Non-Thesis M.Sc. Project Report by

Asım Zorlu

Submitted to the

Graduate School of Engineering

In Partial Fulfillment of the Requirements for the Degree of

Master of Science

in the

Department of Industrial Engineering

Özyeğin University

January 2021

Copyright

* 2021 by Asım Zorlu

Approved by:

Dr. Zehra Melis Teksan, Advisor Department of Industrial Engineering

*Özyeğin University*

Date Approved: 11 March 2013

# ABSTRACT

In this project, we studied an efficient way of task assignment to employees for software testing group. When a new task comes, it was difficult to predict which individuals in the group would finish in how much time and with how much efficiency. For this, test cases were determined by the expert group related to the task and the test suite was written. The testers will complete the task according to the test steps here and return the result. Thus, in line with the criticality of bugs entered into the system, the right people will be identified according to the scores they receive from the relevant assessment.

According to the characteristics of the people in the test group, a proposal will be made according to the features of the people in the test group to determine the new employees to be hired. Rather than having each new applicant apply this task one by one, the system will offer us the ability to do this job according to the features of the existing workers.

According to the constraints entered into the system, appropriate task assignments will be made to the workers using a multi-objective optimization model. The objective function here is based on the score, additional time worked by the workers and the maximum working time of the workers.

Keywords: Task Assignment, Integer Programming, Multi Objective Optimization, Machine Learning

# ÖZETÇE

Bu projede, yazılım test grubu için çalışanlara iş atama yöntemi üzerinde incelemeler yapıldı. Yeni bir iş geldiğinde, gruptaki hangi bireylerin ne kadar sürede ve ne kadar verimli bitireceğini tahmin etmek zordu. Bunun için işle ilgili uzman grup tarafından test senaryoları belirlenmiş ve test paketi yazılmıştır. Test uzmanları, buradaki test adımlarına göre işi tamamlayacak ve sonucu döndürecektir. Böylelikle sisteme girilen hataların kritikliği doğrultusunda, ilgili değerlendirmeden aldıkları puanlara göre doğru kişiler tespit edilecektir.

Test grubundaki kişilerin özelliklerine göre işe alınacak yeni çalışanları belirlemek için test grubundaki kişilerin özelliklerine göre öneri yapılacaktır. Her yeni iş başvurusu için bu işleri tek tek yaptırmak yerine, sistem bize bu işi mevcut çalışanların özelliklerine göre yapma imkanı sunacaktır.

Sisteme girilen kısıtlamalara göre çok amaçlı bir optimizasyon modeli kullanılarak işçilere uygun görev atamaları yapılacaktır. Buradaki amaç fonksiyonu puana, işçilerin çalıştığı ek süreye ve işçilerin maksimum çalışma süresine dayanmaktadır.

Anahtar Kelimeler: İş Atama, Tamsayı Programlama, Çok Amaçlı Optimizasyon, Makine Öğrenme

# TABLE OF CONTENTS

[**ABSTRACT** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **iii**](#_TOC_250004)

[**ÖZETÇE** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **iv**](#_TOC_250003)

**I INTRODUCTION**  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .  **1**

**II PREVIOUS WORK** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **2**

**III METHODOLOGY** . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . **4**

**IV CONCLUSION**  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . **12**

**APPENDIX A — SOME ANCILLARY STUFF**  . . . . . . . . . . . . . . . . . . . . **13**

**REFERENCES**  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

# INTRODUCTION

The firm where the study will be carried out is the electronics factory. Examinations are carried out in the test section of the R&D department of this factory. In this section, television software goes through the weekly approval stage before production approval. Each tester had a fixed weekly assignment in the R&D department of a consumer electronics manufacturer.

Here, inefficiencies such as lack of qualification and time loss observed as problems were determined. In order to solve these problems, all the work was done to all workers first. As a result of the points they got from their jobs, it was determined whether they were qualified in that job. When a new worker is to be recruited, the job proposal is made based on the common features of the qualified workers. Machine learning algorithms are used here.

In the task assignment part, the points that users get from each task, the additional time they worked in the company and the maximum task time of each worker are taken as basis. Subsequently, the parameters and decision variables were determined. The testers who have fixed values ​​as parameters, the scores they get per task, their annual leave, the number of testers and tasks, and the deadlines of the tasks have been determined. As a decision variable, the information on whether a worker is assigned to a task, the amount of time the tester worked, the amount of additional and regular work hours of the tester, the amount of time spent per worker in order to identify the workers who could not be assigned the task, and the maximum task time amount of a worker were determined.

A job in the constraints be made whether more than one person, task duration of the set without exceeding the amount of work the week of the workers, overtime amount of time the legally defined in Turkey per worker, had worked at the factory workers of the total of pure working time with overtime amounts to equality, regular time amount of Turkey The equivalent of the calculation of the amount of idle time, the implementation of fair work distribution has taken place in. After the constraints determined, task assignment operations are applied by two different methods through mathematical modeling. In the weighted-sum method, assignment is applied according to a single objective function. In the Lexicographic method, it is aimed to find values ​​by giving weight to the decision variables and parameters used in the objective function.

1. PREVIOUS WORK

2.1 Workforce Efficiency Optimization

Every workplace has many optimization efforts to increase productivity. Often, the first task is to identify wastes. In a study conducted in Chile, the undisciplined behavior of employees was addressed. Considering this type of behavior on the basis of thousands of workers, this type of behavior has caused a lot of damage to the company. These kinds of mishaps have been taken into consideration and actions have been taken primarily on job satisfaction. It was emphasized that it is quite difficult to satisfy the current workers. It was stated that measuring this difficulty is a problem. The difficulties here were classified and solved according to the skills of existing workers [1]. According to the researches, it is understood that the damage to the company and the country's economy increased after many years due to instability and problems caused by the inability of people to work [2]. Roberts and Escudero, on the other hand, evaluated the wasted time of employees in a workplace study [3]. In our paper, we used data from the current workplace to improve efficiency. First of all, it was predicted that each employee could be successful in other tasks. Everyone will have a specific score for the tasks they work outside their own tasks. The scoring criterion will be based on the bugs entered and resolved. Here the score will increase in proportion to the priority of the bug. Missed bugs and completion time of the task will be inversely proportional to the score. A lot of research has been done here about which features to choose. In the estimation system that deals with job satisfaction, features are examined in two groups. While more abstract features such as expectations, beliefs are used as organizational features, the more concrete ones such as age, gender, education are preferred as individual features [4]. According to the information obtained from the literature, age, gender, educational status features were added to the algorithms developed for comparison. In 2011, in the machine learning job suggestion system, people are proposed according to their job backgrounds. Features include experience, degree of education, title and frequency of job change. This system also learns when the new employee will leave work [5]. Among the features used in this article, the title of employee parameter was added to the paper to compare the algorithm. In a research conducted in 2010, skills were determined according to the current performance of the workers. In the first case, the current worker can be assigned to any job, whereas in the second case, the performance of the external workers is performed by the employer [6]. In another study on task assignment in 2009, a suggestion system was made based on task priorities [7]. Existing algorithms were used as the suggestion method in our own study. The relationship between them will be examined and the decision will be made. Based on this article, comparisons were made using existing algorithms, regardless of the single algorithm. The study of Nigerian scientists focused on the relationship between leadership intelligence and emotional intelligence. Gender, age, marital status, academic qualification data were used while collecting data from employees. As a result, it was revealed that there is a significant relationship between emotional intelligence and work efficiency and these independent variables play a positive role [8]. The marital status argument was added to the writing, inspired by this article. Thus, it will be focused on whether married people can work overtime due to their children and whether divorced people can be aggressive. The relationship between psychosocial characteristics and work performance in a workplace in Japan was examined. In this study, the age, gender, marital status, educational background, daily working hours, seniority of the workers were kept as independent variables. It has been observed that as the daily working hours increase, the work efficiency decreases, as the age increases, the performance increases proportionally, and that married people are more successful than singles. While the above conclusions have been made among men, a correct proportion has not been established among women [9]. According to the conclusions drawn from this article, some variables played an important role while others stood out as ineffective elements. Since the daily working hours will remain constant in our project, other independent variables are taken into consideration and included in the study. A study was conducted on the relationship between job satisfaction and performance in the United Arab Emirates. Here, many features of the employees are considered as independent variables and they are entered into the system before starting to work. These variables are gender, age, term of office, marital status, position and nationality. As a result of various regression and correlation analyzes, it was revealed that some variables were not really important. While nationality, which is one of the variables here, stands out as a feature that directly affects the process, it was observed that age, gender and marital status variables had no effect [10]. Before starting the regression analysis in my study, domain-expert received opinions about what features might be from many unrelated subjects and articles were scanned. As observed, it is noticeable that every feature does not directly contribute to performance. Studies will be carried out with the features we have determined for their extraction. After the study, single and multiple releases will be established.

2.2 Prediction and Recommendation Algorithm

In Bektas’s, Kucukdeniz’s and Ozcan’s paper, the suggestion algorithm deals with wind energy. The estimated value is the wind speed. Forecasting in this area is more difficult than in other areas of study.Because there are continuously variable parameters such as air pressure and temperature. The uncertainty here has been overcome by using data from previous days. This was achieved by using the SVR algorithm. Small sample size has been observed to be a problem with wind speed estimation made easier by SVR [11]. Since the number of employee data I use in my project is less than 100, I added the SVR algorithm to my article for comparison. Sales amount estimation was made with Walmart brand, which is famous for its supply chain. In this article, regression algorithms and time series analysis methods are applied to sales data. The algorithms and methods currently used are: Linear Regression, Decision Tree Regression, Neural Network Regression and Naïve Method. As a result of the study, the performance and consistency of the regression was higher. Naïve Method was used when it was good in economic and financial time series. The accuracy of the regression algorithms was higher than the time series analysis methods [12]. As you can see, the performance of regression algorithms is higher than the time series algorithms. While making comparisons, regression algorithms were preferred because analysis will be made according to the features of the employees. In a study in which health expenditures were estimated, multiple regression models were established. Comparisons were made between 2 algorithms by giving different hyper parameter values. These algorithms are Random Forest Regression and SVR. With the first algorithm, it was possible to approach the optimum by generating a large number of trees. At the maximum level, similar subclasses were created. In SVR algorithm, the largest linear function of the margin was determined in order to differentiate linearly classifiable data. Hyper planes were created with these functions. According to the hyperplane, the data was assigned to classes. In fact, the aim here was to classify. At the end of the study, it was found that Random Forest Algorithm gave better results[13]. Based on this article, it was observed that success was achieved by deriving trees in complex data, and SVR algorithm gave positive results with small data. Using SVR for the company data that has 80 to 100 employees will show us better results.

2.3 Task Assignment

In this section, studies on task assignment are researched. A research has been done on the system where jobs are assigned to machines. Before the modeling started, it was determined for what purpose optimization will be applied. Then, it is aimed to do the job to be assigned to the machines with the least cost. Based on this, objective function has been examined in three cases. These are number of jobs equal number of machines, number of jobs are less than number of machines and number of jobs are greater than number of machines. According to these three cases, constraints are entered in the model. When the solution is applied according to the constraints, the cost is lower when the number of workers is high [14]. Another study was done in a university environment. When there were more than one midterm, final, and resit exam at the universities, there were conflicts in systematically assigning the lecturers to the exams. As an action, the decision variable containing the information that was appointed first was determined. Here, it is aimed to minimize the number of tasks. As a constraint, it is aimed to give each research assistant an equal number of tasks. The other constraint is to ensure the appointment of as many supervisors per day. As a result, the appointment was neutral. There was a balanced distribution [15]. In a military research in China, optimization studies were performed on multiple unmanned combat aerial vehicles. The aim is for military vehicles to hit targets in line with orders sent to the command center. Of course, uncertainties in the system were analyzed before creating this purpose function. Commands are prioritized as ωB and ωC. These variables constitute the decision variable structure of the function. OB is an operational benefit and OC is an operational cost variable. When other necessary restrictions are included in the system, the problem is solved around the objective function created. But of course, in this research, only cost and benefit were shown as targets. It is observed that more information will be needed for more complex models [16]. Decision support system has been developed for software management in a company. It is aimed to assign the available resources to the tasks at minimum cost and time. As a management, giving a resource to each event is determined as a restriction. Here, the duration of the work accumulated on the basis of each employee is kept variable. Whether the accumulated tasks are assigned to the particular worker in the system is kept as a Boolean as the decision variable. Based on this, a mathematical model is created and a solution is applied [17].

1. METHODOLOGY

The features are determined by domain experts and non-domain experts. The persons consulted here are sociologist, test group manager and test group chief. If tester gets the passing grade, he or she is successful. According to the available data, when the new candidate is wanted to be introduced to the system, it presents the result that the new worker can or cannot do this job according to the propositions determined by certain machine learning algorithms.

The features will role in suggesting whether new testers will be able to do the job. Based on the skills of existing workers, certain dependent-independent features were created. A dependent variable is a Boolean value that indicates whether the user can do that job. It returns us as 1 or 0 according to the arguments entered in the system. Independent variables were determined by taking the opinions of many people.

Features:

* Gender
* Age
* Experience
* Marital status
* Number of chldren
* Education status
* Number of previous job
* Seniority
* Title
* Estimated time of arrival

First, the success rate is determined by inserting the relevant algorithms between 1 and 0 for each feature. If the value is 0.5 or less, the user is unsuccessful. Between 0.5 and 1 will be considered successful. The values ​​obtained from all the latest features will be summed and divided by the number of features. If the value is greater than 0.5, it will result in successful, otherwise unsuccessful. In the final project we will compare the results of the algorithms returned to us.

Software group designs and implements all requirements of system. R&D test group’s task is to find bugs of the system. Their bugs are resolved by software group and software is released again. Until R&D group don’t see, this loop goes on. Then the software that there is no issue delivers quality department. Quality assurance tests the same parts. When they see any issues, R&D software group evaluate those bugs in their side. If it’s major or can be reproducable, it will be resolved by software group. After test is finished, software group releases new version. R&D test group re-test again, then quality assurance group re-test again.

Bugs entered into the R & D test group vary according to severity, duration and resolution status. The success criteria of the testers are affected by these variables. Testers's expectations are that the bugs are solved by the developers. If the bug entered into the system is resolved and the criticality of the bug is high, the tester gets a high score per bug entered. The task finish time determined by the engineer who wrote the test suite affects the success rate in inverse proportion. If the tester completes the task in a shorter period of time, the score will be positively reflected. Else, if it finishes longer, it will be inversely proportionally negative. Bugs entered by the testers but which are re-tested shall be interpreted as exhausting the system and scored as negative.

At the approval stage, the testers of the quality assurance unit will test the same test as end user. The bugs found by quality group testers and will be solved will negatively affect the score of the R & D testers.

* 1. Suggestion Algorithms Used in Recruitment

Scores that have been collected is classified as successfully and failed. For once, dependent variable is our decision. If testers succeed in their task, the dependent variable will be updated as “yes”. In order to be successful, they should get min 50 points in each task. Starting from this, the system will recommend us with different algorithms to assign new employee.

Different regression and classification algorithms were used to find a good recommendation. Decision tree regressor, support vector regressor and Naïve bayes algorithms have been selected. After user enters inputs in the system, the application will say that whether new employee can do this task or not. I will work through in the following.

The common feature of the algorithms used in the project is that they all deal with the same dependent variable. Arguments are entered into the system respectively. The model is trained according to the algorithm to be used. Estimates are taken for each argument individually. From this result, values are obtained (between 0 and 1) as to whether or not the candidate can perform the job per independent variable. The arithmetic average of these values obtained from all the latest features is obtained and a proposal is made as to whether or not the candidate will start this job.

* + 1. Naïve Bayes Algorithm

Naïve Bayes is simple machine learning and supervised algorithm. It is a classification algorithm and is extremely fast. It uses Bayes theory of probability. This is called 'Naïve' because the algorithm assumes that all attributes are independent of each other. It is used widely used in text classifications with multiple classes. The data presented for teaching must have a class / category. With the probability operations performed on the taught data, the new test data submitted to the system is operated according to the probability values obtained previously and it is tried to determine in which category the given test data belongs. Of course, the greater the number of taught data, the more accurate it is to determine the actual category of test data.

The Naïve Bayes classification method can be used in many ways, but it is important that what is classified here rather than how it is classified. In other words, the data to be taught can be binary or text data, where the proportional relationship between these data becomes more important than the data type and what it is.

The basis of the algorithm is based on Bayesian Theory. It is found probabilities “yes” and “no” in the below.

For example, if the decision variable is “yes”:

P( Independent Feature| Yes ) =               P( Yes| Independent Feature) P(Independent Feature)

                 —————————————————————————

P( Yes| Independent Feature) P(Independent Feature) +

P( Yes| not Independent Feature) P( not Independent Feature)

All probabilities of independent features are calculated according to “yes” and “no” selection. The system gives us that recommendation, whichever probability is greater.

As we have seen, the Naïve Bayes classification has determined the class or category of data presented to the system by a series of calculations defined according to probability principles.

We gave the system a sample dataset like in the following: Female-32 years old-10 years experience-divorced-has 3 children-postgraduate degree-previous job count is 2-seniority level is 7-junior-estimated time of arrival is 47 mins. According to the Naive Bayes algorithm, a worker with the features in this data set won’t be able to do this job.

* + 1. Decision Tree Regression Algorithm

Decision tree regression algorithm returns a result according to arithmetic mean of the related feature. If the value of feature is not in this range,

* If the value of feature that we enter in system is close to greater, it returns us the arithmetic mean of score of greater ones.
* If the value of feature that we enter in system is close to less, it returns us the arithmetic mean of score of less ones.
* If the value of feature that we enter in system is in the middle, it returns us the arithmetic mean of score of less ones.

Separates arguments into ranges based on information gain. When asked for a value from this range during the prediction, the answer is the average in this range (learned during training). For this reason, decision tree regression is discrete, not continuous, like other regression models. That is, it produces the same results for the desired estimates in a certain range.

The processes mentioned under Suggestion mechanism are as follow. All features are manually entered. Machine learning is performed for each feature. After the training part is completed, estimation is made according to the entered value. If the entered value is in this range, the arithmetic mean of the dependent variable corresponding to this range is taken. If not, the arithmetic mean of the dependent variable corresponding to the argument closest to this range is taken. If the corresponding argument is equal to the two ranges, it is rounded to the smaller one. This results in decimal values between 0 and 1 per feature. If we add these probabilities and divide by the number of arguments, if the value is between 0.5 and 1, the system suggests that the user can do this task. If it is between 0 and 0.5, the new employee cannot do the task. A value of 0.5 also indicates negativity.

We gave the same dataset: Female-32 years old-10 years experience-divorced-has 3 children-postgraduate degree-previous job count is 2-seniority level is 7-junior-estimated time of arrival is 47 mins. The arithmetic mean was obtained individually for each feature here. As a result, it was concluded that the person who will make the relevant application cannot do this job. The average could not exceed the threshold with 0.42555556.

* + 1. Support Vector Regression

The Support Vector Machine (SVM) is also used as a regression method, maintaining all the main features that characterize the algorithm (maximum margin). Support Vector Regression (SVR) uses the same principles as SVM for classification. It separates the data as successful and failed. The working principle of this algoritm is that maximizes between the distances of two classes. It is a method that used to normalize the range of properties of arguments or data.

We cannot always use linear models for our data. In such cases, we try to make sense of the data with other algorithms. One of these is support vector algorithms. When we apply support vector regression, it is to ensure that the range we will draw is within the maximum point. The points at which these maximum intervals are cut are called support points. Hence, Radial Basis Function (RBF) method was used to determine nonlinear range.

With this data set and the polynomial model it will be established, it will be possible to predict whether a tester can do this task according to the problem level it will be tried to solve. Thus, we can conclude that our target variable y is “Accept” and the remaining properties are independent variables.

Firstly, inputs are entered into system. Machine learning is performed for each feature. After the training part is completed, estimation is made according to the entered value. This results in decimal values between 0 and 1 per feature according to support vector regression. If we add these probabilities and divide by the number of arguments, if the value is between 0.5 and 1, the system suggests that the user can do this task. If it is between 0 and 0.5, the new employee cannot do the task. A value of 0.5 also indicates negativity.

The same data set used in the Decision tree regressor algorithm was also used for the support vector regression algorithm. The result obtained here exceeded the threshold value with 0.61212549 and was successful for the system.

In the previous algorithm, this person could not do this job, but he could do it with the support vector regression algorithm. It is predicted that some data in the system should be scaled before entering data in the algorithm. Here, age, experience, arrival time data are scaled with a standard scaler as they can reach 2-digit values according to other features.

Why do we need to scale data?

* It is used to use data under the same metric. Think of this process as roughly turning both of them into meters before processing the kilometer and millimeter. Data mining algorithms cannot separate them.
* It associates and normalizes similar data.
  + 1. Comparison of Algorithms

Decision tree regressor returns incorrect results if the data entered into the system is not available. That's why the data we enter from the keyboard must be in our database. Otherwise, it may return incorrect results. In Naive Bayes, all features are given equal weights, so it can draw wrong conclusions. In the study conducted here, it was observed that the relevant candidate could not do the job when the same data set was entered. Data are in scaled format. The application of the simple vector algorithm for a data set of 80 people has been observed ideally. Due to its logic, it gave an effective result compared to the other regression algorithm because it makes a clear distinction between other features. According to the data set entered, it is concluded that the candidate cannot do this job as a result of the calculation. I chose the SVR algorithm in my study, because my data set consists of 80 people, so it gives better results than the algorithms used.

* 1. MULTI-OBJECTIVE OPTIMIZATION MODEL FOR TASK ASSIGNMENT PROBLEM

Address specific issues in the task assignment optimization problem. The fair distribution of tasks to the workers, the assignment of tasks appropriate to the field in which they are successful and their working less time were discussed. Two different ways were followed in obtaining the solution. In the first way, weights were given to objective functions. In the other way, ranking criteria were taken as basis.

a) List of Parameters:

: Let represent the overall point of tester i for task j

: Casual leave

n: Amount of testers

m: Amount of tasks

: Duration time of task j

i ϵ I I : set of tester

j ϵ J J: set of tasks

J = ⋃

b) Decision Variables:

: Let be a binary variable indicating whether tester i is assigned to task j

: The number of hours the tester worked

: Overtime that represents except normal working hours

: Regular time that represents normal working hours

: The amount of idle time for workers

d: Let d be the variable that is equal to the maximum task duration time of a tester.

c) Objective Function:

We will try to assign to tasks in way to maximize to total point of the testers.

d) Constraints:





































Constraint-1: Some tasks can be assigned to only one tester.

Constraint-2: Some tasks can be assigned to more than one tester.

Constraint-3: The duration of the tasks to be assigned should not exceed the working hours of the workers.

Constraint-4: Over time is less than substraction of maximum working hour and regular time. The daily working hours required by law in Turkey, a worker does not exceed 11 hours[18].

Constraint-5: The sum of regular time and overtime at work must equal Yi.

Constraint-6: When calculating regular time, use of casual leave. Every tester must work at least 45 hours for a week in Turkey [19].

Constraint-7: Company wants to know how long workers will not be doing task.

Constraint-8: It the represent the duration time of the tasks assigned to the workers is minimized.

Constraint-9: Tasks that can be assigned to only one tester were specified.

Constraint-10: Tasks that can be assigned to more than one tester were specified.

Constraint-11: The amout time that workers don’t work must not be less than 0.

Our aim in modeling is to make the workers do all the tasks and to apply the most appropriate weekly assignments with the resulting data. During the assignment, the duration of the tasks, the hourly leave amount used by the workers, the scores they get from the tasks play a role. After these needs were determined, constraints were studied. It was stated as the 1 and 2 constraints that not every job in the factory is one person, and more than one person is required for some. The Boolean decision variable to be used in task assignment takes a float variable in multiple tasks and a binary variable in single-person tasks. Of course again, this range of values ​​is between 1 and 0. The amount of time people spend at the factory should not exceed 66 hours per week. This time, gives the amount of legal weekly working time for Turkey [18]. Under normal circumstances, this factory states that each worker must complete a minimum of 45 hours. The time a worker spends in the office is the difference between the minimum time spent at the factory and the weekly excuse leave. This was included in the system as a constraint. Our next constraint is that the weekly work time of the worker is less than or equal to the time spent in the office. We have already stated in our previous constraint that this worker can use weekly leave. A decision variable has been defined where we can learn which of the workers worked and how long. This period should not exceed 21 due to legality. The factory is open only 12 hours a day. Workers' lunch breaks do not exceed 1 hour. The amount of time they need to fill daily is 9 hours. Therefore, they can work a maximum of 2x5 hours during the week. Workers who are not obliged to work on weekends can only work on Saturdays, depending on their work situation. During this day, the time they work does not exceed 11 hours. Therefore, the maximum amount of time a worker will do is 21 hours per week. The pure time spent by workers per week was also defined as the decision variable. This time is equal to the difference between the time spent in the factory and the additional overtime. Another constraint is that the time spent for the task i will be less than 45 to the difference of excuse leave. If the worker never uses his / her excuse leave, this period will be less than 45. The total duration of the jobs to be assigned will not exceed 45. A separate decision variable was defined in order to find the idle time of the workers in the company. This time should be less than the difference between the amount of time spent in the factory and the amount of work done by the workers.

* + 1. Weighted-Sum Method

A single objective function was used in here. It was aimed to reach a solution in recursive ways by using the decision variables and parameters determined in the modeling phase. Here, with the decision variable representing fair distribution, the decision variable representing the overtime work done by the workers was minimized and the scores they got per task were maximized. It was observed that the solution gave optimum value.

* + 1. Lexicographic Method

In the weighted sum method solution, 3 decision variables were evaluated in a single objective function. Here we divided the variables that we combined in a single objective function in the Basic solution into 3 different objective functions. These 3 functions were processed in 6 different orders from 3! In the end, 3 different results were obtained. The results are as follows. The values in the table were obtained from the sample data set.

Table 1: Lexicographic Method Application

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Highest Priority** | **Medium Priority** | **Low Priority** | **Score** | **Overtime** | **Max Diff in Working** |
| Max Diff in Working | Overtime | Score | 1699 | -30 | -48 |
| Overtime | Max Diff in Working | Score | 1699 | -30 | -48 |
| Max Diff in Working | Score | Overtime | 1699 | -30 | -48 |
| Score | Max Diff in Working | Overtime | 1733 | -30 | -60 |
| Overtime | Score | Max Diff in Working | 1799,13 | -123 | -66 |
| Score | Overtime | Max Diff in Working | 1799,13 | -123 | -66 |

When simplified, 3 different results were obtained.

According to the table, when the priority of the maximum differences in working variable is kept at the lowest level, we see that there is no fair assignment between the workers' tasks and the extra overtime is too much. However, here we observe that the score also increased contrary to these values. When the priority of the score variable is kept the lowest, we see that there is a decrease in the total scores of the workers, but the maximum differences in working value decrease in the time spent on overtime. When the priority of the score is kept the highest and the value of the time spent on overtime is kept the lowest, we see that the difference in maximum differences increases compared to the previous situation. To sum up, we have seen that in 3 different situations one is not dominant over the other. In such cases, pareto should be found optimal [20]. Here, while improving one of the variables, the optimality of the other variables is at risk, so the pareto optimality is tried to be found.

When we compare the 2 methods used in the task assignment section, we see that the lexicographic method is preferred. We see that we can get the solution with the weight given to the employer, whichever is the priority.

1. CONCLUSIONS AND FUTURE WORK

As a result, it has been observed that SVR algorithm gives better results for data of 80 people. While making task assignment, actions were taken to ensure that workers do not work overtime and are assigned fair jobs. In this way, the work will not be delayed for the employer and the employee will not get tired. The task assignment method gives hope. It can be developed in next phase. If it was used higher datasets, it could obtain higher success rate. Different mathematical models could be used to apply task assignment solution. To predict the machine learning, it could be used different classification and regression algorithms.

**A sample Python script for the Scores Algorithm**

from openpyxl import \*  
import xlrd  
  
def calculate(id, args):  
 worker = 80  
 j = 2  
 loc = (r'D:\Projects\Tez\FindScore\Scores.xlsx')  
 wb = load\_workbook(r'D:\Projects\Tez\FindScore\Scores.xlsx')  
 wb2 = xlrd.open\_workbook(loc)  
  
 sheets = wb.sheetnames  
 sheet1 = wb[sheets[id]]  
 sheet2 = wb2.sheet\_by\_index(0)  
  
 for i in range(1, worker + 1):  
 temp = sheet2.cell\_value(i, 0)  
 print("\n", "for ", temp)  
 fixed\_showstopper = input("Enter fixed showstopper bug numbers: ")  
 point = int(fixed\_showstopper) \* 20  
 sheet1.cell(row=i + 1, column=j).value = point  
  
 fixed\_high = input("Enter fixed high bug numbers: ")  
 point2 = int(fixed\_high) \* 10  
 sheet1.cell(row=i + 1, column=j + 1).value = point2  
  
 fixed\_medium = input("Enter fixed medium bug numbers: ")  
 point3 = int(fixed\_medium) \* 4  
 sheet1.cell(row=i + 1, column=j + 2).value = point3  
  
 fixed\_low = input("Enter fixed low bug numbers: ")  
 point4 = int(fixed\_low) \* 2  
 sheet1.cell(row=i + 1, column=j + 3).value = point4  
  
 retest\_showstopper = input("Enter retest showstopper bug numbers: ")  
 point5 = int(retest\_showstopper) \* -20  
 sheet1.cell(row=i + 1, column=j + 4).value = point5  
  
 retest\_high = input("Enter retest high bug numbers: ")  
 point6 = int(retest\_high) \* -10  
 sheet1.cell(row=i + 1, column=j + 5).value = point6  
  
 retest\_medium = input("Enter retest medium bug numbers: ")  
 point7 = int(retest\_medium) \* -4  
 sheet1.cell(row=i + 1, column=j + 6).value = point7  
  
 retest\_low = input("Enter retest low bug numbers: ")  
 point8 = int(retest\_low) \* -2  
 sheet1.cell(row=i + 1, column=j + 7).value = point8  
  
 qa\_showstopper = input("Enter fixed showstopper bug numbers by Quality Assurance: ")  
 point9 = int(qa\_showstopper) \* -20  
 sheet1.cell(row=i + 1, column=j + 8).value = point9  
  
 qa\_high = input("Enter fixed high bug numbers by Quality Assurance: ")  
 point10 = int(qa\_high) \* -10  
 sheet1.cell(row=i + 1, column=j + 9).value = point10  
  
 qa\_medium = input("Enter fixed medium bug numbers by Quality Assurance: ")  
 point11 = int(qa\_medium) \* -4  
 sheet1.cell(row=i + 1, column=j + 10).value = point11  
  
 qa\_low = input("Enter fixed low bug numbers by Quality Assurance: ")  
 point12 = int(qa\_low) \* -2  
 sheet1.cell(row=i + 1, column=j + 11).value = point12  
  
 sure = input("How long did the test take? Enter data in hours: ")  
 point13 = 1 / int(sure) \* 20  
 sheet1.cell(row=i + 1, column=j + 12).value = int(point13)  
  
 total = point + point2 + point3 + point4 + point5 + point6 + point7 + point8 + point9 + point10 + point11 + point12 + point13  
 sheet1.cell(row=i + 1, column=j + 13).value = int(total)  
 wb.save('Scores.xlsx')  
  
 j = 2  
  
print("\nEnter datas for Release Approval Test:")  
calculate(0, 1)  
  
print("\nEnter datas for Portal Test:")  
calculate(1, 1)  
  
print("\nEnter datas for Logging Test:")  
calculate(2, 1)

**A sample Python script for the Naïve Bayes Algorithm**

import pandas as pd  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.preprocessing import LabelEncoder  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score  
  
dataset = pd.read\_excel('Accept.xlsx', sheet\_name='Logging')  
dataset.head()  
  
number = LabelEncoder()  
dataset['Sex'] = number.fit\_transform(dataset['Sex'])  
dataset['Age'] = number.fit\_transform(dataset['Age'])  
dataset['Experience'] = number.fit\_transform(dataset['Experience'])  
dataset['Marriage'] = number.fit\_transform(dataset['Marriage'])  
dataset['Childbearing'] = number.fit\_transform(dataset['Childbearing'])  
dataset['Graduation'] = number.fit\_transform(dataset['Graduation'])  
dataset['Former Job Quantity'] = number.fit\_transform(dataset['Former Job Quantity'])  
dataset['Seniority'] = number.fit\_transform(dataset['Seniority'])  
dataset['Title'] = number.fit\_transform(dataset['Title'])  
dataset['Estimated Time of Arrival'] = number.fit\_transform(dataset['Estimated Time of Arrival'])  
  
dataset['Accept'] = number.fit\_transform(dataset['Accept'])  
features = ["Sex", "Age", "Experience", "Marriage", "Childbearing", "Graduation", "Former Job Quantity", "Seniority",  
 "Title", "Estimated Time of Arrival"]  
target = "Accept"  
  
features\_train, features\_test, target\_train, target\_test = train\_test\_split(dataset[features], dataset[target],  
 test\_size = 1, random\_state = 0)  
  
model = GaussianNB()  
model.fit(features\_train, target\_train)  
  
pred = model.predict(features\_test)  
accuracy = accuracy\_score(target\_test, pred)  
  
sex = input("Enter Gender of Worker: \nFor Female: 0, For Male: 1\n")  
age = input("Enter Age of Worker: \n")  
exp = input("Enter Numbers of Years Worked: \n")  
mar = input("Enter Marrige Status: \n For Divorced: 0, For Married: 1, For Single: 2\n")  
chi = input("Enter Numbers of Children: \n")  
gra = input("Enter Graduate Status: \n For Bacheloors: 0, For Foundation Degree: 1, For High School: 2, For Master: 3, "  
 "For PostGraduate: 4\n")  
pre = input("Enter Numbers of Previous Jobs: \n")  
sen = input("Enter Seniority Level: \n")  
tit = input("Enter Title of Worker: \n For Architect: 0, For Junior: 1, For Senior Architect: 2, "  
 "For Senior Specialist: 3, For Specialist: 4, For Technician: 5 \n")  
arr = input("Enter Estimated Time of Arrival (min): \n")  
  
result = (model.predict([[int(sex), int(age), int(exp), int(mar), int(chi), int(gra), int(pre), int(sen),  
 int(tit), int(arr)]]))  
  
print(result)  
  
if result == 1:  
 print("New Employee can do this task!")  
  
else:  
 print("New Employee cannot do this task!")

**A sample Python script for the Decision Tree Regressor Algorithm**

import numpy as np  
import pandas as pd  
from sklearn.tree import DecisionTreeRegressor  
from sklearn.preprocessing import LabelEncoder  
  
def dt\_recommend():  
  
 dataset = pd.read\_excel('Accept.xlsx', sheet\_name='Logging')  
 dataset.head()  
  
 number = LabelEncoder()  
 dataset['Sex'] = number.fit\_transform(dataset['Sex'])  
 dataset['Marriage'] = number.fit\_transform(dataset['Marriage'])  
 dataset['Graduation'] = number.fit\_transform(dataset['Graduation'])  
 dataset['Title'] = number.fit\_transform(dataset['Title'])  
 dataset['Accept'] = number.fit\_transform(dataset['Accept'])  
  
 dataset['Sex'] = dataset['Sex'].astype(float)  
 dataset['Marriage'] = dataset['Marriage'].astype(float)  
 dataset['Childbearing'] = dataset['Childbearing'].astype(float)  
 dataset['Graduation'] = dataset['Graduation'].astype(float)  
 dataset['Seniority'] = dataset['Seniority'].astype(float)  
 dataset['Title'] = dataset['Title'].astype(float)  
 dataset['Age'] = dataset['Age'].astype(float)  
 dataset['Experience'] = dataset['Experience'].astype(float)  
 dataset['Former Job Quantity'] = dataset['Former Job Quantity'].astype(float)  
 dataset['Estimated Time of Arrival'] = dataset['Estimated Time of Arrival'].astype(float)  
  
  
 x = dataset.iloc[:, 0:1].values  
 y = dataset.iloc[:, -1].values  
  
 regressor = DecisionTreeRegressor(random\_state=0)  
 regressor.fit(x, y)  
  
 sex = input("\nFor Female: 0, For Male: 1\nEnter Gender of Worker: ")  
  
 y\_pred1 = regressor.predict(np.array([sex]).reshape(-1, 1))  
 print("Average score for ", sex, "is= ", y\_pred1)  
  
  
 x = dataset.iloc[:, 1:2].values  
 y = dataset.iloc[:, -1].values  
  
 regressor = DecisionTreeRegressor(random\_state=0)  
 regressor.fit(x, y)  
  
 age = input("\nEnter Age of Worker: ")  
  
 y\_pred2 = regressor.predict(np.array([age]).reshape(-1, 1))  
 print("Average score for ", age, "is= ", y\_pred2)  
  
  
 x = dataset.iloc[:, 2:3].values  
 y = dataset.iloc[:, -1].values  
  
 regressor = DecisionTreeRegressor(random\_state=0)  
 regressor.fit(x, y)  
  
 exp = input("\nEnter Numbers of Years Worked: ")  
  
 y\_pred3 = regressor.predict(np.array([exp]).reshape(-1, 1))  
 print("Average score for ", exp, "is= ", y\_pred3)  
  
  
 x = dataset.iloc[:, 3:4].values  
 y = dataset.iloc[:, -1].values  
  
 regressor = DecisionTreeRegressor(random\_state=0)  
 regressor.fit(x, y)  
  
  
 mar = input("\nFor Divorced: 0, For Married: 1, For Single: 2\nEnter Maritial Status: ")  
  
 y\_pred4 = regressor.predict(np.array([mar]).reshape(-1, 1))  
 print("Average score for ", mar, "is= ", y\_pred4)  
  
  
 x = dataset.iloc[:, 4:5].values  
 y = dataset.iloc[:, -1].values  
  
 regressor = DecisionTreeRegressor(random\_state=0)  
 regressor.fit(x, y)  
  
  
 chi = input("\nEnter Numbers of Children: ")  
  
 y\_pred5 = regressor.predict(np.array([chi]).reshape(-1, 1))  
 print("Average score for ", chi, "is= ", y\_pred5)  
  
 x = dataset.iloc[:, 5:6].values  
 y = dataset.iloc[:, -1].values  
  
 regressor = DecisionTreeRegressor(random\_state=0)  
 regressor.fit(x, y)  
  
  
 grad = input("\nFor Bacheloors: 0, For Foundation Degree: 1, For High School: 2, For Master: 3, "  
 "For PostGraduate: 4\nEnter Graduate Status: ")  
  
 y\_pred6 = regressor.predict(np.array([grad]).reshape(-1, 1))  
 print("Average score for ", grad, "is= ", y\_pred6)  
  
  
 x = dataset.iloc[:, 6:7].values  
 y = dataset.iloc[:, -1].values  
  
 regressor = DecisionTreeRegressor(random\_state=0)  
 regressor.fit(x, y)  
  
  
 job = input("\nEnter Numbers of Previous Jobs: ")  
  
 y\_pred7 = regressor.predict(np.array([job]).reshape(-1, 1))  
 print("Average score for ", job, "is= ", y\_pred7)  
  
 x = dataset.iloc[:, 7:8].values  
 y = dataset.iloc[:, -1].values  
  
 regressor = DecisionTreeRegressor(random\_state=0)  
 regressor.fit(x, y)  
  
  
 sen = input("\nEnter Seniority Level: \n")  
  
 y\_pred8 = regressor.predict(np.array([sen]).reshape(-1, 1))  
 print("Average score for ", sen, "is= ", y\_pred8)  
  
  
 x = dataset.iloc[:, 8:9].values  
 y = dataset.iloc[:, -1].values  
  
 regressor = DecisionTreeRegressor(random\_state=0)  
 regressor.fit(x, y)  
  
  
 tit = input("\nFor Architect: 0, For Junior: 1, For Senior Architect: 2, For Senior Specialist: 3, For Specialist: 4, For Technician: 5 \nEnter Title of Worker: ")  
  
 y\_pred9 = regressor.predict(np.array([tit]).reshape(-1, 1))  
 print("Average score for ", tit, "is= ", y\_pred9)  
  
  
 x = dataset.iloc[:, 9:10].values  
 y = dataset.iloc[:, -1].values  
  
 regressor = DecisionTreeRegressor(random\_state=0)  
 regressor.fit(x, y)  
  
  
 est = input("\nEnter Estimated Time of Arrival (min): ")  
  
 y\_pred10 = regressor.predict(np.array([est]).reshape(-1, 1))  
 print("Average score for ", est, "is= ", y\_pred10)  
  
 total= y\_pred1+y\_pred2+y\_pred3+y\_pred4+y\_pred5+y\_pred6+y\_pred7+y\_pred8+y\_pred9+y\_pred10  
 avg = total/10  
  
 print(avg)  
  
 if avg > 0.5:  
 print("New Employee can do this task!")  
  
 else:  
 print("New Employee cannot do this task!")  
  
dt\_recommend()

**A sample Python script for the Support Vector Regression Algorithm**

import numpy as np  
import pandas as pd  
from sklearn.svm import SVR  
from sklearn.preprocessing import StandardScaler  
from sklearn.preprocessing import LabelEncoder  
  
  
def experience\_decision():  
  
 dataset = pd.read\_excel('Accept.xlsx', sheet\_name='Logging')  
 dataset.head()  
  
 number = LabelEncoder()  
 dataset['Sex'] = number.fit\_transform(dataset['Sex'])  
 dataset['Marriage'] = number.fit\_transform(dataset['Marriage'])  
 dataset['Graduation'] = number.fit\_transform(dataset['Graduation'])  
 dataset['Title'] = number.fit\_transform(dataset['Title'])  
 dataset['Accept'] = number.fit\_transform(dataset['Accept'])  
  
 dataset['Sex'] = dataset['Sex'].astype(float)  
 dataset['Marriage'] = dataset['Marriage'].astype(float)  
 dataset['Childbearing'] = dataset['Childbearing'].astype(float)  
 dataset['Graduation'] = dataset['Graduation'].astype(float)  
 dataset['Seniority'] = dataset['Seniority'].astype(float)  
 dataset['Title'] = dataset['Title'].astype(float)  
 dataset['Age'] = dataset['Age'].astype(float)  
 dataset['Experience'] = dataset['Experience'].astype(float)  
 dataset['Former Job Quantity'] = dataset['Former Job Quantity'].astype(float)  
 dataset['Estimated Time of Arrival'] = dataset['Estimated Time of Arrival'].astype(float)  
  
  
 x = dataset.iloc[:, 0:1].values  
 y = dataset.iloc[:, -1].values  
  
 regressor = SVR(kernel='rbf', gamma='auto')  
 regressor.fit(x, y)  
  
 sex = input("Enter Gender of Worker: \nFor Female: 0, For Male: 1\n")  
  
 y\_pred1 = regressor.predict(np.array([sex]).reshape(-1, 1))  
 print("Average score for ", sex, "is= ", y\_pred1)  
  
  
 x = dataset.iloc[:, 1:2].values  
 y = dataset.iloc[:, -1].values  
  
 sc\_X = StandardScaler()  
 sc\_y = StandardScaler()  
 x = sc\_X.fit\_transform(x)  
 y = sc\_y.fit\_transform(y.reshape(-1, 1))  
  
 regressor = SVR(kernel='rbf', gamma='auto')  
 regressor.fit(x, y)  
  
 age = input("Enter Age of Worker: \n")  
  
 y\_pred2 = regressor.predict(np.array([age]).reshape(-1, 1))  
 print("Average score for ", age, "is= ", y\_pred2)  
  
  
 x = dataset.iloc[:, 2:3].values  
 y = dataset.iloc[:, -1].values  
  
 sc\_X = StandardScaler()  
 sc\_y = StandardScaler()  
 x = sc\_X.fit\_transform(x)  
 y = sc\_y.fit\_transform(y.reshape(-1, 1))  
  
 regressor = SVR(kernel='rbf', gamma='auto')  
 regressor.fit(x, y)  
  
 exp = input("Enter Numbers of Years Worked: \n")  
  
 y\_pred3 = regressor.predict(np.array([exp]).reshape(-1, 1))  
 print("Average score for ", exp, "is= ", y\_pred3)  
  
  
 x = dataset.iloc[:, 3:4].values  
 y = dataset.iloc[:, -1].values  
  
 regressor = SVR(kernel='rbf', gamma='auto')  
 regressor.fit(x, y)  
  
  
 mar = input("Enter Marrige Status: \n For Divorced: 0, For Married: 1, For Single: 2\n")  
  
 y\_pred4 = regressor.predict(np.array([mar]).reshape(-1, 1))  
 print("Average score for ", mar, "is= ", y\_pred4)  
  
  
 x = dataset.iloc[:, 4:5].values  
 y = dataset.iloc[:, -1].values  
  
 regressor = SVR(kernel='rbf', gamma='auto')  
 regressor.fit(x, y)  
  
  
 chi = input("Enter Numbers of Children: \n")  
  
 y\_pred5 = regressor.predict(np.array([chi]).reshape(-1, 1))  
 print("Average score for ", chi, "is= ", y\_pred5)  
  
 x = dataset.iloc[:, 5:6].values  
 y = dataset.iloc[:, -1].values  
  
 regressor = SVR(kernel='rbf', gamma='auto')  
 regressor.fit(x, y)  
  
  
 grad = input("\n For Bacheloors: 0, For Foundation Degree: 1, For High School: 2, For Master: 3, "  
 "For PostGraduate: 4\nEnter Graduate Status: \n")  
  
 y\_pred6 = regressor.predict(np.array([grad]).reshape(-1, 1))  
 print("Average score for ", grad, "is= ", y\_pred6)  
  
  
 x = dataset.iloc[:, 6:7].values  
 y = dataset.iloc[:, -1].values  
  
 regressor = SVR(kernel='rbf', gamma='auto')  
 regressor.fit(x, y)  
  
  
 job = input("Enter Numbers of Previous Jobs: \n")  
  
 y\_pred7 = regressor.predict(np.array([job]).reshape(-1, 1))  
 print("Average score for ", job, "is= ", y\_pred7)  
  
 x = dataset.iloc[:, 7:8].values  
 y = dataset.iloc[:, -1].values  
  
 regressor = SVR(kernel='rbf', gamma='auto')  
 regressor.fit(x, y)  
  
  
 sen = input("Enter Seniority Level: \n")  
  
 y\_pred8 = regressor.predict(np.array([sen]).reshape(-1, 1))  
 print("Average score for ", sen, "is= ", y\_pred8)  
  
  
 x = dataset.iloc[:, 8:9].values  
 y = dataset.iloc[:, -1].values  
  
 regressor = SVR(kernel='rbf', gamma='auto')  
 regressor.fit(x, y)  
  
  
 tit = input("Enter Title of Worker: \n For Architect: 0, For Junior: 1, For Senior Architect: 2, "  
 "For Senior Specialist: 3, For Specialist: 4, For Technician: 5 \n")  
  
 y\_pred9 = regressor.predict(np.array([tit]).reshape(-1, 1))  
 print("Average score for ", tit, "is= ", y\_pred9)  
  
  
 x = dataset.iloc[:, 9:10].values  
 y = dataset.iloc[:, -1].values  
  
 sc\_X = StandardScaler()  
 sc\_y = StandardScaler()  
 x = sc\_X.fit\_transform(x)  
 y = sc\_y.fit\_transform(y.reshape(-1, 1))  
  
 regressor = SVR(kernel='rbf', gamma='auto')  
 regressor.fit(x, y)  
  
  
 est = input("Enter Estimated Time of Arrival (min): \n")  
  
 y\_pred10 = regressor.predict(np.array([est]).reshape(-1, 1))  
 print("Average score for ", est, "is= ", y\_pred10)  
  
 total= y\_pred1+y\_pred2+y\_pred3+y\_pred4+y\_pred5+y\_pred6+y\_pred7+y\_pred8+y\_pred9+y\_pred10  
 avg = total/10  
  
 print(avg)  
  
 if avg > 0.5:  
 print("New Employee can do this task!")  
  
 else:  
 print("New Employee cannot do this task!")  
  
experience\_decision()

**A sample Python script for the Weighted Sum Algorithm**

import setup  
from docplex.mp.model import Model  
from docplex.util.environment import get\_environment  
  
n = 10 #worker  
m = 20 #number of tasks  
  
EN = [i for i in range(1, n+1)]  
EM = [j for j in range(1, m+1)]  
  
Jm = [j for j in range(1, 5)]  
J1 = [j for j in range(5, 21)]  
  
  
N = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
  
  
a = [[40, 30, 20, 60, 30, 50, 70, 70, 80, 90, 80, 40, 30, 60, 20, 30, 50, 40, 90, 50],  
 [30, 20, 10, 90, 80, 70, 60, 50, 40, 55, 35, 25, 90, 85, 80, 40, 15, 10, 35, 25],  
 [90, 80, 70, 60, 50, 40, 30, 40, 50, 20, 45, 65, 95, 60, 80, 45, 35, 80, 65, 90],  
 [65, 45, 95, 35, 65, 85, 90, 60, 40, 30, 95, 80, 55, 65, 45, 90, 65, 75, 65, 80],  
 [55, 65, 35, 95, 60, 80, 65, 90, 55, 45, 35, 95, 90, 80, 70, 65, 35, 90, 80, 50],  
 [60, 50, 40, 90, 80, 75, 80, 90, 80, 85, 90, 65, 75, 85, 65, 85, 95, 95, 90, 80],  
 [55, 65, 60, 50, 55, 55, 65, 75, 70, 60, 55, 50, 55, 60, 60, 65, 60, 80, 65, 55],  
 [20, 30, 20, 40, 20, 50, 40, 45, 65, 60, 45, 35, 45, 60, 65, 65, 45, 35, 30, 50],  
 [50, 40, 45, 80, 90, 65, 65, 35, 90, 65, 85, 95, 90, 50, 55, 65, 95, 85, 90, 55],  
 [90, 65, 35, 45, 65, 65, 75, 85, 70, 70, 90, 65, 60, 95, 55, 65, 65, 85, 65, 60]  
 ]  
  
  
F = [50, 40, 35, 45, 20, 10, 15, 30, 20, 20, 25, 15, 10, 25, 20, 15, 10, 25, 30, 20]  
  
  
twoDim = [(i, j) for i in EN for j in EM]  
  
oneSize = [i for i in EN]  
oneSize\_j = [j for j in EM]  
  
  
  
mdl = Model('IDEAL TASK ASSIGNMENT')  
  
x1 = mdl.binary\_var\_dict(twoDim, name='x1') #Let Xij be a binary variable indicating whether tester i is assigned to task j  
xm = mdl.continuous\_var\_dict(twoDim, name='xm')  
mdl.add\_constraints(xm[i, j] >= 0 for i in EN for j in Jm)  
mdl.add\_constraints(xm[i, j] <= 1 for i in EN for j in Jm)  
  
y = mdl.continuous\_var\_dict(oneSize, name='y') #  
b = mdl.continuous\_var\_dict(oneSize, name='b') #  
  
o = mdl.continuous\_var\_dict(oneSize, name='O') #Overtime that represents except normal working hours  
R = mdl.continuous\_var\_dict(oneSize, name='R') #Regular time that represents normal working hours  
d = mdl.continuous\_var(name='d') #Let d be the variable that is equal to the maximum task duration time of a tester.  
  
  
  
mdl.maximize(mdl.sum(x1[i, j]\*a[i-1][j-1] for i in EN for j in J1)+  
 mdl.sum(xm[i, j]\*a[i-1][j-1] for i in EN for j in Jm) -  
 (mdl.sum(o[i] for i in oneSize))-20\*d)  
  
mdl.add\_constraints(mdl.sum(x1[i, j] for i in EN) == 1 for j in J1)  
  
mdl.add\_constraints(mdl.sum(xm[i, j] for i in EN) == 1 for j in Jm)  
  
mdl.add\_constraints(mdl.sum(x1[i, j]\*F[j-1]for j in J1)+  
 mdl.sum(xm[i, j] \* F[j - 1]for j in Jm) <= y[i] for i in EN)  
  
mdl.add\_constraints(o[i] <= 21 for i in oneSize)  
  
mdl.add\_constraints(y[i] == o[i]+R[i] for i in oneSize)  
  
mdl.add\_constraints(R[i]+N[i-1] <= 45 for i in EN)  
  
mdl.add\_constraints(mdl.sum(x1[i, j]\*F[j-1]for j in J1)+  
 mdl.sum(xm[i, j] \* F[j - 1]for j in Jm) == y[i]-b[i] for i in EN)  
  
mdl.add\_constraints(mdl.sum(x1[i, j]\*F[j-1]for j in J1)+  
 mdl.sum(xm[i, j] \* F[j - 1] for j in Jm)<= d for i in EN)  
  
#ALL DECISION VARIABLES MUST BE GREATER THAN 0.  
  
mdl.add\_constraints(b[i] >= 0 for i in oneSize)  
mdl.add\_constraints(y[i] >= 0 for i in oneSize)  
mdl.add\_constraints(o[i] >= 0 for i in oneSize)  
mdl.add\_constraints(R[i] >= 0 for i in oneSize)  
  
#print(mdl.export\_to\_string())  
  
mdl.solve(log\_output=True)  
mdl.print\_solution()  
print(mdl.solve\_details)  
  
with get\_environment().get\_output\_stream("solution.json") as fp:  
 mdl.solution.export(fp, "json")

**A sample Python script for the Lexicographic Algorithm**

from docplex.mp.model import Model  
  
def order\_1\_2\_3():  
 sense = "max"  
 exprs = [score, overtime, fair]  
 priorities = [1, 2, 3]  
 weights = [1]  
 mdl.set\_multi\_objective(sense, exprs, priorities, weights, abstols=None, reltols=None, names=None)  
  
 cozum = mdl.solve(log\_output=True)  
 mdl.print\_solution()  
  
 get\_values = cozum.get\_objective\_value()  
 print("Objective value is: "+str(get\_values[0]+get\_values[1]+get\_values[2])+"\n")  
  
 print("Solution for "+str(priorities) +": \n")  
 print(mdl.solve\_details)  
  
 for v in mdl.iter\_integer\_vars():  
 print(v, " = ", v.solution\_value)  
  
def order\_1\_3\_2():  
 sense = "max"  
 exprs = [score, overtime, fair]  
 priorities = [1, 3, 2]  
 weights = [1]  
 mdl.set\_multi\_objective(sense, exprs, priorities, weights, abstols=None, reltols=None, names=None)  
  
 cozum = mdl.solve(log\_output=True)  
 mdl.print\_solution()  
  
 get\_values = cozum.get\_objective\_value()  
 print("Objective value is: "+str(get\_values[0]+get\_values[1]+get\_values[2])+"\n")  
  
 print("Solution for "+str(priorities) +": \n")  
 print(mdl.solve\_details)  
  
 for v in mdl.iter\_integer\_vars():  
 print(v, " = ", v.solution\_value)  
  
def order\_2\_1\_3():  
 sense = "max"  
 exprs = [score, overtime, fair]  
 priorities = [2, 1, 3]  
 weights = [1]  
 mdl.set\_multi\_objective(sense, exprs, priorities, weights, abstols=None, reltols=None, names=None)  
  
 cozum = mdl.solve(log\_output=True)  
 mdl.print\_solution()  
  
 get\_values = cozum.get\_objective\_value()  
 print("Objective value is: "+str(get\_values[0]+get\_values[1]+get\_values[2])+"\n")  
  
 print("Solution for "+str(priorities) +": \n")  
 print(mdl.solve\_details)  
  
 for v in mdl.iter\_integer\_vars():  
 print(v, " = ", v.solution\_value)  
  
def order\_2\_3\_1():  
 sense = "max"  
 exprs = [score, overtime, fair]  
 priorities = [2, 3, 1]  
 weights = [1]  
 mdl.set\_multi\_objective(sense, exprs, priorities, weights, abstols=None, reltols=None, names=None)  
  
 cozum = mdl.solve(log\_output=True)  
 mdl.print\_solution()  
  
 get\_values = cozum.get\_objective\_value()  
 print("Objective value is: "+str(get\_values[0]+get\_values[1]+get\_values[2])+"\n")  
  
 print("Solution for "+str(priorities) +": \n")  
 print(mdl.solve\_details)  
  
 for v in mdl.iter\_integer\_vars():  
 print(v, " = ", v.solution\_value)  
  
def order\_3\_1\_2():  
 sense = "max"  
 exprs = [score, overtime, fair]  
 priorities = [3, 1, 2]  
 weights = [1]  
 mdl.set\_multi\_objective(sense, exprs, priorities, weights, abstols=None, reltols=None, names=None)  
  
 cozum = mdl.solve(log\_output=True)  
 mdl.print\_solution()  
  
 get\_values = cozum.get\_objective\_value()  
 print("Objective value is: "+str(get\_values[0]+get\_values[1]+get\_values[2])+"\n")  
  
 print("Solution for "+str(priorities) +": \n")  
 print(mdl.solve\_details)  
  
 for v in mdl.iter\_integer\_vars():  
 print(v, " = ", v.solution\_value)  
  
def order\_3\_2\_1():  
 sense = "max"  
 exprs = [score, overtime, fair]  
 priorities = [3, 2, 1]  
 weights = [1]  
 mdl.set\_multi\_objective(sense, exprs, priorities, weights, abstols=None, reltols=None, names=None)  
  
 cozum = mdl.solve(log\_output=True)  
 mdl.print\_solution()  
  
 get\_values = cozum.get\_objective\_value()  
 print("Objective value is: "+str(get\_values[0]+get\_values[1]+get\_values[2])+"\n")  
  
 print("Solution for "+str(priorities) +": \n")  
 print(mdl.solve\_details)  
  
 for v in mdl.iter\_integer\_vars():  
 print(v, " = ", v.solution\_value)  
  
mdl = Model(name='Lexicographic Solution')  
  
n = 10  
m = 20   
  
EN = [i for i in range(1, n+1)]  
EM = [j for j in range(1, m+1)]  
  
Jm = [j for j in range(1, 5)]  
J1 = [j for j in range(5, 21)]  
  
  
N = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
  
  
a = [[40, 30, 20, 60, 30, 50, 70, 70, 80, 90, 80, 40, 30, 60, 20, 30, 50, 40, 90, 50],  
 [30, 20, 10, 90, 80, 70, 60, 50, 40, 55, 35, 25, 90, 85, 80, 40, 15, 10, 35, 25],  
 [90, 80, 70, 60, 50, 40, 30, 40, 50, 20, 45, 65, 95, 60, 80, 45, 35, 80, 65, 90],  
 [65, 45, 95, 35, 65, 85, 90, 60, 40, 30, 95, 80, 55, 65, 45, 90, 65, 75, 65, 80],  
 [55, 65, 35, 95, 60, 80, 65, 90, 55, 45, 35, 95, 90, 80, 70, 65, 35, 90, 80, 50],  
 [60, 50, 40, 90, 80, 75, 80, 90, 80, 85, 90, 65, 75, 85, 65, 85, 95, 95, 90, 80],  
 [55, 65, 60, 50, 55, 55, 65, 75, 70, 60, 55, 50, 55, 60, 60, 65, 60, 80, 65, 55],  
 [20, 30, 20, 40, 20, 50, 40, 45, 65, 60, 45, 35, 45, 60, 65, 65, 45, 35, 30, 50],  
 [50, 40, 45, 80, 90, 65, 65, 35, 90, 65, 85, 95, 90, 50, 55, 65, 95, 85, 90, 55],  
 [90, 65, 35, 45, 65, 65, 75, 85, 70, 70, 90, 65, 60, 95, 55, 65, 65, 85, 65, 60]  
 ]  
  
  
F = [50, 40, 35, 45, 20, 10, 15, 30, 20, 20, 25, 15, 10, 25, 20, 15, 10, 25, 30, 20] #480  
  
twoDim = [(i, j) for i in EN for j in EM]  
  
oneSize = [i for i in EN]  
oneSize\_j = [j for j in EM]  
  
x1 = mdl.binary\_var\_dict(twoDim, name='x1') #Let Xij be a binary variable indicating whether tester i is assigned to task j  
xm = mdl.continuous\_var\_dict(twoDim, name='xm')  
mdl.add\_constraints(xm[i, j] >= 0 for i in EN for j in Jm)  
mdl.add\_constraints(xm[i, j] <= 1 for i in EN for j in Jm)  
  
y = mdl.continuous\_var\_dict(oneSize, name='y') #OKK  
b = mdl.continuous\_var\_dict(oneSize, name='b') #  
  
o = mdl.continuous\_var\_dict(oneSize, name='O') #Overtime that represents except normal working hours  
R = mdl.continuous\_var\_dict(oneSize, name='R') #Regular time that represents normal working hours  
d = mdl.continuous\_var(name='d') #Let d be the variable that is equal to the maximum task duration time of a tester.  
  
  
mdl.add\_constraints(mdl.sum(x1[i, j] for i in EN) == 1 for j in J1)  
  
mdl.add\_constraints(mdl.sum(xm[i, j] for i in EN) == 1 for j in Jm)  
  
mdl.add\_constraints(mdl.sum(x1[i, j]\*F[j-1]for j in J1)+  
 mdl.sum(xm[i, j] \* F[j - 1]for j in Jm) <= y[i] for i in EN)  
  
mdl.add\_constraints(o[i] <= 21 for i in oneSize)  
  
mdl.add\_constraints(y[i] == o[i]+R[i] for i in oneSize)  
  
mdl.add\_constraints(R[i]+N[i-1] <= 45 for i in EN)  
  
mdl.add\_constraints(mdl.sum(x1[i, j]\*F[j-1]for j in J1)+  
 mdl.sum(xm[i, j] \* F[j - 1]for j in Jm) == y[i]-b[i] for i in EN)  
  
mdl.add\_constraints(mdl.sum(x1[i, j]\*F[j-1]for j in J1)+  
 mdl.sum(xm[i, j] \* F[j - 1] for j in Jm)<= d for i in EN)  
  
#ALL DECISION VARIABLES MUST BE GREATER THAN 0.  
  
mdl.add\_constraints(b[i] >= 0 for i in oneSize)   
mdl.add\_constraints(y[i] >= 0 for i in oneSize)  
mdl.add\_constraints(o[i] >= 0 for i in oneSize)  
mdl.add\_constraints(R[i] >= 0 for i in oneSize)  
  
score = mdl.continuous\_var(name='score')  
overtime = mdl.continuous\_var(name='overtime', lb=-150)  
fair = mdl.continuous\_var(name='fair', lb=-150)  
  
mdl.add\_constraint(score == mdl.sum(x1[i, j]\*a[i-1][j-1] for i in EN for j in J1)+  
 mdl.sum(xm[i, j]\*a[i-1][j-1] for i in EN for j in Jm))  
mdl.add\_constraint(overtime == (-1)\*(mdl.sum(o[i] for i in oneSize)))  
mdl.add\_constraint(fair == (-1)\*d)  
  
  
order\_1\_2\_3()  
order\_1\_3\_2()  
order\_2\_1\_3()  
order\_2\_3\_1()  
order\_3\_1\_2()  
order\_3\_2\_1()

**Optimization Scenario and Examples**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Highest Priority** | **Medium Priority** | **Low Priority** | **Score** | **Overtime** | **Max Diff in Working** |
| Max Diff in Working | Overtime | Score | 1699 | -30 | -48 |
| Overtime | Max Diff in Working | Score | 1699 | -30 | -48 |
| Max Diff in Working | Score | Overtime | 1699 | -30 | -48 |
| Score | Max Diff in Working | Overtime | 1733 | -30 | -60 |
| Overtime | Score | Max Diff in Working | 1799,13 | -123 | -66 |
| Score | Overtime | Max Diff in Working | 1799,13 | -123 | -66 |

According to the table, when the priority of the maximum differences in working variable is kept at the lowest level, we see that there is no fair assignment between the workers' tasks and the extra overtime is too much. However, here we observe that the score also increased contrary to these values. When the priority of the score variable is kept the lowest, we see that there is a decrease in the total scores of the workers, but the maximum differences in working value decrease in the time spent on overtime. When the priority of the score is kept the highest and the value of the time spent on overtime is kept the lowest, we see that the difference in maximum differences increases compared to the previous situation. To sum up, we have seen that in 3 different situations one is not dominant over the other. In such cases, pareto should be found optimal [20]. Here, while improving one of the variables, the optimality of the other variables is at risk, so the pareto optimality is tried to be found.

**Parameters**

Overall point of tester i for task j

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 40 | 30 | 20 | 60 | 30 | 50 | 70 | 70 | 80 | 90 | 80 | 40 | 30 | 60 | 20 | 30 | 50 | 40 | 90 | 50 |
| 30 | 20 | 10 | 90 | 80 | 70 | 60 | 50 | 40 | 55 | 35 | 25 | 90 | 85 | 80 | 40 | 15 | 10 | 35 | 25 |
| 90 | 80 | 70 | 60 | 50 | 40 | 30 | 40 | 50 | 20 | 45 | 65 | 95 | 60 | 80 | 45 | 35 | 80 | 65 | 90 |
| 65 | 45 | 95 | 35 | 65 | 85 | 90 | 60 | 40 | 30 | 95 | 80 | 55 | 65 | 45 | 90 | 65 | 75 | 65 | 80 |
| 55 | 65 | 35 | 95 | 60 | 80 | 65 | 90 | 55 | 45 | 35 | 95 | 90 | 80 | 70 | 65 | 35 | 90 | 80 | 50 |
| 60 | 50 | 40 | 90 | 80 | 75 | 80 | 90 | 80 | 85 | 90 | 65 | 75 | 85 | 65 | 85 | 95 | 95 | 90 | 80 |
| 55 | 65 | 60 | 50 | 55 | 55 | 65 | 75 | 70 | 60 | 55 | 50 | 55 | 60 | 60 | 65 | 60 | 80 | 65 | 55 |
| 20 | 30 | 20 | 40 | 20 | 50 | 40 | 45 | 65 | 60 | 45 | 35 | 45 | 60 | 65 | 65 | 45 | 35 | 30 | 50 |
| 50 | 40 | 45 | 80 | 90 | 65 | 65 | 35 | 90 | 65 | 85 | 95 | 90 | 50 | 55 | 65 | 95 | 85 | 90 | 55 |
| 90 | 65 | 35 | 45 | 65 | 65 | 75 | 85 | 70 | 70 | 90 | 65 | 60 | 95 | 55 | 65 | 65 | 85 | 65 | 60 |

Duration time of task j

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 50 | 40 | 35 | 45 | 20 | 10 | 15 | 30 | 20 | 20 | 25 | 15 | 10 | 25 | 20 | 15 | 10 | 25 | 30 | 20 |

Casual leave

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

[1] H.A. Eiselt,V. Marianov / Computers & Operations Research 35 (2008) 513–524

[2] CCH:http://www.toolkit.cch.com/text/P05\_5315.asp.

[3] Roberts S, Escudero L. Scheduling of plant maintenance personnel. Journal of Optimization Theory and Applications 1983; 39:323–43.

[4] Cron, W.L. and Slocum, J.W. (1986), “The inﬂuence of career stages on salespeople’s job attitudes, work perceptions, and performance”, Journal of Marketing, Vol. 23, pp. 119-29.

[5] Ioannis Paparrizos, B. Barla Cambazoglu, Aristiden Gionis, Machine Learned Job Recommendation, page 325

[6] Brilon, S., Job Assignment with Multivariate Skills, Preprints of the Max Planck Institute for Research on Collective Goods, 2010

[7] Moreira, M., and Costa, A., A minimalist yet efficient tabu search algorithm for balancing assembly lines with disabled workers, XLI SBPO Pesquisa Operacional na Gestão do Conhecimento, 2009: p. 660-671.

[8] Yekeen Adebayo, J., Olayide, R., & Saheed, O. (2012). Influence of Leadership Styles and Emotional Intelligence on Job Performance of Local Government Workers in Osun State, Nigeria. Journal of AlternativePerspectives in the Social Sciences (2012) Vol 3, No 4.

[9] M Nagami, A Tsutsumi, M Tsuchiya, K Morimoto - Industrial health, 2010

[10] ME Ibrahim, S Al Sejini, Al Qassimi Omaima Abdul Aziz - Al Qassimi Omaima Abdul Aziz, 2004

[11] Z. Bektas, T. Kucukdeniz, T. Ozcan, A Comparison of Support Vector Regression and Multivariable Grey Model for ShortTerm Wind Speed Forecasting, 2017: p. 48-51.

[12] C. Catal, K. Ece, B. Arslan and A. Akbulut, Benchmarking of Regression and Time Series Analysis Techniques for Sales Forecasting, 2019: p. 21-24.

[13] Songül ÇINAROĞLU, Comparison of Machine Learning Regression Methods to Predict Health Expenditures, 2017: p. 179-181

[14] Fathallah Fadhil Khalaf Al-Abdulhameed, A linear Programming Formulation of Assignment Problems, 2011:21-24

[15] Mehmet ÖZÇALICI, Assignment of Surveillance Assessment Problems Optimization with Working Pages, 2016: 104-111

[16] Jun Wang, Pengcheng Luo, Xinwu Hu, Xiaonan Zhang, Combining an Extended SMAA-2 Method with Integer Linear Programming for Task Assignment of Multi-UCAV under Multiple Uncertainties, 2018: 5-8

[17] Nepomuceno, V.S., Fontana, M.E., Decision Support System to Project Software Management, 2013

[18] İŞ KANUNUNA İLİŞKİN ÇALIŞMA SÜRELERİ YÖNETMELİĞİ

[19] Mehmet CENGİZ, Arif TEMİR, Birsen ERDİN, Working Times in Labor Law in 50 questions, p. 6

[20] proximedia. "Pareto Front". www.cenaero.be. Retrieved October 8, 2018.