An International Journal of Optimization and Control: Theories & Applications
ISSN: 2146-0957 eISSN: 2146-5703
Vol.15, No.2, pp.330-342 (2025)
https://doi.org/10.36922/ijocta.1694

RESEARCH ARTICLE

End of day process optimization through multi-mode resource constrained project scheduling - A banking case study

Ezgi Karabulut Türkseven[1*], Ezgi Genç[1], and Ilgın Şafak[2]

[1]Department of Industrial Engineering, Sabanci University, Istanbul, Turkey
[2]Fibabanka R&D Center, Istanbul, Turkey
ezgi.turkseven@sabanciuniv.edu, ezgi.genc@alumni.sabanciuniv.edu, Ilgin.safak@fibabanka.com.tr

ARTICLE INFO

Article History:
Received: September 29, 2024
Accepted: February 8, 2025
Published Online: April 8, 2025

Keywords:
Multi-mode resource constrained project scheduling
Thread allocation
Simulated annealing
End of Day process optimization
Banking case study

AMS Classification:
90C11; 90C27

ABSTRACT

An End-of-Day process is a batch job that includes a sequence of programs, wherein tasks are completed automatically at times specified by a scheduler. The efficient allocation of resources for the timely execution of tasks allows a company to reduce the overall time needed for the completion of the work and improve customer satisfaction by delivering orders on time. This paper presents a case study of a Turkish bank with the objective of minimizing the duration of the end-of-day process through the optimization of work scheduling and resource allocation. This problem is modeled as a multi-mode resource-constrained project scheduling problem, optimally resolved by mixed-integer programming, and approximated via simulated annealing heuristic. The scheduler in this paper can also be used to assess the importance of scheduling and crashing tasks, along with the sufficiency of the infrastructure to optimize the End-of-Day process.

CC BY

1. Introduction

Financial institutions such as banks carry out a financial End of Day (EOD) process to wrap up the day's operations and prepare the system for the next day's transactions. This is considered to be one of the most important parts of the maintenance of any banking application. The EOD process is employed by banks to produce and disseminate reports to stakeholders for regulatory oversight, business users, and downstream systems. An EOD process is a batch job including a sequence of programs, wherein tasks are automatically executed at specified intervals via a scheduler. Batch processing is a technique for handling substantial volumes of data and transmitting this data in groups to the designated system. Scheduling is the process of allocating limited resources to process tasks over some period in an optimal manner. A certain amount of specified resources is required to complete each task within a specified period, commonly referred to as the processing time.

Efficient allocation of resources to perform tasks on time will allow a company to achieve its objectives, such as minimizing the time required to complete all tasks, maximizing customer satisfaction, maximizing plant throughput, maximizing profit, or minimizing production costs.[1] Several methods can be used to optimize the EOD process, including code optimization, code parallelization, data partitioning, and load balancing. However, deploying applications in the same physical location and using identical technology significantly increases the probability of achieving a coordinated EOD process for each application, leading to a considerably faster information flow and a reduced duration of the EOD process. In large-scale enterprise applications that are inherently dispersed and utilize diverse technologies, such as microservices, the coordination of an

*Corresponding Author

EOD process can be more challenging, leading to an extended EOD process duration. As a result, traditional batch processing methods may not be sufficiently efficient for managing EOD batch processes in such systems. Therefore, there is a need for a new EOD algorithm optimized for enterprise systems, including microservices applications.

This paper presents a case study of a Turkish bank that seeks to minimize the duration of the EOD process by optimizing resource allocation and task sequencing. We thus consider the order of execution of the EOD tasks and the amount of resources allocated to execute each task. This problem can be modeled as a variant of Project Scheduling Problem, called Multi-Mode Resource Constrained Project Scheduling Problem (MRCPSP). In this paper, we propose an exact solution to this problem to be used as a benchmark and a heuristic approach for practical implementation. Our contribution is a mixed integer programming formulation of the problem, and performance analysis of the heuristic approach in this case study. We have further demonstrated how our optimization tool can be utilized as a simulator for analyzing how efficiently the resources are used and identifying bottleneck resources.

The paper is organized as follows: Section 2 provides a brief overview of the literature on Resource Constrained Project Scheduling Problem (RCPSP). In Section 3 we describe the system in our case study and establish the specifics of this problem. Section 4 presents the two solution approaches. Computational results are provided in Section 5, and concluding remarks are in Section 6.

2. Literature survey

Project Scheduling Problem (PSP) is a relatively simple problem in the literature to determine the order in which the tasks should be executed in an optimal manner with respect to the precedence constraints such that the total makespan of the project is minimized. PERT Algorithm[2] solves this problem in polynomial time.

RCPSP has the same goal and precedence constraints as PSP. In addition, in RCPSP each task requires a non-perishable resource usage during execution, and the limited amount of resources available places an additional constraint on the tasks that can be executed simultaneously.

Kone et al.[3] provide a thorough overview of the literature on RCPSP. The two most popular Mixed Integer Programming (MIP) formulations of RCPSP are time-indexed formulation,[4] which requires time-discretization, and flow-based formulation.[5] The time-indexed formulation provides fairly good linear relaxation bounds. However, it performs poorly with broad time horizons. In either of the two cases above, solving a MIP for large-scale projects is cumbersome. Hence, a wide majority of heuristic algorithms have been developed for RCPSP.

A significant portion of these heuristics is sequencing algorithms that are constructive greedy heuristics. These algorithms assign each task with a priority based on certain properties, and sequences them in a serial or parallel manner in a single or a multi-pass with respect to the resource constraints. There exists a wide variety of publications for different priority rules, sequence order, or number of passes.[6–13] Kolisch and Hartmann[14,15] and Hartmann and Briskorn[16] provide a broad overview and comparison of these algorithms. In addition to these priority rule-based algorithms, Li and Willis[17] propose a forward-backward scheduling algorithm, Valls et al.[18] propose a hybrid genetic algorithm, and Lee and Kim[19] use search heuristics to solve RCPSP.

A more complicated variant of RCPSP is MRCPSP, where the tasks have multiple modes with different resource requirements and durations. It is important to note that increased resource usage will not only reduce task durations but also decrease the number of tasks that can be executed simultaneously. Hence, it will lead to an increase in the total makespan. Due to this additional trade-off, MRCPSP requires more complex heuristics than straightforward sequencing algorithms. Exact solution methods such as[20–24] have been stated to be effective for problems up to 20 activities and two modes per activity.[24] For larger instances, various meta-heuristic based algorithms, such as Simulated Annealing[25–28] and Genetic Algorithm[29–31] have been proposed to solve MRCPSP.

A general distribution of the work in the aforementioned literature is summarized in Table 1. Where we stand in the literature is the second category, that focuses on the exact solution of the Multi-Mode Resource Constrained Project Scheduling Problem. All of the stated papers utilize the time-indexed formulation of the problem; and our contribution is the flow-based formulation. In this paper we also describe Simulated Annealing implementation of the same problem. However, that should not be considered as a scientific contribution (due to the abundance of existing studies), but rather as an implementation choice from a practical application context.

Table 1. Literature Summary

	Multi-Mode	Exact Method	Heuristic
Alvarez et al.[6]			
Boctor[7]			
Özdamar et al.[8]			
Ulusoy et al.[9]	·	·	✓
Lawrence[10]			
Kolisch et al.[11]			
Özdamar et al.[12]			
Ulusoy et al.[13]			
Talbot[20]			
Patterson et al.[21]			
Sprecher et al.[22]	✓	✓	·
Hartmann et al.[23]			
Sprecher et al.[24]			
Boctor[25]			
Boctor[26]			
Bouleimen et al.[27]			
Józefowska et al.[28]	✓	·	✓
Mori et al.[29]			
Özdamar[30]			
Alcaraz et al.[31]			

3. Problem definition

This section describes the specifics of the considered system. However, due to confidentiality considerations, the numerical values provided will be approximate rather than exact, intended to demonstrate the scale of the problem.

The EOD system consists of 400 tasks executed at the end of each day. 170 of these tasks referred to as *FixedTasks*, have fixed scheduled execution times. Hence, these tasks are not included in the optimization problem. The remaining 230 EOD tasks are partitioned into different stages. The stage structure is modeled with dummy stage tasks with zero duration and zero resource consumption that denote the beginning of a stage, STAGE1, STAGE2, STAGE3, STAGE4, and STAGE5. All tasks in the first stage have STAGE2 task as their final successor, and all tasks in the second stage have STAGE2 task as their original predecessor. Due to this structure, instead of solving a large scheduling problem with 230 tasks, we break the problem down based on the stages, and determine the schedule for each stage separately.

The system infrastructure can simultaneously execute 20 tasks in parallel and allocate 250 threads for the execution of EOD operations. A thread is the smallest unit of execution in a process and enables a software to execute numerous tasks simultaneously, utilizing shared resources like memory and file handles. Tasks and threads are the only renewable resources in the system. Although *FixedTasks* are not part of the scheduling problem, they still consume resources from the same pool. In a deterministic setting where the duration of each task is constant in execution, resource reduction due to *FixedTasks* would be modeled as varying resource availabilities throughout the time span. However, due to variances in task execution durations, a method that requires such a level of precision was not deemed to be a good fit.

To account for the necessity of allocating a portion of these two resources for the *FixedTasks*, which are beyond the model's scope, the proposed method designates a fraction of the resources for the *FixedTasks* and solves the optimization problem assuming the remaining resource availability. Additional sensitivity analysis is conducted to observe the impact of the reserved resources on the EOD project span, and the results are provided in Section 5.

An important feature of our problem is determining the number of threads to allocate for the execution of each task. Task durations are generally defined as a problem parameter to preserve the linearity of the formulation. However, in our problem, task durations are directly impacted by the thread allocation decisions. In order to preserve linearity of the mathematical model, we refrain from defining the number of threads allocated to each task as an independent integer variable. Instead, by careful examination of the data and the task structure, we determine several alternative thread counts, and define each alternative as a mode for executing the task. Hence, we arrive at a combinatorial problem of finding an optimal mode of execution for each task. Furthermore, recall that the two resources in the system mentioned above are both renewable and currently integrated into the infrastructure. Therefore, our problem should not be considered as a time/cost trade-off RCPSP as described in,[32] as having only renewable resources and no additional costs differentiates the current situation from *crashing*. It can, however, be interpreted as a time/resource trade-off RCPSP, wherein only one of the two resources influences the task duration.

To be used as model parameters, we are given two sets of data: (1) infrastructural data, which includes information on the precedence relations, the stage information of each task, and whether each task can be parallelized or not; and (2) daily job logs, that include information regarding past 6 months on what time each task started and ended execution (and hence the duration) and how many threads are allocated to each task. The number of threads allocated to each task has been kept

constant during the 6-month time interval; and this thread allocation amount is the default mode for each task.

The MRCPSP formulation requires a resource usage value for each resource and a duration value for each mode of each task. For the default mode of each task, the resource usage value is the constant value observed in the daily logs, and the duration value is the median of the daily duration calculated from daily logs.

Infrastructural data sets indicate that 40% of the tasks cannot be parallelized, and must be executed by a single thread. For the remaining tasks, depending on the duration of the task, we define two or three alternative candidate thread numbers as the resource requirements of new modes. Unfortunately, the durations of the tasks in these modes have not been observed, and must therefore be estimated. We base our estimation on the following inverse relation

$$(dur_{new}) = \frac{(dur_{default})(thrdCnt_{default})}{thrdCnt_{new}} \quad (1)$$

where dur refers to the duration of the task, $thrdCnt$ refers to the thread count, and the subscripts $default$ refers to the observed values in the data set and new refers to the proposed alternatives.

An essential characteristic of our problem is its iterative and data-driven nature. Since every day different subsets of the EOD tasks are executed, the scheduler needs to be called regularly on a daily basis. As non-default task modes are selected and executed, observations on their daily durations are collected in daily job logs. These observations can be used to improve the estimations in (1). This enhancement plan is anticipated to ultimately converge and utilize higher quality estimations for task durations.

4. Methodology

The solution methodology applied to solve this MRCPSP has been two-fold: optimization via mixed integer programming, and approximation via heuristics. The ultimate goal of this case study is creating a tool, a scheduler for the bank's implementation. Although MIP formulation is guaranteed to give optimal results, efficiently solving a large-scale MIP to optimality requires licensed commercial solvers. Therefore, for implementation purposes, a heuristic algorithm has been devised. More specifically, similar to other work on MRCPSP, *Simulated Annealing* algorithm has been deployed. The MIP formulation and the optimal solutions are used as benchmark values on comparing the quality of solutions generated by the heuristic. The details of the MIP formulation and the Simulated Annealing algorithm are provided in Sections 4.1 and 4.2, respectively.

4.1. Mathematical optimization, MIP

The data in our case study has highly varying task durations, with some tasks taking as short as 0.015 minutes for completion, and some tasks taking as long as 60 minutes for completion. Formulating this problem with a time-indexed MIP model requires an immense number of time intervals for discretization of time; hence leads to inefficiency. For this reason, we base the MIP formulation of our problem on the flow based model described in.[5] To the best of our knowledge, including the multi-mode concept in the flow-based formulation is our contribution.
The model consists of the following variables:

S_i	Start time of task i
x_{ij}	binary variable indicating whether task i is completed before task j begins (the completion time of task i is at most the start time of task j)
z_{im}	binary variable indicating whether task i is executed in mode m
y_{ijm}	binary variable corresponding to $x_{ij} \times z_{im}$
f_{ijr}	the amount of resource r transferred from task i to task j.

Let $A = \{0, \ldots, n+1\}$ be the set of tasks such that 0 is the starting task and $n+1$ is the ending task; $E \subseteq A \times A$ be the set of precedence relations such that $(i, j) \in E$ indicates that task i is a predecessor of task j, M_i be the number of execution modes of task i, R be the number of renewable resources, b_{imr} be the amount of resource r task i requires in mode m, and p_{im} be the estimated duration of task i in mode m. By design, the dummy tasks $i \in \{0, n+1\}$ have only a single mode, and for each of these two tasks, the amount of resource r required to execute the task in the default mode 1 (b_{i1r}) equals the availability of resource r at any given point in time.

The flow formulated in this model can be interpreted as follows: Given a network of tasks connected via the precedence relations, all of the available resources are gathered at task 0, split among the tasks during their execution, and hence flow through the tasks, and finally accumulate at task $n+1$.

We will illustrate the flow-based formulation concept with a simple single-mode resource example. Consider the following tasks in Figure 1, where p_i and b_i denoted on top of each node represent the duration and resource usage of tasks respectively, and arcs (i, j) are precedence relations. Notice by the selection of b_0 and b_4 that the availability of the resource is 5.

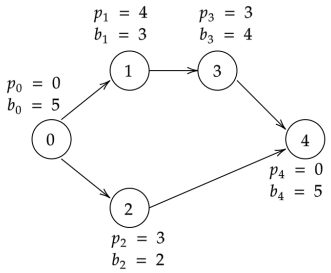

Figure 1. An example project scheduling problem

For this project, assume the start times are selected such that $S_0 = 0$, $S_1 = 0$, $S_2 = 0$, $S_3 = 4$, and $S_4 = 7$. This is a precedence-feasible schedule. Based on the start times, the binary variables x_{ij} indicating whether task i is completed before task j begins take the values $x_{01} = x_{02} = x_{03} = x_{04} = 1$, $x_{13} = x_{14} = 1$, $x_{23} = x_{24} = 1$, $x_{34} = 1$ and $x_{ij} = 0$ for the remaining. For resource feasibility, we'd like to find a feasible flow in the graph induced by only allowing arcs with $x_{ij} = 1$. In this example, the network in which we search for a flow is given in Figure 2.

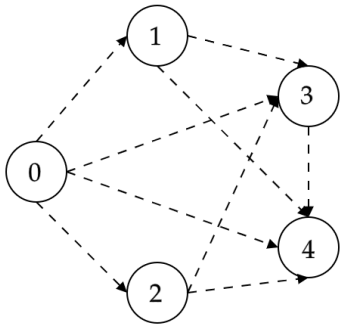

Figure 2. Flow network of the first solution

Now consider another set of start times, $S_0 = 0$, $S_1 = 0$, $S_2 = 2$, $S_3 = 4$, and $S_4 = 7$, which is also precedence-feasible. However, in this solution, task 2 is not completed before task 3 begins, therefore $x_{23} = 0$. This change updates the flow network as shown in Figure 3. Unfortunately, this network does not have a flow with the required in- and out-flow values. The in-flow constraints for tasks 1 and 2 enforce $f_{01} = 3$ and $f_{02} = 2$. The out-flow constraint of task 0 then enforces $f_{03} = 0$. The out-flow constraint of 1 enforces $f_{13} \leq 3$. As $(0, 3)$ and $(1, 3)$ are the only incoming arcs to 3, the in-flow constraint of 4 ($f_{03} + f_{13} = 4$) cannot be satisfied. Therefore we can conclude that these start times are not resource-feasible.

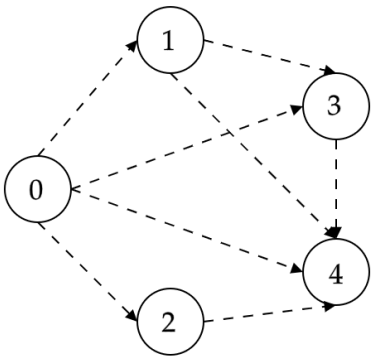

Figure 3. Flow network of the second solution

In this network, we search for a flow such that the in-flow of node i equals the out-flow of node i, which also equals the resource usage of task i, b_i, for $i \in \{1, 2, 3\}$. For $i = 0$, we only enforce the out-flow constraint, and for $i = 4$ we only enforce the in-flow constraint. Such a feasible flow in this network can be given as $f_{01} = 3$, $f_{02} = 2$, $f_{13} = 3$, $f_{23} = 1$, $f_{24} = 1$, $f_{34} = 4$, and the remaining flows equal to zero. Hence, we can conclude that the selected start times are both precedence and resource feasible, and a feasible solution to the single mode single resource constraint project scheduling problem.

This logic is generalized to multiple resources by creating flow networks (such as Figures 2 and 3) independently for each resource.

Let's assume that the problem in Figure 1 is changed as follows. The resource usage and duration of task 1 is no longer fixed to 3 and 4 respectively, but rather we are given the following options: Use 3 units of resource and have a duration of 4, as before; use 2 units of resource and have a duration of 6 (less resource - more time); or use 4 units of resource and have a duration of 3 (more resource - less time). Now our model also needs to choose which configuration to execute. We model this as 3 separate modes for

task 1. $p_{11} = 4$ and $b_{111} = 3$ denote the first mode, $p_{12} = 6$ and $b_{121} = 2$ denote the second mode, and $p_{13} = 3$ and $b_{131} = 4$ denote the third mode. To generalize our model to such settings, we utilize another set of binary variables z_{im} to determine the active mode for each task and repeat the same procedure of finding the flow with the selected p_{im} and b_{imr} values.

A more formal description of the flow-based MRCPSP is formulated below, with the given assumptions and variable definitions:

$$\min \quad S_{n+1} \tag{2}$$

$$\text{s.t.} \quad S_0 = 0 \tag{3}$$

$$x_{ij} = 1 \qquad \forall\ (i,j) \in E \tag{4}$$

$$x_{ij} + x_{ji} \leq 1 \qquad \forall\ i,j \in A \tag{5}$$

$$x_{ik} \geq x_{ij} + x_{jk} - 1 \quad \forall\ i,j,k \in A \tag{6}$$

$$\sum_{m=1}^{M_i} z_{im} = 1 \qquad \forall\ i \in A \tag{7}$$

$$y_{ijm} \leq z_{im} \qquad \forall\ i,j \in A\ , \quad m = 1,\ldots,M_i \tag{8}$$

$$\sum_{m=1}^{M_i} y_{ijm} = x_{ij} \qquad \forall\ i,j \in A \tag{9}$$

$$y_{ijm} \geq x_{ij} + z_{im} - 1 \quad \forall\ i,j \in A \quad m = 1,\ldots,M_i \tag{10}$$

$$S_j - S_i \geq \sum_{m=1}^{M_i} p_{im} y_{ijm} - \hat{M}_{ij}(1 - x_{ij}) \qquad \forall\ i,j \in A \tag{11}$$

$$\sum_{j \in A} f_{ijr} = \sum_{m=1}^{M_i} b_{imr} z_{im} \quad \forall\ i \in A \backslash \{n+1\}\ , \quad r = 1,\ldots,R \tag{12}$$

$$\sum_{j \in A} f_{jir} = \sum_{m=1}^{M_i} b_{imr} z_{im} \quad \forall\ i \in A \backslash \{0\}\ , \quad r = 1,\ldots,R \tag{13}$$

$$f_{ijr} \leq \bar{M}_{ijr} x_{ij} \qquad \forall\ i,j \in A\ , \quad r = 1,\ldots,R \tag{14}$$

The objective function (2) is minimizing the starting time of the ending task, hence minimizing the makespan of the project, and (3) initializes the beginning task to time equals zero. (4) enforces the precedence constraints. (5) and (6) are logical constraints, stating either task i is completed before task j begins, or vice versa, but not both; and if task i is completed before task j begins, and task j is completed before task k begins, then task i is completed before task k begins. (7) states that each task must be executed in exactly one mode. (8) - (10) ensure $y_{ijm} = x_{ij} \times z_{im}$. (11) ensures that if task i is completed before task j begins, then the start time of task j should be at least the start time of task i plus the duration of task i. (12) and (13) are the node balance constraints for the flow of resources, indicating the flow of resource into and out of each task is equal to the resource requirement. Notice that the outflow constraint (12) is not enforced for task $n+1$, and in-flow constraint (13) is not enforced for task 0. Finally, (14) ensures the resources flowing forward, i.e., task i is allowed to pass on resources to task j only if task i is completed before task j begins.

In our case study, the number of tasks differ for each stage subproblem. For all stage problems there are two renewable resources, $R = 2$, where the first resource corresponds to the threads, and the second resource corresponds to the number of parallel tasks. Hence b_{im1} equals the thread allocations of all tasks except 0 and $n+1$ in each mode, determined as discussed in Section 3; and b_{im2} equals to 1 for all tasks except 0 and $n+1$ and for all modes. The values used in this study as big-M parameters in constraints (11) and (14) are $\hat{M}_{ij} = \hat{M}$ for all i and j, where $\hat{M}$ equals the sum of the longest task durations of each task (i.e. $\sum_{i \in A} \max_m p_{im}$), as an upperbound on the makespan; and $\bar{M}_{ijr} = \bar{M}_r$ for all i and j equals the total resource availability of resource r.

It should also be noted that the proposed formulation in (2) - (14) is also adaptable to settings with fluctuating resource availabilities through time, with discrete steps. Consider the availability of a resource over time, shown in Figure 4.

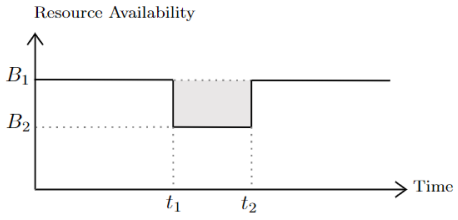

Figure 4. An example for varying resource availability over time

To model the decrease between t_1 and t_2, it is possible to set up the optimization model with a fixed level of resource B_1, and assume there exists a dummy task (shown in gray shade) that starts at t_1, with a duration of $t_2 - t_1$, and a resource requirement of $B_1 - B_2$. We may safely assume that for this dummy task, the requirement of other resources (if any) is zero.

Following this line of thought, the availability of each resource r ($b_{0,1,r}$) is set to the maximum

availability over time; and each decrease of each resource as a rectangular block is modelled as a dummy task as described above. Let A_d be the set of such dummy tasks; and $A' = A \cup A_d$. The tasks in A_d have predetermined start times σ_i, 0 as their sole predecessor, $n+1$ as their sole successor; and they all have a single mode.

The varying resource model can be formulated as (2) - (14) using A' instead of A, and an additional constraint

$$S_i = \sigma_i \qquad \forall\, i \in A_d. \qquad (15)$$

Although this new formulation appears to be suitable for dealing with *FixedTasks* in our problem, it is not directly applicable. The main reason for this is that although the start times of *FixedTasks* are fixed, the durations are not. In other words, although t_1 values are predetermined, t_2 changes during the execution. Underestimating t_2 values would cause the model to utilize a resource that has not yet been released by the *FixedTask*. Therefore, for our specific implementation, we stick with the resource reduction strategy explained in Section 3.

4.2. Simulated annealing

The lack of access to a commercial solver throughout the bank's implementation requires the utilization of a heuristic strategy for the solution tool's deployment. For this purpose, we preferred Simulated Annealing algorithm, which has been used as an approximation tool for RCPSP and MRCPSP.[26–28] Simulated Annealing (SA) is a neighborhood-based local search algorithm that, unlike greedy search algorithms, allows itself to explore solutions with worse objective function values. At every iteration a neighbor solution is randomly selected. If this new solution has a better objective, then the search jumps on to the next point. If the new solution does not have a better solution, then the search either remains on the current solution, or jumps on to the new solution with a probability that is a function of the difference in the objective values and a concept called the *temperature*. Just like the actual annealing process applied to metals, SA algorithm starts with high temperature, more likely to jump to other solutions and explore the region; and gradually decreases the temperature to ensure convergence.

The search is conducted for C chains, where each chain is initialized at different solutions. In each chain, there are S steps, where the number of neighbors generated n_s, and the temperature T is constant at each step. After each step, n_s is increased and T is decreased. The general framework of the algorithm is given in Figure 5.

A very important concept in SA is the concept of neighborhood among the solutions. The neighboring solutions should be easy to generate. With minor changes in the solution vector, one should be able to create feasible solutions. Hence, it is of the utmost importance that the project execution schedules have an appropriate representation. The variables in Section 4.1 (S, x, z, y, and f) constitute a representation; a subset of these variables may also constitute a representation (such as S and z). However, minor modifications in these vectors are very likely to lead to infeasible solutions. In our algorithm, we use a representation that consists of two lists, similar to the work in.[27]

```
Find initial solution x_0, and set x_{best} = x_{current} = x_0
and f_{best} = f_{current} = f(x_0)
For C chains
    Initialize T and n_s
    For S steps
        For n_s neighbors
            Generate a neighbor x of x_{current}
            Delta = f(x) - f_{current}
            If Delta < 0
                x_{current} = x, f_{current} = f(x)
                If f_{current}<f_{best}
                    x_{best} = x_{current}, f_{best} = f_{current}
            Else
                With probability e^{-Delta /T}
                    x_{current} = x, f_{current} = f(x)
        Decrease T, increase n_s
```

Figure 5. A Generic SA Algorithm

- *Activity List* indicates the precedence-feasible order of tasks to be scheduled.

- *Mode List* indicates the mode in which the tasks are executed. It is the equivalent of z_{im} variable defined in Section 4.1.

For a solution represented by *Activity List* and *Mode List*, the corresponding objective function value $f(x)$, i.e. the makespan of the EOD process, is calculated using Serial Schedule Generation Scheme (SGS).[14]

We will illustrate how this algorithm works by going back to the example in Figure 1. Assume for the given problem, we have the *Activity List* $[0, 1, 3, 2, 4]$ and the *Mode List* is $[1, 2, 1, 1, 1]$. This *Activity List* means that we will start by scheduling task 0 to the first, followed by task 1, task 3, task 2 and task 4. The *Mode List* indicates we will execute tasks 0, 2, 3, 4 in mode 1, and task 1 in mode 2; therefore, we have the resource usages $[5, 2, 2, 4, 5]$ and durations $[0, 6, 3, 3, 0]$. The steps of the algorithm are illustrated in Figures 6 - 10. We start by fixing task 0 to the earliest resource and precedence feasible time $S_0 = 0$.

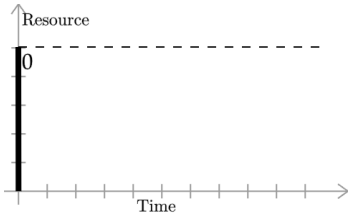

Figure 6. Step 1 of SGS

The earliest resource and precedence feasible time for task 1 is $S_1 = 0$.

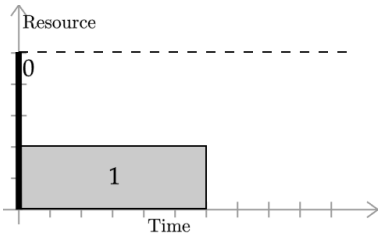

Figure 7. Step 2 of SGS

The earliest resource and precedence feasible time for task 3 is $S_3 = 6$.

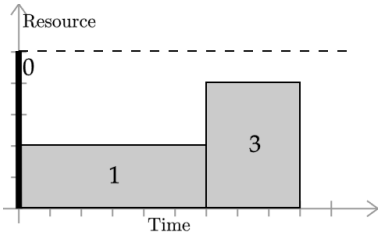

Figure 8. Step 3 of SGS

The earliest resource and precedence feasible time for task 2 is $S_2 = 0$.

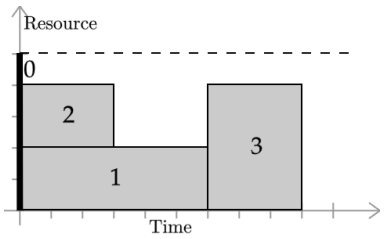

Figure 9. Step 4 of SGS

The earliest resource and precedence feasible time for task 4 is $S_4 = 9$.

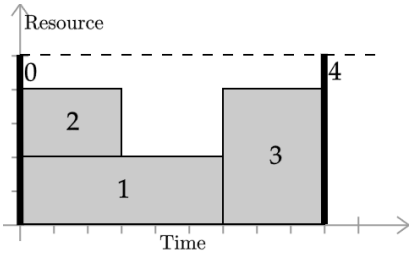

Figure 10. Step 5 of SGS

SGS with *Activity List* $= [0, 1, 3, 2, 4]$ and *Mode List* $= [1, 2, 1, 1, 1]$ calculates the makespan as 9. Notice that not all such orderings correspond to unique schedules. Different orderings, such as *Activity List* $= [0, 1, 2, 3, 4]$, may lead to the same schedule.

SA algorithm requires the selection of a random neighbor at every iteration. We perform this random selection similar to the work in[28] as follows:

(1) A task i is randomly selected, at position x in *Activity List*. Let x_p be the maximum position among the predecessors of i, and let x_s be the minimum position among the successors of i. In other words, x_p and x_s are the positions of the closest predecessor and successor of i in *Activity List*.
(2) Task i is assigned to a random position x' (selected uniformly) between x_p and x_s simply by shifting all tasks in *Activity List* between positions x and x' left if $x < x'$, or right if $x' < x$.
(3) A random mode for task i is selected (uniformly) among all possible M_i mode options, and *Mode List* is updated accordingly.

Following our example, let's say the randomly selected task is $i = 2$, with its position $x = 4$. The maximum position among the predecessors of task 2 is $x_p = 1$ (the only predecessor task 0 is at position 1), and the minimum position among

the successors of task 2 is $x_s = 5$ (the only successor task 4 is at position 5). We choose a random position among the possible options $\{2, 3, 4\}$, let's say we choose the new position $x' = 2$. We place task 2 in position 2, and shift the *Activity List* accordingly to get $[0, 2, 1, 3, 4]$. Since task 2 only has a single mode, we have to choose 1 again for its mode, and therefore *Mode List* remains as $[1, 2, 1, 1, 1]$. Given the solution *Activity List* $= [0, 1, 3, 2, 4]$ and *Mode List* $= [1, 2, 1, 1, 1]$; the new solution *Activity List* $= [0, 2, 1, 3, 4]$ and *Mode List* $= [1, 2, 1, 1, 1]$ is considered a neighbor.

Notice that this random neighbor selection scheme may result in one of the following four outcomes: (1) *Activity List* is changed, *Mode List* remains the same, (2) *Mode List* is changed, *Activity List* remains the same, (3) both *Activity List* and *Mode List* are changed, and (4) both *Activity List* and *Mode List* remain the same. In the case of outcome (4), the random neighbor selection scheme repeats itself instead of returning identical lists.

For details on selecting the parameters for the SA algorithm described in Figure 5, the reader is referred to.[27] In our application, we used $C = 2$ chains, where T and n_0 are initialized as 10 and 80 respectively. Each chain consists of $S = 5$ steps, at each step n_s neighbor comparisons are made (which may or may not result in updating the current solution), and at the end of each stage s, T is updated as $T/4$, and n_s is updated as $n_s \cdot (1 + s)$. These values are chosen as a result of preliminary experiments, to minimize the duration of the search while maintaining solution quality.

5. Results

We wish to emphasize that the SA algorithm serves as a solution to our problem that does not require a commercial solver while achieving the optimal solution for this specific data set. Our experiments confirmed that for these specific instances, the solutions obtained by SA indeed match the optimal solution found by Gurobi. Therefore, no further research has been conducted for a better heuristic alternative, as SA is both fast enough for this implementation, and reaches the optimal solution.

This section provides an analysis of the existing EOD system and the impact of the MRCSP on the proposed EOD process optimization. Our analysis focuses on three primary aspects: the impact of optimizing only the scheduling decisions, the impact of scheduling and reassigning resource usages (i.e., mode selection), and the impact of resource availabilities.

As mentioned in Section 3, the EOD process consists of five stages. The number of tasks in each stage and the average duration of the stages are shown in Table 2, demonstrating the size and impact of the problem at hand. Since stages 3 and 5 are small both in terms of the number of tasks and duration; meaningful improvement can be observed only in stages 1, 2, and 4.

Table 2. Stage Information

	Number of Tasks	Average Duration (minutes)
Stage 1	70	26.225
Stage 2	80	109.95
Stage 3	10	7.575
Stage 4	60	103.1
Stage 5	10	0.425

The first analysis concerns the impact of scheduling, i.e., the order of tasks, on the makespan of the project. The goal of this analysis is not to determine how many threads to allocate to each task; rather use the same thread allocations that have been used, keep the thread usage and duration constant for each task, and only solve for the order of execution of tasks. We adapt our model to solve this problem simply by setting the number of modes of each task to one.

Due to the variability of task durations each day, the scheduling decision, which must be made before execution, cannot rely on actual daily durations. Instead, for the scheduling problem we use estimate values for the task durations. Due to the daily deviation in task durations, the estimated makespan and the realized makespan of each stage may differ significantly.

To provide a fair comparison, we randomly selected seven days from the range of the daily job logs (i.e., historical data). For each day, we

- solve the optimization problem for each stage, where all tasks are restricted to a single mode (which is the existing setting), and the durations are the estimated task durations based on the median of observations in the daily logs,
- obtain an order of execution from the optimization problem,
- use the realized task durations of that day to calculate the makespan of the stage if it had been executed according to the new schedule.

The makespan calculated in this manner is referred to as *New Schedule*. This value is compared with the actual stage durations observed in the daily logs, referred to as *Actual Schedule*. Tables 3 - 5 report these values for all 7 days for Stages 1, 2, and 4; and the values provided are

in minutes. Since in daily job logs, the start and end times of tasks are given as time stamps in minutes, the values in these tables are prone to ±1 minute error.

Table 3. Impact of Scheduling in Stage 1

	Actual Schedule	New Schedule
Day 1	24	24
Day 2	22	22
Day 3	21	21
Day 4	27	26
Day 5	20	20
Day 6	9	9
Day 7	8	9

Table 4. Impact of Scheduling in Stage 2

	Actual Schedule	New Schedule
Day 1	103	107
Day 2	109	106
Day 3	93	101
Day 4	104	98
Day 5	90	94
Day 6	85	88
Day 7	86	86

Table 5. Impact of Scheduling in Stage 4

	Actual Schedule	New Schedule
Day 1	86	86
Day 2	92	86
Day 3	94	86
Day 4	92	84
Day 5	72	71
Day 6	78	62
Day 7	91	72

We observe three different behaviours for the three stages. In stage 1, seen in Table 3, the actual order of execution and the optimized order of execution are the same; hence stage 1 requires no assistance for scheduling. In stage 4, seen in Table 5, the system significantly benefits from optimized scheduling most of the time (such as days 2, 3, 4, 6, and 7), and in the remaining days the two schedules yield almost identical results. The impact of the scheduling problem in stage 2, seen in Table 4, is uncertain. Due to the discrepancy between the actual task durations and the estimated durations used in the optimization problem, the resulting schedule yields a shorter makespan on some days (such as days 2 and 4) and a longer makespan on others (such as days 1, 3, 5, and 6). Paired t-tests conducted to these three groups of schedules, indicate that the mean duration of Stages 1 and 2 can be assumed to be identical, with corresponding p-values of 1 and 0.457, respectively. However the mean durations of Stage 4 are statistically different with a p-value of 0.021.

Next, we analyze the impact of multiple modes in each stage. As studied in critical path analysis, shortening an arbitrary task does not necessarily shorten the whole process. Only those on the critical path have a direct impact on the project duration. Similarly, providing shorter or longer modes for arbitrary tasks does not have a directly foreseeable outcome. Furthermore, with different precedence structures in different stages, the impact of multiple modes is expected to differ between stages.

For the analysis whose results are displayed in Table 6, we solve the single mode resource-constrained scheduling problem to get an estimated duration of each stage without multiple modes; and the multi-mode resource constrained scheduling problem to get an estimated duration and report the percentage improvement in makespans. In other words, the values in Table 6 are calculated as *(multi-mode duration - single mode duration) / (single mode duration) · 100*. The most significant enhancement occurred in Stage 1, with an 18% reduction, while the least effect was noted in Stage 2, with an 8% reduction in makespan.

Table 6. Improvement in Stage Makespans with Multiple Modes

	Improvement
Stage 1	-18.1%
Stage 2	-7.9%
Stage 4	-15.2%

Note that for this analysis, we have to use the estimated makespans given by the optimization problem, rather than the actual observations, such as the ones that were used in Tables 3 - 5. The reason behind this is that although we have actual observations for the single-mode makespans, we do not have actual observations for the multi-modes (i.e. historical data only has the default thread usages, we don't have any observations of the task under the proposed thread count selection). Therefore, for fairness, we base the comparison on the optimization problem results using the estimations.

The values in Tables 2 - 6 are generated under the existing system parameters with 200 threads and 15 parallel tasks available. To make the EOD process more efficient, one might also consider investing in the infrastructure to increase resource availability. However, before this investment, we would like to observe the potential benefit of increasing the resources and understand how efficiently we are using the existing resources.

Tables 7 - 9 report the estimated makespan of stages 1, 2, and 4 respectively, when MRCSP is solved with varying resource availabilities. The

number of tasks allowed to run in parallel {10, 12, 15, 18, 20} are shown in the columns, and the number of threads {150, 180, 200, 250} are shown in the rows. Tables 7 and 9 reveal that Stages 1 and 4 are not utilizing the full capacity of 200 threads and 15 parallel tasks. The same makespans can be achieved with values as low as 150 threads and 10 parallel tasks. Stage 2 however, as shown in Table 8, significantly benefits from increasing the thread count up to 200, with a slower rate of improvement from 200 to 250 threads; and as the other stages it is not impacted by the change in the number of parallel tasks in the range of 10-20.

Table 7. Estimated Stage 1 durations in minutes with varying resource availability

	10	**12**	**15**	**18**	**20**
150	27.0	27.0	27.0	27.0	27.0
180	27.0	27.0	27.0	27.0	27.0
200	27.0	27.0	27.0	27.0	27.0
250	27.0	27.0	27.0	27.0	27.0

Table 8. Estimated Stage 2 durations in minutes with varying resource availability

	10	**12**	**15**	**18**	**20**
150	167.0	167.0	167.0	167.0	167.0
180	150.7	150.7	150.7	150.7	150.5
200	112.2	112.2	112.2	112.2	112.2
250	110.8	110.8	110.8	110.8	110.8

Table 9. Estimated Stage 4 durations in minutes with varying resource availability

	10	**12**	**15**	**18**	**20**
150	75.3	75.3	75.3	75.3	75.3
180	75.3	75.3	75.3	75.3	75.3
200	75.3	75.3	75.3	75.3	75.3
250	75.3	75.3	75.3	75.3	75.3

6. Conclusion

Banks use EOD processes to generate and deliver reports to stakeholders for regulatory monitoring, business users, and downstream stream systems. EOD processes enable the company to meet its objectives, such as minimizing the time required to complete all tasks, maximizing customer satisfaction, maximizing profit, or minimizing costs. The EOD process may last longer and more challenging to coordinate in corporate systems when applications are extensive, distributed, and utilize diverse technologies, as seen in microservices architectures. Consequently, conventional batch processing techniques may prove inadequate for managing EOD batch processes in these systems, necessitating the development of novel EOD algorithms.

This paper presents a case study of an EOD process in a Turkish bank, where the objective is to reduce the duration of the EOD process by optimizing the resources allocated to each task and their order of execution. We model this problem as a multi-mode resource-constrained project scheduling problem, and present a flow-based formulation of MRCPSP. We also provide a SA heuristic for implementation purposes and verify that SA reaches the optimal solution for this particular problem setting. The motivating purpose of the proposed scheduler is to determine the execution order of EOD tasks, along with the number of threads to be used during execution to minimize the makespan of the EOD process.

It is also shown that the scheduler can be used for analytical purposes. Considering various methods such as code optimization, parallelization, data partitioning and load balancing that can be used to optimize the EOD process, the scheduler in this paper can be used to detect the importance of scheduling and crashing tasks, along with the sufficiency of the infrastructure for optimizing the EOD process. Our analyses have effectively identified three distinct structures and requirements for the three most critical stages of the EOD process. The existing infrastructure is shown to be sufficient for two of the stages, whereas the third stage benefits from increasing the available threads. For one of the stages, merely solving for the optimal schedule and not changing the thread allocations can give shorter makespans. All three stages benefit from using multiple modes. In other words, the flexibility of changing the thread allocations and changing task durations lead to shorter makespans in all three stages.

The work in this research can be further developed to deal with Fixed Tasks in smarter ways; i.e. dynamic resource availabilities. Furthermore, the flow-based MRCPSP can be tested for larger instances, and for problem sizes that exceed the capacity of commercial solvers, smarter algorithms can be devised.

Acknowledgments

The authors would like to thank Aykut Yüksek and Yalçın Yılmaz from Fibabanka for their contributions.

Funding

This research has been funded by Fibabanka, with grant number E.D.VR-21-02416.

Conflict of interest

The authors declare that they have no conflict of interest regarding the publication of this article.

Author contributions

Conceptualization: All authors
Formal analysis: Ezgi Karabulut Türkseven, Ezgi Genç
Methodology: Ezgi Karabulut Türkseven
Project administration: Ilgın Şafak
Software: Ezgi Karabulut Türkseven, Ezgi Genç
Writing – original draft: Ezgi Karabulut Türkseven, Ilgın Şafak
Writing – review & editing: Ilgın Şafak

Availability of data

All the data is included in the paper.

References

1. Cetin K, Tuzkaya G, Vayvay O. A mathematical model for personnel task assignment problem and an application for banking sector. *Int J Optim Control Theor Appl (IJOCTA)* 2020;10:147-158. https://doi.org/10.11121/ijocta.01.2020.00825
2. Malcolm D, Roseboom J, Clark C, Fazar W. Application of a technique for research and development program evaluation. *Oper Res* 1959;7:646-669. https://doi.org/10.1287/opre.7.5.646
3. Koné, O., Artigues C, Lopez P, Mongeau M. Comparison of mixed integer linear programming models for the resource-constrained project scheduling problem with consumption and production of resources. *Flex Serv Manuf J* 2013;25:25-47. https://doi.org/10.1007/s10696-012-9152-5
4. Christofides N, Alvarez-Valdés R, Tamarit J. Project scheduling with resource constraints: A branch and bound approach. *Eur J Oper Res* 1987;29:262-273. https://doi.org/10.1016/0377-2217(87)90240-2
5. Artigues C, Michelon P, Reusser S. Insertion techniques for static and dynamic resource-constrained project scheduling. *Eur J Oper Res* 2003;149:249-267. https://doi.org/10.1016/S0377-2217(02)00758-0
6. Alvarez-Valdarés R, Tamarit J. Heuristic algorithms for resource-constrained project scheduling. In: *Advances In Project Scheduling*. Elsevier; 1989: l34-143.
7. Boctor F. Some efficient multi-heuristic procedures for resource-constrained project scheduling. *Eur J Oper Res* 1990;49:3-13. https://doi.org/10.1016/0377-2217(90)90116-S
8. Özdamar L, Ulusoy G. A local constraint based analysis approach to project scheduling under general resource constraints. *Eur J Oper Res* 1994;79(2):287-298. https://doi.org/10.1016/0377-2217(94)90359-X
9. Ulusoy G, Özdamar L. Heuristic performance and network/resource characteristics in resource-constrained project scheduling. *J Oper Res Soc* 1989;40:1145-1152. https://doi.org/10.1057/jors.1989.196
10. Lawrence S. Resource constrained project scheduling-A computational comparison of heuristic scheduling techniques. In: *Technical Rep*. Graduate School of Industrial Administration And Carnegie-Mellon Univ; 1985.
11. Kolisch R. Efficient priority rules for the resource-constrained project scheduling problem. *J Oper Manag* 1996;14:179-192. https://doi.org/10.1016/0272-6963(95)00032-1
12. Özdamar L, Ulusoy G. An iterative local constraints based analysis for solving the resource constrained project scheduling problem. *J Oper Manag* 1996;14:193-208. https://doi.org/10.1016/0272-6963(95)00015-1
13. Ulusoy G, Ozdamar L. A constraint-based perspective in resource constrained project scheduling. *Int J Prod Res* 1994;32:693-705. https://doi.org/10.1080/00207549408956961
14. Kolisch R, Hartmann S. *Heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis*. Springer; 1999. https://doi.org/10.1007/978-1-4615-5533-9$_7$
15. Kolisch R, Hartmann S. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *Eur J Oper Res* 2006;174:23-37. https://doi.org/10.1016/j.ejor.2005.01.065
16. Hartmann S, Briskorn D. An updated survey of variants and extensions of the resource-constrained project scheduling problem. *Eur J Oper Res* 2022;297:1-14. https://doi.org/10.1016/j.ejor.2021.05.004
17. Li K, Willis R. An iterative scheduling technique for resource-constrained project scheduling. *Eur J Oper Res* 1992;56:370-379. https://doi.org/10.1016/0377-2217(92)90320-9
18. Valls V, Ballestin F, Quintanilla S. A hybrid genetic algorithm for the resource-constrained project scheduling problem. *Eur J Oper Res* 2008;185:495-508. https://doi.org/10.1016/j.ejor.2006.12.033
19. Lee J, Kim Y. Search heuristics for resource constrained project scheduling. *J Oper Res Soc* 1996;47:678-689. https://doi.org/10.1057/jors.1996.79
20. Talbot F. Resource-constrained project scheduling with time-resource tradeoffs: The nonpreemptive case. *Manag Sci* 1982;28:1197-1210. https://doi.org/10.1287/mnsc.28.10.1197
21. Patterson J, Slowinski R, Talbot F, Weglarz J. An algorithm for a general class of precedence and resource constrained scheduling problems. *Advances In Project Scheduling* 1989;3-28. https://doi.org/10.1016/B978-0-444-87358-3.50005-5

22. Sprecher A, Hartmann S, Drexl A. An exact algorithm for project scheduling with multiple modes. *Oper Res Spektrum* 1997;19:195-203. https://doi.org/10.1007/BF01545587
23. Hartmann S, Drexl A. Project scheduling with multiple modes: A comparison of exact algorithms. *Networks: An International Journal* 1998;32:283-297. https://doi.org/10.1002/(SICI)1097-0037(199812)32:4¡283::AID-NET5¿3.0.CO;2-I
24. Sprecher A, Drexl A. Multi-mode resource-constrained project scheduling by a simple, general and powerful sequencing algorithm. *Eur J Oper Res* 1998;107:431-450. https://doi.org/10.1016/S0377-2217(97)00348-2
25. Boctor F. A new and efficient heuristic for scheduling projects with resource restrictions and multiple execution modes. *Eur J Oper Res* 1996;90:349-361. https://doi.org/10.1016/0377-2217(95)00359-2
26. Boctor F. Resource-constrained project scheduling by simulated annealing. *Int J Prod Res* 1996;34:2335-2351. https://doi.org/10.1080/00207549608905028
27. Bouleimen K, Lecocq H. A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *Eur J Oper Res* 2003;149:268-281. https://doi.org/10.1016/S0377-2217(02)00761-0
28. Jozefowska J, Mika M, Rozycki R, Waligora G, Weglarz J. Simulated annealing for multi-mode resource-constrained project scheduling. *Ann Oper Res* 2001;102:137-155.
29. Mori M, Tseng C. A genetic algorithm for multi-mode resource constrained project scheduling problem. *Eur J Oper Res* 1997;100:134-141. https://doi.org/10.1016/S0377-2217(96)00180-4
30. Özdamar L. A genetic algorithm approach to a general category project scheduling problem. *IEEE Trans Syst Man Cybern C (Applications And Reviews)* 1999;29:44-59. https://doi.org/10.1109/5326.740669
31. Alcaraz J, Maroto C, Ruiz R. Solving the Multi-Mode Resource-Constrained Project Scheduling Problem with genetic algorithms. *J Oper Res Soc* 2003;54:614-626. https://doi.org/10.1057/palgrave.jors.2601563
32. Weglarz J, Jozefowska J, Mika M, Waligora G. Project scheduling with finite or infinite number of activity processing modes-A survey. *Eur J Oper Res* 2011;208:177-205. https://doi.org/10.1016/j.ejor.2010.03.037

Ezgi Karabulut Türkseven *works as an assistant professor at Sabanci University Industrial Engineering program. Her BSc and MSc degrees are from Boğaziçi University Industrial Engineering department, and she received her PhD in OR from Georgia Institute of Technology. With her background in operations research, her recent work focuses on machine learning and optimization.*
https://orcid.org/0000-0001-8642-7715

Ezgi Genç *is a Senior Analyst at Dcipher Analytics in Istanbul, Turkey. She received both her BSc (2019) and MSc (2021) degrees in Industrial Engineering from Sabanci University. During her studies, she gained practical experience through internships at NielsenIQ, Allianz, and BMC. Since joining Dcipher Analytics in 2021, she has been involved in conducting in-depth data analysis, contributing to international projects, and deriving actionable insights. Her main research interests include statistics, machine learning, and AI applications.*
https://orcid.org/0000-0001-6815-4207

Ilgın Şafak *is an Expert Researcher at Fibabanka's R&D Center in Istanbul, Turkey and a Postdoctoral Researcher at the University of Jyväskylä, Finland. She received her MSc and PhD degrees in 2006 and 2013 respectively, in Electrical and Electronics Engineering from Hacettepe University in Turkey. She worked as a Product Manager in Product Development and Innovation at Mastercard Payment Transaction Services Turkey in Istanbul, Turkey between 2013 and 2016. She worked as a Manager in Product Development and Innovation at Mastercard in Purchase, NY from 2016 until 2018. She worked as a Solution Engineer from 2019 until 2020 at HAVELSAN, Turkey. She has been working at Fibabanka since 2020. She published 27 articles and has filed 11 patent applications. Her research interests are in cybersecurity, distributed ledger technology, artificial intelligence/machine learning, and the internet of things.*
https://orcid.org/0000-0002-2788-7276

This work is licensed under a Creative Commons Attribution 4.0 International License. The authors retain ownership of the copyright for their article, but they allow anyone to download, reuse, reprint, modify, distribute, and/or copy articles in IJOCTA, so long as the original authors and source are credited. To see the complete license contents, please visit http://creativecommons.org/licenses/by/4.0/.