources based on the availability matrices of ACO-GA, and coding the differences in values between the resources used and the minimum resource required by QGA. With extensive simulation, the authors proved that evolutionary algorithms, in this case the quantum one, have a better performance than other alternative solutions such as dynamic programming or other meta-heuristics.

There is a similar work to that carried out in this project, the authors combine an evolutionary algorithm with a greedy algorithm to make a task allocation for multi-agent systems [3]. Authors argue that the evolutionary algorithm (in this case D-NSGA III) is specifically used to optimize different targets simultaneously, thus ensuring diversity of responses and search capacity; and the combination of the evolutionary algorithm with the greedy serves to improve the ability to search for local optimums.

This paper is based on a real problem where there is a number of tasks that need to be fulfilled by agents on a business day. So, in order to do that, the scheduling must assign the work orders to the agents so they can complete the activities. Constraints that were taken into account involve the available hours that a worker has on a certain day, the abilities that the worker has, and the ones that a certain work order requires. At the same time, two objective functions are set to be optimized: minimize the total cost of scheduling and minimize the maximum amount of orders that a single agent completes in a day.

The first step to solve this problem was to propose a mathematical model. Then, that mathematical model was translated into an optimization model in Java and Python; the next step was to implement the different solving methods. The linear solving method was implemented in Python with the help of Pyomo [4], which is a collection of Python software packages to formulate optimization models. Meanwhile, the evolutionary algorithms were implemented in Java using JMetal [5], which is an object-oriented Java-based framework for multi-objective optimization with meta-heuristics.

The main goal of this work was to establish which of the solving methods is optimal. For this, it was proposed that the metrics to compare were the results of the objective functions (total cost of scheduling, the maximum amount of work orders completed by an agent in a day), and the time it takes the algorithm to get to the final solution.

The remainder of this paper is organized as follows. In Sect. 2, the general problem will be stated. In the third one, the mathematical optimization model approach will be explained. In Sect. 4, there will be an explanation of the implementation of the mathematical model in Python and Java. And finally, we will discuss the results and future work of the project.

2 General Problem Statement

The problem used in this paper exemplifies the actual condition of some Colombian company that needs a making decision app that supports the day by day agent task scheduling. Each task is required by one company client, and it is necessary to specify at least three parameters: a specific limited time to attend the task, the required skills to do the activity, and the location where the activity is going to take place. In relation to the first parameter, if the time limit is surpassed and the activity has not been attended this will represent a monetary cost for the company because they will break the Service Level Agreement (SLA) previously covenanted with the client.

The agents that solve the activities are company employees, and we need to have some characteristics defined: they have a skill set, a defined working time, an hourly wage defined by the company, and a starting location in the city. It is important to remark that some agents need to go to the central office before they start with the tasks, but some others can just go directly from their house to the location of the activity.

The company defined five optimization attributes:

– Cost: this attribute is related to the agent's salary; the company wants to use the agents with the highest hourly wage in the activities that really need them to be solved. This saves the company operation costs because they will be spending less money on payroll.
– Distances traveled by the agents: this attribute, does not need a deeper explanation; the company expects that an agent does not have to go across the whole cite to take care of a task if there is another one available and closer.
– Fairness of activities assigned to the agents: the third attribute, is related to giving an equal amount of work to all the agents. Sometimes, occurs that some agents have to work all day without rest meanwhile some others just have one task scheduled. In order to promote equality, the company wants the difference in allocation between agents to be as small as possible.
– Matching abilities between the task and the agent: This attribute, maximize the number of matching skills between the task and the agent assigned. This can be confusing because it will make no sense to assign someone how does

not have the skills to an activity; but in some cases, this is happening. In an ideal world, the company will hire as many people as needed; but in the real-world, the resources are limited so they have to do what they can; for this reason, it is common to see employees unqualified in certain activities. Although this is planed, it is wanted to happen with the minimum possible frequency.

- Number of orders solved before the limit time: The last attribute, maximize the number of orders solved before the limit time of each task, according to the SLA. The company wants to fulfill their responsibilities as much as possible.

3 Mathematical Optimization Model Approach

The first step was to determine the sets in the problem. For this, the files that the company had given us were reviewed; from them, it was deduced that the problem could be represented taking into account only this four attributes: orders, employees, hours, and skills. To be more realistic with the number of tasks that can be done in one day, we decide to add an estimate of the transportation time between it.

The sets, and parameters required by our mathematical model are described in the Table 1.

Table 1. Notations of the proposed model.

Sets	Description
E	Set of employees
O	Set of orders
H	Set of working time hours available
S	Set of skills
Parameters	Description
B_{es}	Binary parameter that represents if an employee $e \in E$ has the skill $s \in S$
C_{os}	Binary parameter that represents if an order $o \in O$ require the skill $s \in S$
D_{eh}	Binary parameter that represents if an employee $e \in E$ has availability at time $h \in H$
F_e	Integer parameter that represents the hourly wage of the employee $e \in E$

For this multi-objective proposal, we defined the decision variable as the relation between the employees, orders and hours. This relation is represented through the binary variable X_{eoh} which takes the value of 1 if the employee $e \in E$ goes to the activity $o \in O$ at the hour $h \in H$ (see Eq. 2).

$$X_{eoh} = \begin{cases} 1 \text{ employee } e \in E \text{ goes to the activity } o \in O \text{ at the hour } h \in H \\ 0 \text{ otherwise} \end{cases} \quad (2)$$

3.1 The Objective Function

Considering the main needs for the decision making app, we select only the cost and fairness attributes as part of the main function.

For the cost, we minimize the sum of all the employees' hourly wage multiplied by the sum of all the hours and all the tasks in X_{eoh} as you can see in Eq. 3.

$$min \sum_{e \in E} F_e \sum_{h \in H} \sum_{o \in O} X_{eoh} \tag{3}$$

For fairness, we decided to minimize the variable P, this variable represents the number of agent tasks for the agent with the most assigned activities (see Eq. 4).

$$\begin{aligned} &min \quad P \\ &\text{where: } P = max(\sum_{h \in H} \sum_{o \in O} X_{eoh}) \qquad \forall e \in E \end{aligned} \tag{4}$$

3.2 Model Constraints

In order to fulfill the initial optimization requirements, the attributes that were not used for the objective function were modeled as constraints.

The first constraint is related to the fact that an order can only be assigned to one employee at a specific time (see Eq. 5).

$$\sum_{h \in H} \sum_{e \in E} X_{eoh} = 1 \qquad \forall o \in O \tag{5}$$

The next constraint is close related with the previous one, it is desirable that all the orders are attends at the working day, as you can see in Eq. 6.

$$\sum_{o \in O} X_{eoh} \leq 1 \qquad \forall e \in E \quad \forall h \in H \tag{6}$$

Our third constraint ensures that an agent e can only be assigned to an order o at a specific hour h if he has the available time to fill it (see Eq. 7).

$$D_{eh} \geq X_{eoh} \qquad \forall e \in E \quad \forall h \in H \quad \forall o \in O \tag{7}$$

And the final constraint seeks that an employee e must have at least the same skills that the order o requires, as you can see in Eq. 8.

$$B_{es} X_{eoh} \geq C_{os} X_{eoh} \qquad \forall e \in E \quad \forall h \in H \quad \forall o \in O \quad \forall s \in S \tag{8}$$

4 Implementation

As previously mentioned, based on the data files provided by the company, we established some assumptions as: a regular day involved 10 working hours, and there were 2 skills that the task needed and/or the workers might have. Also, from the files, we got that a task lasts approximately 15 min.

After having worked with different optimization frameworks, the ones that provided what we required were Pyomo and JMetal. Pyomo is a Python-based, open-source optimization modeling language with a diverse set of optimization capabilities [6], and JMetal is an optimization framework based on Java [7].

4.1 Meta-heuristics Implementation

Taking into account that the company wants to find a set of optimal solutions, we select the use of Multi-Objective Evolutionary Algorithms (MOEA) because they allow us to find the Pareto front in a single run [8]. Specifically, algorithms Non-dominated Sorting Genetic Algorithm (NSGA-II) and Strength Pareto Evolutionary Algorithm (SPEA-II) were implemented.

NSGA-II and SPEA2. In this section, we show the original pseudocodes for NSGA-II [9] and SPEA2 [10]. These pseudocodes were the basis to solve our resource allocation problem, which are presented in the Algorithms 1 and 2.

Algorithm 1. NSGA-II Pseudocode.

```
1: $Initialize\ P$
2: $P' = Non \cdot dominated \cdot sort(P)$
3: $StopRunning = false$
4: while StopRunning is false do
5:     $Generate\ F\ fronts\ from\ P'$
6:     $Apply\ Crossover\ and\ Mutation\ to\ F$
7:     $D = selection(F)$
8:     $N = combine(F, D)$
9:     $P' = Non \cdot dominated \cdot sort(N)$
10: end while
11: $return\ P'$
```

The pseudocode for the Non-dominated Sorting Genetic Algorithm (NSGA-II) (Algorithm 1) is described in more detail in the following items:

- A population P is initialized, which corresponds to a set of possible solutions of our problem (line 1).
- The previous solutions are ordered (in a non-dominated manner) and classified in different fronts F (lines 2 and 5).
- Crossover and Mutation are applied to the best fronts F (line 6).

Algorithm 2. SPEA2 Pseudocode.

```
 1: Set $G = numberOfGenerations$
 2: Initialize $P$
 3: $S = \emptyset$
 4: for 1 to G do
 5:     Apply Fitness to $P$ and $S$
 6:     Define $S$ with feasible solutions
 7:     Add non dominated solutions from $P$ and $S$ to $S$
 8:     Apply Truncation to $S$ if its capacity is exceeded
 9:     Apply Tournament Selection to $S$
10:     Apply Crossover and Mutation to $S$
11: end for
12: return $S$
```

- Once the fronts F are selected, a new population D is created (line 7).
- F and D populations are combined to create a new population N (line 8).
- The previous feasible solutions of N are ordered in a non-dominated manner (line 9).
- A crowd-sorting process is performed to the best front's solutions (line 9).
- A new population is created (line 9).
- If a stop-condition was achieved, we show the final population obtained. Otherwise, we use the previous population to recalculate again the algorithm (lines 4 and 11).

Likewise, the pseudocode for the Strength Pareto Evolutionary Algorithm (SPEA-II) (Algorithm 2) is detailed in the following items:

- A number of generations is established. Notice that a high value of this number allows us to find better solutions (line 1).
- A population P is initialized, which corresponds to a set of possible solutions of our problem (line 2).
- An empty set S is created. This set will be composed of non-dominated solutions (line 3).
- Fitness of P and S is calculated, and the population S is established according to the objective functions and constraints of our problem (lines 5 and 6).
- Non-dominated solutions from P and S are added to S (line 7).
- If the size of S exceeds a determined value, the Truncation operator is applied for deleting repeated solutions (line 8).
- The Tournament Selection process is applied to S in order to select just one individual from a comparison of two random individuals according to their fitness (line 9).
- Crossover and Mutation are applied to S to obtain more variations of the solutions (line 10).
- If the limit of iterations has not been exceeded, we proceed to do another "for" iteration. Otherwise, we show the final population obtained (lines 4 and 12).

It is important to remark that, given the framework limitations the decision variable, for the NSGA-II, were handled as integers, with an inferior limit of 0 and a superior limit of 1; and, for the SPEA2 was established to be binary.

Encoding Chromosome Definition. An evolutionary algorithm can generate many solutions, at which each solution is called *Individual*, and a collection of individuals is called *Population*. Each individual is established using a template called *Chromosome*. In essence, a chromosome encodes information related to decision variables or parameters, and, also, its structure defines the way the fitness function is calculated. A chromosome can be either numerical, binary, symbols, or characters depending on the problem [11].

In order to implement individuals based on our chromosome in JMetal, we need to make a minor adjustment to our decision variable. As we observed in Eq. 2, our decision variable X_{eoh}, depends on three sets; which is why it should be modeled as a three-dimensional array (see Fig. 1). Unfortunately, the selected framework only allows the implementation of chromosomes as a one-dimensional arrays. Therefore, we proposed a "translation" in order to comply with the framework requirements (see Fig. 2).

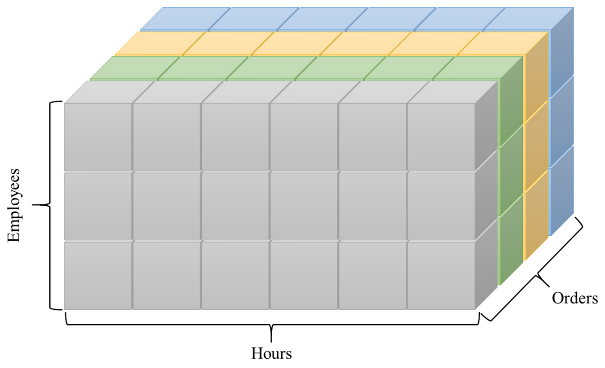

Fig. 1. Original chromosome proposed based on the mathematical model

Taking into account the previous consideration, the chromosome developed by us will be briefly explained. A solution obtained by our evolutionary algorithm is equivalent to the values of each decision variable previously mentioned in the mathematical model formulation. Since the decision variables of this problem are binary, a solution (individual) will be an array of binary elements. Due to there are a high amount of decision variables involved in a relatively big scenario, if the individuals are generated randomly, the computer would take an excessive amount of time to find a feasible solution. To solve this problem, some modifications were applied to the function that creates individuals. The most important aspect of these modifications consists of verifying that each work order was solved in the day only once.

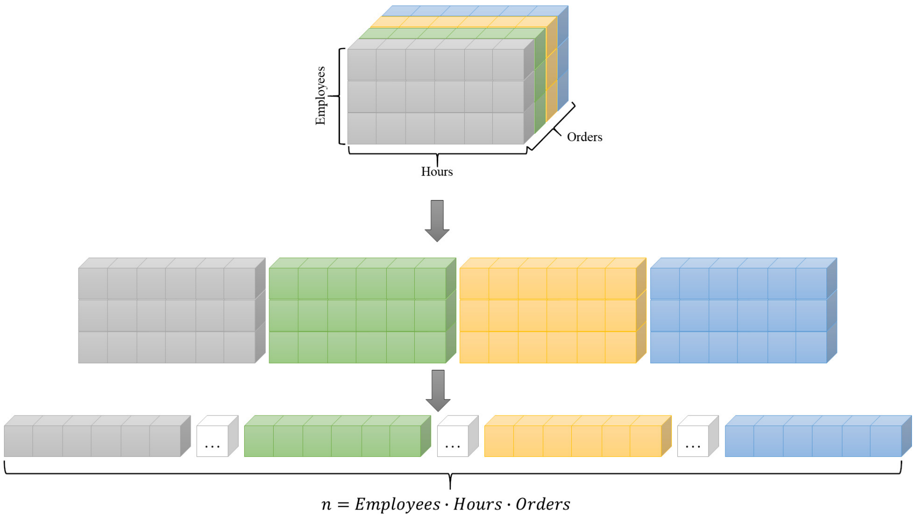

Fig. 2. Final chromosome proposed

The previous modification is explained as follows. First, imagine a scenario with 5 work orders, 4 work hours and 3 agents to fulfill that order. The number of decision variables in this case would be 60 because there is one variable for each combination of the three elements. Each order would be associated to 12 decision variables, involving the combination of each of the hours and workers. To ensure the condition previously mentioned, the process of creating an individual started with a cycle. For each of the work orders the following procedure was executed: choose a random number between 0 and the number of decision variables associated with each order minus one (in this case: $12 - 1 = 11$). Then, you assign that order in that specific index (in terms of the problem, this means to place a 1 in the index and leave the rest of the variables with 0). This ensures that each one of the work orders will be fulfilled just once. After this procedure, maybe the individual created implicates that the employee attends two different work orders in the same work hour, which must be avoided. To solve this problem, each time an order is assigned to the worker, there is a previous checking process to guarantee that there is no other order assigned in that hour.

Generation of Initial Population. It is important to highlight that we modified the original creation process implemented in JMetal. In the framework, the chromosome creation is completely random; when we testing our proposal with this implementation, a high execution time was presented to find a feasible solution. Therefore, based on [3], we decided to modify this process in order to create our chromosomes as a feasible solutions, allowing the algorithm to find solutions faster. This technique improved dramatically the execution time of the algorithm. Finally, we define 100 chromosomes as the initial population size; this size was selected by trial and error.

Genetic Operators. The next step was the selection and configuration of the genetic operators (selection, crossover, and mutation) to be used in our implementation.

For the selection operator we used a binary tournament operator in order to create the mating pool. Specifically, Binary Tournament consists of selecting randomly two individuals. After, these two individuals are evaluated according to its fitness values in order to select the one with the best fitness [12].

Related to the crossover operator, it was selected the single cross point operator with a probability value of 0.9. The single cross point method consists of splitting in two parts the two individuals selected by the binary tournament method, and then, combining these parts to generate a new individual [11].

Finally, for the mutation operator was configured the bit flip mutation function, which is independently applied to each bit in a solution and changes the value of the bit [13]; the probability assigned to this operator was 1 divided by the total number of variables ($p = \frac{1}{|E| \cdot |H| \cdot |O|}$).

Completion Criteria. Based on the experimentation carried out, it was defined that the completion criteria is the number of generations; 250 generations were specifically defined since any increase in this value did not generate improvements to the solution found. In other words, 250 generations were enough to obtain the best results of our specific problem.

4.2 Mathematical Model Implementation

On the other hand, with the Pyomo framework, the implementation followed the three-dimension matrix proposed in the mathematical model so there was no issue with the index use.

The real issue with Pyomo occurred when we tried to implement the fairness optimization attribute because we were not allowed to do the min-max optimization. Our solution propose was to make an adaptation to the model inspired by the ϵ-constrain technique. Basically, this technique allows us to optimize an objective function as a constraint in cases where is difficult to implement all objective functions. In other words, an objective function is included as a constraint in the model, but at the expense of not being included as an objective function anymore. More details of this technique are described in [14]. In this sense, we defined a new constraint that set a maximum limit to the number of orders that could be assigned to an employee. In order to see the model behavior with different maximum limits, we configured the model to run several times changing the limit between the values that the other methods suggested.

The other parts of the model were implemented just as shown in the mathematical model.

5 Experimental Results

With the aim of verifying the correct performance of the model and the proposed algorithms, we defined two main scenarios according to the typical situations of order scheduling on the enterprise. The first one is made up by 150 orders, 30 employees and 10 working hours; whiles for the second scenario have been configured 75 orders, 15 employees and also 10 working hours. Also was defined the following assumptions:

- Each employee has a different hourly wage.
- 2 possible activities and employee skills.
- Not all employees possess all the skills and not all orders require all skills.

5.1 Experimental Results for the First Proposed Scenario

The proposed algorithms finds a feasible solution to the task scheduling problem set up; the Pareto Front obtained is shown in Fig. 3 and the values are presented in Table 2. As expected, the mathematical model presented the best Pareto front since mathematical optimization methods guarantee to find optimal solutions, while meta-heuristic optimization methods cannot guarantee that; for this reason, NSGA-II and SPEA2 Pareto Fronts are not equal to Pyomo's Pareto Front. However, NSGA-II Pareto Front is closer to Pyomo's Pareto Front than SPEA2 Pareto Front, whereby NSGA-II obtained better feasible solutions than SPEA2.

Comparing the execution times of the algorithms deployed, it was found that NSGA-II executed in around 11 s, SPEA2 did it in 24.1 s and the linear optimization algorithm that Pyomo executes in 330 s to obtain the solutions. Finally, in terms of execution time, NSGA-II was the fastest approach.

Table 2. First scenario results

P value	NSGA-II cost	SPEA2 cost	Pyomo cost (ϵ-constraint)
6	\$ 13.590.000,00	–	\$ 12.000.000,00
7	\$ 12.960.000,00	\$ 13.365.000,00	\$ 10.675.000,00
8	\$ 12.525.000,00	\$ 13.290.000,00	\$ 13.675.000,00
9	–	\$ 13.210.000,00	\$ 13.675.000,00

5.2 Experimental Results for the Second Proposed Scenario

For the second scenario, the Pareto fronts of the three methods are presented in Fig. 4 and the values are shown in Table 3. As similar to the first scenario, the mathematical model presented the best Pareto front, and also, NSGA-II Pareto Front is closer to Pyomo's Pareto Front than SPEA2 Pareto Front, whereby NSGA-II obtained again better feasible solutions than SPEA2.

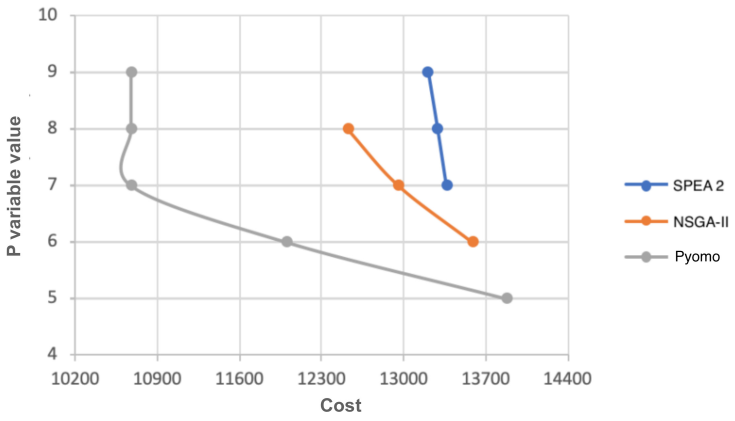

Fig. 3. Fist scenario Pareto fronts.

The execution times of our algorithms are the following, for the NSGA-II was around 3 s, SPEA2 did it in 7 s and the Pyomo optimization runs took 2500 s.

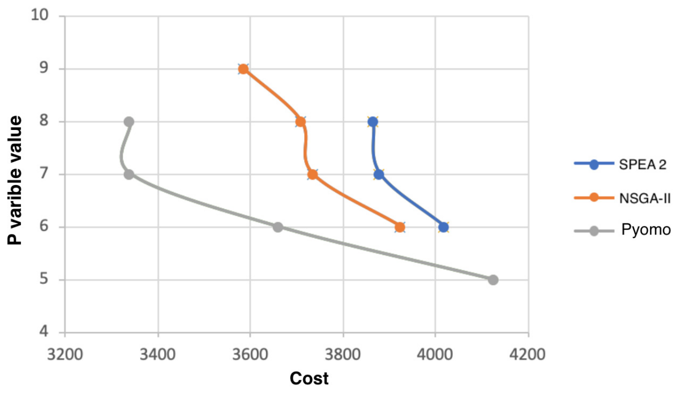

Fig. 4. Second scenario Pareto fronts.

Table 3. Second scenario results

P value	NSGA-II cost	SPEA2 cost	Pyomo cost (ϵ-constraint)
6	\$ 3.925.000,00	\$ 4.020.000,00	\$ 3.660.000,00
7	\$ 3.735.000,00	\$ 3.880.000,00	\$ 3.340.000,00
8	\$ 3.710.000,00	\$ 3.865.000,00	\$ 3.340.000,00
9	\$ 3.585.000,00	–	\$ 3.340.000,00

6 Conclusions and Future Works

In this work, we presented a mathematical optimization model and meta-heuristic approaches for a particular task scheduling problem. Specifically, evolutionary algorithms such as NSGA-II and SPEA2 were implemented and adapted to our problem in order to obtain feasible solutions. In this sense, several components such as chromosome, crossover, and mutation methods were presented. The NSGA-II and SPEA2 results were compared against the optimal results offered by the mathematical optimization model.

As expected, the best Pareto front was obtained by the mathematical optimization model (Pyomo implementation), and the second and third places were for NSGA-II and SPEA2, respectively. In terms of execution time, NSGA-II was the fastest in the evaluated scenarios, while SPEA2 and the mathematical optimization model obtained the second and last place, respectively. That is, mathematical optimization model always obtained the worst execution times. In this sense, if a company requires optimal solutions without execution time requirements, we recommend to use a mathematical optimization approach. However, if a company needs to obtain solutions as soon as possible, a mathematical optimization approach is not suitable, whereby it is recommended to use a meta-heuristic. Therefore, for our particular problem, we suggest to use the NSGA-II evolutionary algorithm.

According to future works, we propose to complement this work by adding the following capabilities:

- Offer path solutions for each employee in order to accomplish the orders taking into account time requirements and other limitations.
- Obtain resource allocation solutions for many working days, instead of one day.
- Test our scenarios by implementing more meta-heuristics such as Particle Swarm Optimization and Ant Colony.
- For large amounts of data, offer solutions by implementing parallelized methods.

References

1. Katoh, N., Shioura, A., Ibaraki, T.: Resource allocation problems. In: Pardalos, P., Du, D.Z., Graham, R. (eds.) Handbook of Combinatorial Optimization. Springer, New York (2013). https://doi.org/10.1007/978-1-4613-0303-9_14
2. Xia, W., Shen, L.: Joint resource allocation using evolutionary algorithms in heterogeneous mobile cloud computing networks. China Commun. **15**(8), 189–204 (2018)
3. Zhou, J., Zhao, X., Zhang, X., Zhao, D., Li, H.: Task allocation for multi-agent systems based on distributed many-objective evolutionary algorithm and greedy algorithm. IEEE Access **8**, 19306–19318 (2020)
4. Center for Computing Research at Sandia National Laboratories: Pyomo. http://www.pyomo.org/

5. Khaos Investigación. JMetal. https://jmetal.github.io/jMetal/
6. Hart, W.E., et al.: Pyomo-Optimization Modeling in Python, 2nd edn. Springer, Heidelberg (2017). https://doi.org/10.1007/978-3-319-58821-6
7. Durillo, J., Nebro, A.: JMetal: a Java framework for multi-objective optimization. Adv. Eng. Softw. **42**(10), 760–771 (2011)
8. Deb, K.: Multi-Objective Optimization using Evolutionary Algorithms. Wiley-Interscience Series in Systems and Optimization. Wiley, West Sussex (2001)
9. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
10. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: improving the strength pareto evolutionary algorithm. Institut für Technische Informatik und Kommunikationsnetze (TIK) **103**, 5–6 (2001)
11. Tan, K.C., Khor, E.F., Lee, T.H.: Multiobjective Evolutionary Algorithms and Applications. Advanced Information and Knowledge Processing Series. Springer, Heidelberg (2004). https://doi.org/10.1007/1-84628-132-6
12. Rahman, R., Ramli, R., Jamari, Z., Ku-Mahamud, K.: Evolutionary Algorithm with Roulette-Tournament Selection for Solving Aquaculture Diet Formulation. Hindawi Publishing Corporation, London (2016)
13. Chicano, F., Sutton, A., Whitley, L., Alba, E.: Fitness probability distribution of bit-flip mutation. Evol. Comput. **23**(2), 217–248 (2014)
14. Mavrotas, G.: Effective implementation of the e-constraint method in multi-objective mathematical programming problems. Appl. Math. Comput. **213**, 455–465 (2009)