RAIRO-Oper. Res. 56 (2022) 2721–2749
https://doi.org/10.1051/ro/2022094

RAIRO Operations Research
www.rairo-ro.org

REAL TIME READ-FREQUENCY OPTIMIZATION FOR RAILWAY MONITORING SYSTEM

Mahdi Jemmali[1,2,3,*], Loai Kayed B. Melhim[4] and Fayez al Fayez[1]

Abstract. Trains have a key role in transporting people and goods with the option of moving from source to destinations by passing through several stations, with time-based features like date scheduling and known arrival times, which makes time a critical factor. The main challenge here, is to ensure that the train trip or train schedules are not affected or delayed in any way during the whole train trip; by giving the control unit in the railway system, the required time to process requests regarding all collected data. This an NP-hard problem with an optimal solution of handling all collected data and all service requests by the control unit of the railway system. Operational research will be used to solve this problem by developing many heuristics to deal with tasks of real-time systems, to produce a significant time optimization in the railway systems. To solve this problem, the proposed approach employs optimization by adapting 22 heuristics based on two categories of algorithms, the separated blocks category algorithm and the blocks interference category algorithm. The proposed approach receives data from many different sources at the same time, then collects the received data and save it to a data base in the railway system control unit. Experimental results showed the effectiveness of the developed heuristics, more over the proposed approach minimized the maximum completion time that was elapsed in handling the received requests.

Mathematics Subject Classification. 90C90, 90C59, 90C27.

Received December 28, 2019. Accepted June 11, 2022.

1. Introduction

Nowadays, time optimization is one of the most desirable criteria. Customers require mobility, smart phones, wireless communications, various applications for each performed task or presented services, smart cities, and many other digital needs due to the developed modern world. In this world people use means of transportation for many reasons, to move from one point to another, to travel, to carry goods and for many other reasons. Modern life imposes new standards into customer's transportation; pushing transportation process to a different

Keywords. Railway system, optimization, monitoring system, railway track, real-time system, heuristics, simulation.

[1] Department of Computer Science and Information, College of Science, Majmaah University, AL-Majmaah 11952, Saudi Arabia.

[2] MARS Laboratory, University of Sousse, Sousse, Tunisia.

[3] Department of Computer Science, Higher Institute of Computer Science and Mathematics, Monastir university, Monastir 5000, Tunisia.

[4] Department of Health Information Management and Technology, College of Applied Medical Sciences, University of Hafr Al Batin, Hafr Al Batin 39524, Saudi Arabia.

*Corresponding author: m.jemmali@mu.edu.sa; mah_jem_2004@yahoo.fr

level, destination arrival is no more the major concerns of these customers; arrival time is the major concern to most of these customers.

Among the different means of transportation, this study will consider trains transportation, as trains have a key role in transporting people and goods. Moreover, many big cities rely on train transportation; due to the train features that fit the varying needs and provide the dynamic requirements of these cities. Trains have the option of moving from source to destinations by passing through several stations, with a pre known fixed features like date schedule and arrival times, such features are of interest to many customers. All of these features are time-based features; hence time is considered a critical factor in this study.

The railway system, in this context, is composed of many subsystems that are integrated to ensure the effective operation of the railway, which is reached by achieving a set of goals such as meeting all journeys' time table, efficient goods transportation, passengers safety, efficient customer service,... etc. Therefore, the train system receives the required data to ensure the achievement of those goals through the monitoring system. The monitoring system uses sensors, cameras, and other tools to receive data and various requests from railway subsystems. All these requests and data are sent to the control unit to accomplish the required tasks as soon as possible. Therefore, the delay in processing any of the requests or data will lead to weaknesses or deficiencies in the performance of the subsystem to which these requests or data belong, and this is an undesirable thing. To avoid this, this work presents a framework that will provide the necessary time, required for the control unit to complete all requests and data sent without any delay.

To ensure that trains meet the given time-based features, all factors and variables regarding the train system should be considered. One of the suggested solutions when time is critical is to utilize the capabilities of scheduling algorithms [12, 46], it's worth to mention that scheduling here, is not date scheduling, it is how the railway system addresses large volume of services and requests within a limited time and resources [45]. There are many challenges and obstacles facing railway systems [14, 15], some of these challenges will be addressed in this paper like, data handling and receiving and how scheduling is used to handle these problems to derive the seriousness of each, to specify the impact of these problems on the train trip or schedule. The priority here is to handle these problems so that the train trip or schedule is not affected or delayed in any way during the whole train trip. To avoid any latency, train options are limited to delaying or stopping the trip, because the train has a fixed route on its railway. Time needed to process the received data, is the major concern of this research. Data can be received from many sources: service requests, monitoring systems, visual-based systems, sensors, detection systems or any other sources. Once the data is received, it is collected and saved into a database for processing and for further analysis by the control unit of the railway monitoring system. To meet the required features of the railway system and the critical time issue; time and computing limitations should be considered, to give the control unit, the required time to handle the largest number of collected data. These limitations can be managed by optimization, through the adaptation of several heuristics developed for the parallel machine problem, which is an NP-hard problem with an ideal solution of handling all collected data and all service requests by the control unit.

The rest of this paper will be organized as follows, Section 2 presents the related work while Section 3 discusses problem presentation. Formulating the optimization problem will be presented in Section 4, the next section explains the scheduling problem. The sequencing-based train algorithm will be addressed in Section 6. Discussion of the experimental results will be presented in Section 7. Finally, the conclusion will be presented in Section 8.

2. Related studies

Several researchers worked about scheduling in railways. The researchers in [30] addressed the parallel jobs scheduling problem. the considered preferences were related to release dates and identical parallel machines. The main objective of their work was to minimize the makespan and to perform a parallel work, with a set of constraints on a group of identical machines, this work is achieved by an online model that handles sequentially the arrived tasks based on the obligation of the schedule before the next task is known.

While authors in [13] explain the case of an NP-hard problem where the number of tasks and machines are already known. The assigned tasks are restricted by single task per single machine and pre-known processing time. The assigned tasks are executed by both a meta-heuristic and an exact algorithm to achieve the goal of minimizing the maximum completion time. The discussed results by the authors concluded the ability of the developed algorithm to solved optimally the majority of the given instances efficiently. The remaining of the instances were solved optimally by the exact algorithm developed by Haouari and Jemmali [18].

Researchers in [31] used cost minimal schedule with column generation-based solution algorithm to explore the effect on cost when scheduling-fairness is increased. In this research, the authors addressed the problem of distributing highest possible fairness, undesirable tasks among train drivers. The considered constraint was to cover completely all train movements without violating minimal increase in cost work regulations. The authors claimed that schedule fairness was significantly improved with only slight increases in schedule cost.

In the same manner researchers in [44] presented a solution of the crew rescheduling problem based on re-timing with frequent train disruption, by using column generation techniques combined with Lagrangian heuristics. To encounter any delay or a budget loss when any disruption occurs, the available resources related to that trip at that time are rescheduled. The presented algorithm is based on the rescheduling of the train driver duties. The authors claimed that the used algorithm showed better results than the used classical approaches.

The use of identical parallel machines to schedule parallel jobs with release dates, were treated by Li and Zhang [35]. The authors discussed the solution of two problems in their work. The first problem considers two similar machines with speeds of 1 and s where ($s > 1$). While the second problem considered the m identical machines problem. The main objective in this research was, how to minimize the makespan and the jobs arrival time. The authors showed a competitive ratio for on-line LPT algorithm related to the first problem and for a lower bound related to the second problem.

Other researchers used the weaknesses, the strengths, the volatility and the robustness parameters of a train schedule in the presence of the delay. For example, Burdett et.al presented an analytical approach that handles unusual specific behavior, to determine an appropriate action plan or to modify a schedule in advance [11]. In addition, the adopted approach can quantitatively derive the delay cost. A case study was given to declare the main features of the given approach.

To minimize makespan for the identical parallel-machine scheduling problem, authors in [33] developed a cuckoo search algorithm (ICSA). A heuristic approach was developed using model operator with heuristic procedure that is based on the pairwise exchange neighborhood to generate the required cuckoos. Experimental results showed a better performance when comparing with different existing algorithms.

Optimal makespan was discussed in [42] where the authors solved the Resource-Constrained Project Scheduling Problem based on meta-heuristics. To improve time and solutions quality, hybrid techniques were employed by many researchers like [16], in this research, the authors solved combinatorial optimization problems (COPs) by integrating machine learning and algorithms. The results indicate a remarkable improvement in the term of computational time and quality of results compared with random operators.

Researchers in [2] utilized operational research ccc minimize the job's maximum completion time, the authors classified the proposed heuristics into two classes, the resolution of a subset-sum problem with knapsack problem and the dispatching rule. The proposed heuristics were developed to fit and to obtain, the exact solution of the railway scheduling problems. The data captured by railway track sensors will be saved in railway monitoring system database, the developed algorithms utilize the read frequency rate by reading the maximum number of saved data within a predefined time limit.

For the object of cost-effective maintenance, the authors in [43] constructed a problem taxonomy from the results obtained after reviewing railway track-related maintenance and scheduling plans in the literature. In the same approach, the authors in [8] presented an opportunistic preventive maintenance policy with the goal of minimizing the total maintenance cost. The proposed policy executes a steady-state genetic algorithm to search for a maintenance threshold that will decide when to perform the preventive tamping maintenance. The authors stated that considering the proposed policy may reduce the performance cost by 46%. The automatic inspection of railway performed by a track running robot is presented by Pradeep *et al.* [40] to monitor the railway for any railway defects. The proposed approach uses a Wi-Fi camera to send a live recording of the defect location, the required directions to locate the exact location of the defected parts are also sent to the inspection team. The group of factors that affect the railway maintenance planning processes were presented by Kovenkin and Podverbnyy [32]. The authors in [6] used Bayesian approach to build a model that can expose the actual defect rate and the probability of not locating any defects in order to optimize the railway track maintenance process. A different approach was presented by Movaghar and Mohammadzadeh [38] where the authors utilize Bayesian framework to generate data regarding uncertainties in railway track degradation model, the results presented by the authors reveal how the proposed model can affect the restricted budget and the limited resources for preventive maintenance processes scheduling.

Current railway systems consist of a set of subsystems; each subsystem is dedicated to handle different operation management functions. Previous studies in the literature present scheduling problem in the railway systems. Most of these studies proposed solutions that consider railway subsystems, for example train timetables problem was solved by researchers in [45], while train schedules problem was solved by Donzella *et al.* [15] and Burdett and Kozan [11]. For the railway crew scheduling, many solutions were proposed by Jütte *et al.* [31] and Veelenturf [44]. While the solution for obstacle detection problem was proposed by Lan *et al.* [34] and the railway tracks detection and turnouts recognition problem were discussed, and solutions were proposed by Qi *et al.* [41]. To the best of our knowledge the only solutions for improving railway system efficiency were presented by Dong *et al.* [14] in which the authors proposed a solution for bridging the gap in the information exchange between the subsystems in high-speed railway systems. This solution proposes that the huge amount of sent data will be executed by the control unit part of the presented approach without any delay. While the proposed solution in this research focuses on, how the received data will be managed and executed by the control unit of the railway monitoring system before any delay or cancelation of the train trip. The other study was presented by al Fayez *et al.* [2]. Unlike the previous studies, the proposed solution which is an extension of the work presented by al Fayez *et al.* [2] minimizes the maximum time required by the control unit to complete the received requests, this allows the control unit to gain more time to achieve integration between all railway subsystems, the control unit must coordinate between these subsystems to ensure safe and reliable railway systems, which is the main objective of this research.

The algorithms used in several different scheduling problems can be utilized to expand and enhance the studied problem [1, 5, 7, 20, 22, 23, 27, 28].

The proposed algorithms can be improved and adopted to the problems studied in [4, 9, 10, 17, 37]. Several applications of scheduling problems in real life can be exploited to enhance the proposed algorithms [3, 21, 24–26, 29, 36].

3. Problem presentation

In this study the focus is on the method of handling the received data from the many sensors that are placed along the railway track. Various data are collected to guarantee the arrival of trains within a fixed time at their destinations. Indeed, railway could be subject to damages, fires, incidents, floods, and other rail obstacles like rocks, sand, trees or animals that influence the regular operation of trains. In order to avoid any disruption or latency of the train trips, it is crucial for the railway track sensors to detect and to provide the required data to the control unit of the railway system, before train reaches the location of these obstacles. The provided data are collected in the control unit that handles and extracts the required information to provide the required interest within an appropriate time. The treatment of this data requires processing time which is critical in this case. This data is modeled as tasks, which are modeled as processing time and the problem is modeled as a parallel machine problem. This problem is NP-hard and we are willing in this study to propose and develop suitable heuristics to solve the addressed problem within an acceptable time.

3.1. Notation

The railway monitoring system relies on a set of sensors, distributed on the railway tracks and on the train itself, the used notation to describe these sensors is as follows, Temperature: Te, Humidity: Hu, Fog: Fo, Wind: Wi and Image from located sensor: Im. The objective of these sensors is to continuously capture data and forward it to the control unit of the railway monitoring system. For the data sent by the sensors (Te, Hu, Fo, Wi) the related information is directly inferred from the received data, while for the Im sensor, the system executes a function called $ImageProc()$ to handle and to analyze the received images. The input of this function is Im. The output of $ImageProc()$ will be values assigned to the set of variables: Ob, An, Tu and Sd, with the following details:

– Ob is the variable related to the obstacle detection. The values of Ob variable will be $Ob = 1$ if an obstacle detected in the received image and $Ob = 0$ otherwise.
– An is the variable related to animal detection. The values of An variable will be, $An = 1$ in case an animal is detected in the received image and $An = 0$ otherwise.
– Tu is the variable related to a turnout detection. The values of Tu variable will be, $Tu = 1$ if a turnout is detected in the received image and $Tu = 0$ otherwise.
– Sd is the variable related to sand detection. The values of Sd variable will be, $Sd = 1$ if sand is detected in the received image and $Sd = 0$ otherwise.

Denoted by:

– $H_{Im} = (Te, Hu, Fo, Wi, Im)$: data vector sent to control unit.
– $H = (Te, Hu, Fo, Wi)$: $H_{Im} \setminus Im$.
– $V(Te, Hu, Fo, Wi, Ob, An, Tu, Sd)$: variation vector.
– n_s: sensors number.
– i: sensor index.
– S: sensor.
– t_s: the starting time of the trip.
– t_a: train arrival time of the trip.
– $X(S_i)$: the position of the sensor S_i.
– t_{S_i}: estimated train arrival time at sensor S_i.
– $X_t(T)$: the position of the train T at time t.
– $sp(t)$: speed of the train at time t (supposing that speed between S_i and S_{i+1} is constant).
– $freq$: sensor's data sending rate in (minutes).
– n_{pr}: number of processors.
– P: set of processes.
– Pr: set of processors.
– p^r: processor index.

The total number of the received data sent by sensor S_i when the train arrives is: $N_{rd} = (t_a - t_s) \times freq$.

3.2. Significant variation vector

In the monitoring system, the control unit receives frequently the sensors' data. The vectors $H_0(S_i)$ and $H_t(S_i)$ are instances of vector H at time = 0 and at time = t, respectively. The control unit handles and analyzes the received data after that the derived information will be values that will be compared with the saved initial values $H_0(S_i)$ for each sensor. For the monitoring system to identify any change in the state of railway system at time t, it should compare $H_0(S_i)$ with $H_t(S_i)$ at time t for the sensor (S_i). The comparison process between the saved initial values and the values sent by sensor S_i at time t produces a value change, which will be used to detect any change in the monitored subsystem status. The vector of the variation values at time t is denoted by $V_t^r(S_i)$:

$$V_t^r(S_i) = |H_0(S_i) - H_t(S_i)|. \tag{3.1}$$

In practice, if the variation in $V_t^r(S_i)$ is not remarkable, it will be discarded by the control unit and will not be considered in the generated instructions.

Example 3.1. In $H_0(S_4)$ the saved temperature is 32. But the temperature in $H_t(S_4) = 34$. The temperature increased by 2 degrees yielding a variation of 2. This variation is not significant because 2 degrees, is not expected to have any impact on the railway system different components. So, the control unit will not consider this variation *i.e.* the control unit will not perform any process for this variation.

To avoid unnecessary processes, the proposed approach is designed with a predefined threshold for each variable. The corresponding vector of all threshold values is denoted by $H_{th}(S_i)$. By using $H_{th}(S_i)$, it will be faster for the control unit to calculate significant variations easily, resulting in a fewer computations and higher performance. The resulted significant variation values will be stored in a new vector called the significant variation vector, denoted by $V_t^s(S_i)$ at time t. So, for $V_t^s(S_i) = 0$ this means that there are no variations and the S_i location at time t is under normal situation. The vector $V_t^s(S_i) = (Te, Hu, Fo, Wi, Ob, An, Tu, Sd)$ contains 8 elements. Each element stores the variation of the corresponding variable.

We denoted by $[V_t^s(S_i)]_j$ the jth element for the vector $V_t^s(S_i)$ and by $[H_{th}(S_i)]_j$ the jth element for the vector $H_{th}(S_i)$. For example, $[H_{th}(S_i)]_4$ denotes the threshold of the 4th element in the vector $H_{th}(S_i)$. It should be noted that, this approach will consider a fixed value of $[H_{th}(S_i)]_4$, and it will be equal to 0. Choosing $[H_{th}(S_i)]_4 = 0$ means that the proposed approach will consider any change in $[H_{th}(S_i)]_4$ *i.e.* consider any change in the captured image, even if the change was so small.

The calculation of $V_t^s(S_i)$ is performed by the function *sig_var_vector*() as shown in Algorithm 1.

Algorithm 1. Function *sig_var_vector*().

```
1:  for ($j = 1$ to $j = 4$) do
2:      if ($[V_t^r(S_i)]_j \leq [H_{th}(S_i)]_j$) then
3:          $[V_t^s(S_i)]_j = 0$
4:      else
5:          $[V_t^s(S_i)]_j = [V_t^r(S_i)]_j - [H_{th}(S_i)]_j$
6:      end if
7:      $[V_t^s(S_i)]_5 = Ob$;
8:      $[V_t^s(S_i)]_6 = An$;
9:      $[V_t^s(S_i)]_7 = Tu$;
10:     $[V_t^s(S_i)]_8 = Sd$;
11: end for
```

To obtain the values of the variables Ob, An, Tu and Sd, the sensor sends an image to the control unit, then the control unit executes $ImageProc()$ based on image processing techniques, $ImageProc()$ will calculate the variation between the stored $H_0(S_i)$ values and the received $H_t(S_i)$ new values. The result of $ImageProc()$ is the new deviated values of the variables Ob, An, Tu and Sd.

Example 3.2. The same values of $H_0(S_4)$ and $H_t(S_4)$ given in Example 3.1, will be used in this example: In $H_0(S_4)$ the temperature was 32. But in $H_t(S_4) = 34$. The threshold of $[H_{th}(S_4)]_1 = 8$. Applying the function *sig_var_vector*() will have $[V_t^s(S_4)]_1 = 0$.

This work considers only the significant variation vector, for all data readings. For every time t values with significant values of $V_t^s(S_i)$ will be stored in DC table as a (Received Data) in the database. Figure 1 shows the process from the stage of sending the sensor's data to the stage of storing it in the database.

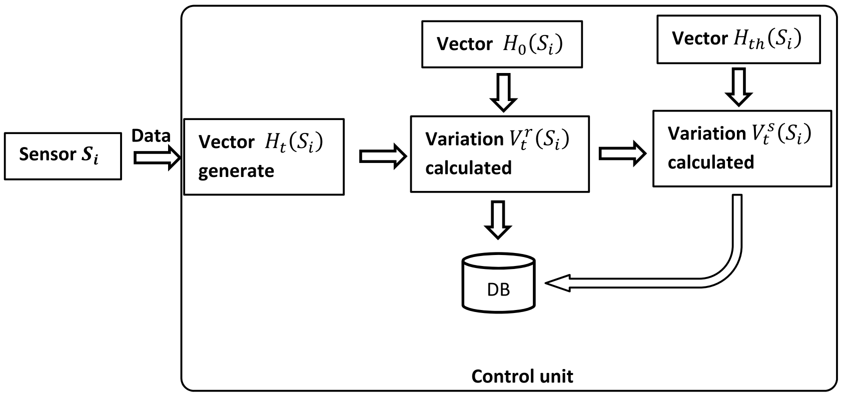
Sensor S_i
Data
Vector $H_t(S_i)$ generate
Vector $H_0(S_i)$
Variation $V_t^r(S_i)$ calculated
Vector $H_{th}(S_i)$
Variation $V_t^s(S_i)$ calculated
DB
Control unit

FIGURE 1. $V_t^s(S_i)$ storage.

4. Formulating optimization problem

The variations of all $V_t^s(S_i)$ values stored in the DC, will be checked by the control unit. The system is expected to be free of problems when the variations value is 0 and the control unit has no processes to perform. However, if any of the elements in $V_t^s(S_i)$ has a value greater than 0; the control unit will begin to produce the proper instructions required to handle any problem that is specified by the values of $V_t^s(S_i)$. These instructions will be denoted by I_i, where i is an integer and I_i can be as:

- I_1: send SMS to the monitoring administrator.
- I_2: send a voice alert to the driver of train.
- I_3: call the maintenance service.
- I_4: call the fireman.
- I_5: call electricity service.
- I_6: call police.
- I_7: execute a special program of image processing to detect the seriousness of an obstacle (in the case of obstacles).
- I_8: execute a special program of image processing to detect the seriousness of animal detection (in the case of animal).
- I_9: execute a special program of image processing to detect the seriousness of turnouts (in the case of railway turnouts).
- I_10: execute a special program of image processing to detect the seriousness of sand (in the case of sand).

For each instruction I_i there is a corresponding processing time denoted by prc_i. The monitoring system will execute one or more set of the predefined instructions for all non-null elements of vector $V_t^s(S_i)$. The monitoring system utilizes the corresponding stored data to generate in advance a suitable set of instructions that suits each variable variation. For example, for sensor S_i, if the received data $V_t^s(S_i)$ requires the attention of the processes I_3, I_6 and I_7 then the needed processing time to run the selected instructions is given as $p_i = prc_3 + prc_6 + prc_7$.

As shown in Figure 3, N_{rd} is the total number of the data that are sent by all sensors from t_s to t_a. In this work, data sent by (S_i) at time t_j are grouped in the same time slot SD(j). Where:

- j is the index that represents data sending frequency for all sensors, $j \in \{1, \cdots, N_{rd}\}$.
- SD(j) the period related to j.
- p_i^j all instructions processing time that are defined by the system and related to sensor i at period SD(j).

Figure 2 presents an illustration of the above process.

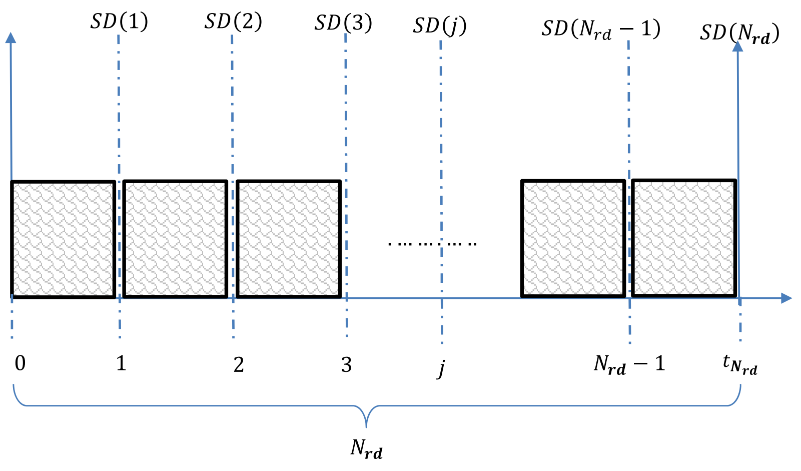

FIGURE 2. The total number of saved data in DC table from t_s to t_a is N_{rd}.

Example 4.1. Assume that we have at SD(3), $V_t^s(S_4) = (1,0,0,3,1,1,0,1)$. $Te = [V_t^s(S_4)]_1 = 1$ *i.e.* indicates a sudden high increase in temperature, the system should signal fire-fighters, this process is linked with the instruction I_4. For $Wi = [V_t^s(S_4)]_4 = 3$ an indication high risk of wind speed, the system must interfere by sending SMS to the monitoring administrator and alert the train driver, this process is linked to instructions I_1 and I_2. For $Ob = [V_t^s(S_4)]_5 = 1$, an indication of obstacle detection on the railway track in the position $X(S_4)$, this process is linked with instructions I_1, I_3 and I_7. For $An = [V_t^s(S_4)]_6 = 1$, an animal detected on the railway track in the position $X(S_4)$, this process is linked with instructions I_1, I_3 and I_8. For $Sd = [V_t^s(S_4)]_8 = 1$, an indication that there is serious amount of sand on the railway track, the system responds by calling maintenance section, which is related to instructions I_3 and I_10.

The total processing time at time t is: $p_4^3 = prc_4 + prc_1 + prc_2 + prc_1 + prc_3 + prc_7 + prc_1 + prc_3 + prc_8 + prc_3 + prc_10$.

Remark 4.2. For each sensor S_i, at period SD(j), the system will group all the called instructions then calculate the total processing time. The grouped instructions will be replaced by a single process denoted by P_i^j with a processing time given by p_i^j. In Example 4.1, time calculations are performed for one sensor S_4. Now, suppose that there were 3 other sensors that have values in the vector $V_t^s(S_i)$ *i.e.* the system will execute several processes in the same period, a process for each sensor with a total of 4 processes each of them has its own processing time p_i^j.

The railway system has to perform all the processes of the previous period SD($j-1$) to retrieve data from the DC table to execute the latter processes SD(j). The required time to complete the execution of all assigned processes $n_{\mathrm{SD}(j)}$ of the period SD(j) is given by $C_{\max}^j$. To calculate $C_{\max}^j$, we start by $j = 1$ until SD(j) = SD$_{\max}$, which is the last period before the end of the trip or the train arrival. The main purpose here is to increase the rate of the reading frequency from the DC table.

Proposition 4.3. *Maximizing read-frequency is equivalent to minimizing $C_{\max}^j$ at each period* SD(j).

Proof. If the value of $C_{\max}^j$ at period SD(j) was minimized then the minimization amount will be utilized to read another data from the DC table, which in turn increases the number of read processes at period SD(j). So, for each $C_{\max}^j$ minimization, we will have an increase in the number of read processes thus an increase in the read frequency rate. □

Table 1. The 3 periods data sending instances.

i	S_1	S_2	S_3	S_4	S_5	S_6	S_7
SD(1)	50	20	40	25	0	0	0
SD(2)	0	0	0	25	10	80	0
SD(3)	0	0	0	0	0	0	30

Remark 4.4. The optimal solution in such problems is to read all the DC table saved data *i.e.* $\text{SD}_{\max} = N_{rd}$. Such problems is known as scheduling problems, where the set of processes is represented by $P = \{P_i^j, 1 \leq i \leq n_s, 1 \leq j \leq \text{SD}_{\max}\}$ and the related processing times by p_i^j. For a fixed SD(j) values the system handles a scheduling problem. Thus, the proposed system has an $\text{SD}_{\max}$ scheduling problem that should be solved, this problem type is known as blocks scheduling.

The control unit in the proposed approach has a number of identical processors, which allows to consider the scheduling problem as an identical parallel processor scheduling problem. The objective of this problem is to reduce the makespan $C_{\max}$, which is the time required to finish executing all given processes for the most loaded processor. To complete the identical parallel processors scheduling problem, the proposed solution must address this question: will the system be capable of reading data from DC table before the execution of all previous processes? For the proposed system to handle this question, it must select between the following categories of algorithms:

– Separated blocks category algorithm (SB): The case for which the system can't read data from DC table before it executes all processes of the current period.
– Blocks interference category algorithm (BI): The case for which the system can read data from DC table before it completes the execution of all the processes of the current period.

Example 4.5. Let us discuss a case where the control unit has 3 processors, applying SB algorithm will produce the following results $t_a = 10\,\text{h:}10\,\text{min}$, $t_s = 10\,\text{h:}13\,\text{min}$, $freq = 1$ data/min So, $N_{rd} = (t_a - t_s) \times frq = 3 \times 1 = 3$ So, for this case the best solution will be to have 3 data retrieval from 10 h:10 to 10 h:13 *i.e.* we can retrieve all data from DC. Consider the below instances (time in seconds) (Table 1):

Figure 3 illustrates how the system schedules the given processes by applying SB algorithm, where it can be noticed that $\text{SD}_{\max} = 2$.

Example 4.6. To achieve a change in schedule 1 that was described in Example 4.5 in order to minimize $C^1_{\max}$. Executing the enhanced algorithm will produce the results given in Figure 4.

Figure 4 shows that $\text{SD}_{\max} = 3$. it is easy to obtain $\text{SD}_{\max} = \text{SD}(3) = 3$. Thus, the used algorithm in this example to schedule processes at period SD(1) in Figure 4 gains better results than the algorithm cited in Figure 3. Besides that, the case of this example is the best one because $\text{SD}_{\max} = \text{SD}(3) = 3$.

5. Scheduling problem

To solve the scheduling problem based on identical parallel processors problems, many heuristics will be developed and adapted for the case when the period SD(j) is fixed and the concerned problem is denoted by Pb^j. The function $RetrivalDC(\text{SD}(j))$ that will retrieve data from DC table based on the period SD(j) is called by the control unit, the called function will provide the number of the corresponding processes and the table that has all the processes P in the period SD(j).

The control unit calls a function named $H(J, X)$ to execute the heuristic H described by the algorithm above, where $H(J, X)$ produces the appropriate schedule during the period SD(j) for the set of processes P and

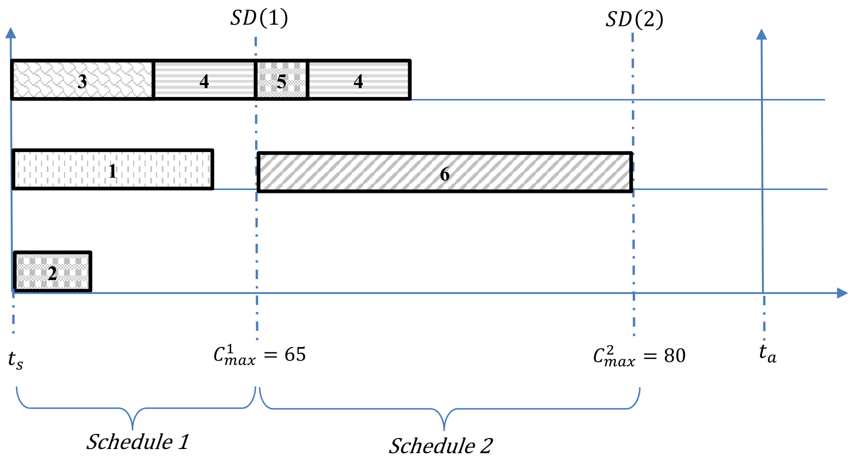

FIGURE 3. 3 processors scheduling with $\mathrm{SD}_{\max} < N_{rd}$.

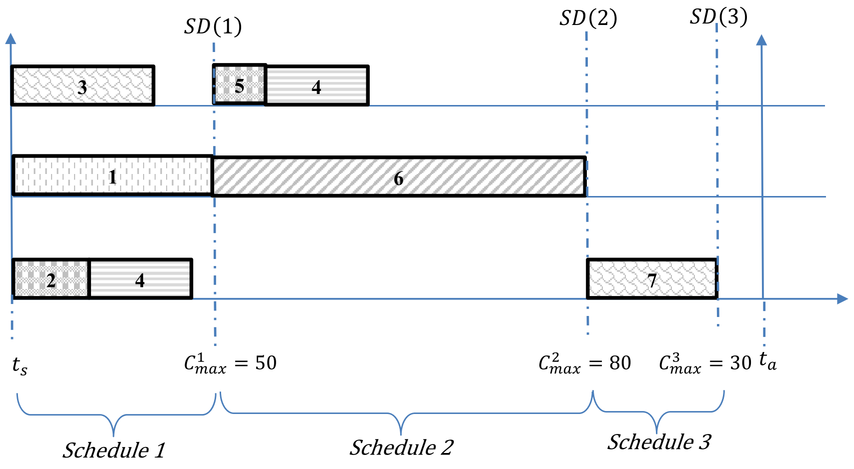

FIGURE 4. Best case with $\mathrm{SD}_{\max} = N_{rd}$.

the number of related processes X. Execution of the function $H(J, X)$ provides processes assignment and the corresponding $C^j_{\max}$. Where $P(\mathrm{SD}(j))$ is the set of processes P^j_i that will be sent during the period $\mathrm{SD}(j)$ and will be scheduled to the available processors.

The following Section 5.1 is devoted to explaining the heuristics already cited in literature review regarding parallel machines, which will be utilized in this paper, by the developed algorithms for the studied problem.

5.1. Parallel processors heuristics

Many parallel processors algorithms will be utilized to develop new heuristics that are expected to solve the presented problem. The first algorithm is the longest processing time and is denoted by LPT. This algorithm is based on sorting the given processes according to the non-increasing order of their processing time. The complexity of the LPT algorithm is $O(n \log n)$. The second algorithm is the dispatching rule SPT, in this algorithm the processes are sorted based on the non-decreasing order of their processing time. The complexity of the SPT algorithm is $O(n \log n)$. While the third algorithm is the subset-sum based-heuristic (SS). The complexity of the SS algorithm is $O(n)$. The chosen approach for SS is the utilization of the greedy algorithm to solve iteratively different subset-sum problems (SSP) that is denoted by Pb_l with $\{l = 1, \cdots, n_{pr} - 1\}$ and presented in equation (5.1). The processing time of the process ps is denoted by p_{ps}.

$$Pb_l : \begin{cases} \min \sum_{ps \in W_l} p_{ps} y_{ps}, \\ \text{subject to} \sum_{ps \in W_l} p_{ps} y_{ps} \geq L(W_l, n_{pr} - l + 1), \end{cases} \tag{5.1}$$

with $y_{ps} \in \{0, 1\}$ for all $ps \in W_l$. Where $W_1 = P$ and $W_{l+1} = W_l \setminus Ps_l$ where Ps_l is the returned processes when applying the subset-sum algorithm for Pb_l and where $l = \{1, 2, \cdots, n_{pr} - 1\}$. $L(S, l)$ is a lower bound of the problem with a resulted makespan for the instance that is defined by $l \leq n_{pr}$ processors and a subset of processes $W \subset P$. As a result, the system will continue assigning processes to the first processor until reaching L on Pb_1. The remaining processes will be distributed over the rest of the available processors, this will generate the second problem Pb_2 that will be solved by the new SSP until the limit L is reached and so on [19]. A pseudo-polynomial based on dynamic programming algorithm detailed in [39] is used to solve the subset sum problem.

The heuristic multi-start subset-sum (MSS) described in [19] will be derived based on solving the Pb^j problem. The MSS algorithm can be explained as follows. As shown in [19] the $P_2||C_{\max}$ will be reformulated as a subset-sum problem. Based on this proposition, a multi-start local search method will be implemented, this method requires a repetitive solving of a two-processor problem. Given a feasible schedule σ and assuming that all processors are indexed such that $C_1 \leq, \cdots, \leq C_{n_{pr}}$. Define a binary variable y_{ps} with a value 1 if the process ps is assigned to Pr_1, and a value 0 otherwise. Then, $P_2||C_{\max}$ will be interactively solved by applying SSP.

The idea developed in (MSS) will be used in the multi-start knapsack heuristic (MSK), a difference of how the problem is being solved in each iteration. MSK uses knapsack problem (KP) to solve the given problem for each iteration rather than using the SSP [18]. Since the KP problem can be solved efficiently in pseudo-polynomial time. The complexity of the MSS algorithm is $O(n^2)$. The complexity of the knapsack $O(nW)$ where, n is the number of items and W is the capacity of knapsack. The MSK heuristic utilize the knapsack problem with multi restart.

In this paper, we propose five categories of algorithms. Each category utilizes the parallel machines heuristics presented earlier. The first category is the separated blocks algorithms. The second one is the reverse-interference based algorithms. The third category is the fictitious-processes based algorithms. The fictitious-processes with the MSS heuristic for each iteration will represent the fourth category. The fifth category is the interference processes-blocks based algorithms. The first category is presented by the separated blocks category, while the categories 2, 3, 4 and 5 are presented by the blocks interference category.

5.2. Separated blocks category algorithm

For the separated blocks algorithm (SB), the constraint to consider when retrieving data from DC table is the following: "For the current period, the system can't retrieve data from DC table before the execution of all processes" as explained in Example 4.1 above. Calculate the corresponding $C^j_{\max}$ for each period SD(j), where the total of $C_{\max}$ is the sum of all $C^j_{\max}$. So, $C_{\max} = \sum_{j=1}^{\mathrm{SD}_{\max}} C^j_{\max}$.

In this algorithm current time is returned by the function $timer()$ and the algorithm SB is structured as follows:

The disadvantage of the SB is the lose of opportunity to assign processes of the period SD$(j+1)$ to the available time slot in the period SD(j). This is because of the data retrieval constraint, which prevents reading from the DC table before processing all processes in SD(j).

In this paper, $H()$ represents one of the parallel processors algorithms described in the Section 5.1 and $H(P, X)$ is the related result, which is obtained by the heuristic H for the processes set P and the number of processes X.

Based on the five presented algorithms in Section 5.1, Algorithm 2 can be used by replacing $H(J, X)$ by one of the parallel processors algorithms presented in Section 5.1. Thus, we denoted by $U^{\mathrm{SB}}_{\mathrm{LPT}}$, $U^{\mathrm{SB}}_{\mathrm{SPT}}$, $U^{\mathrm{SB}}_{\mathrm{SS}}$, $U^{\mathrm{SB}}_{\mathrm{MSS}}$ and

Algorithm 2. Separated blocks algorithm.

```
1: Initialize $j = 1$; $t = 0$ ; $C_{\max} = 0$
2: while $(\mathrm{SD}(j) \leq N_{rd}$ and $t < t_a)$ do
3:     Call $RetrivalDC(\mathrm{SD}(j))$
4:     $X = n_{\mathrm{SD}(j)}$
5:     Calculate $P(\mathrm{SD}(j))$
6:     $C^{j}_{\max} = H(P, X)$.
7:     $j++$.
8:     $t = timer()$.
9: end while
10: $C_{\max} = \sum_{j=1}^{\mathrm{SD}_{\max}} C^{j}_{\max}$
11: Return $C_{\max}$.
```

$U^{\mathrm{SB}}_{\mathrm{MSK}}$ the values of algorithm SB replacing the function $H()$ by LPT, SPT, SS, MSS and MSK, respectively. The complexity of the separated blocks category algorithm depends on the $H(J, X)$.

5.3. Blocks interference category algorithm

For the blocks interference algorithm (BI), the constraint for data retrieval from the DC table is the following: "The system can retrieve data from DC table before the execution of all processes at the current period". This means, for the first block we try to utilize the SD(1) processors availability. After assignment of processes in SD(2) we use SD(2) processors availability by processes given in SD(3) and so on. The total $C_{\max}$ is calculated after scheduling of all processes.

The problem here, is how to utilize the processors availability between blocks $\mathrm{SD}(j)$ and $\mathrm{SD}(j + 1)$.

Remark 5.1. For each block or period $\mathrm{SD}(j)$ we apply a heuristic H_1 to schedule processes at this period. To merge two blocks $\mathrm{SD}(j)$ and $\mathrm{SD}(j + 1)$ we must apply a method that ensures merging. The fictitious-processes based algorithm given in Section 5.3.2 is one of the methods that ensure merging. For this latter algorithm, we have to choose one of heuristics H_2 that can be applied for merging. The heuristics applied for each block and the heuristics applied to merge blocks are not necessary the same. The question here is: to have a best solution, what is the best choice for H_1 and the best choice for H_2?

This study presents algorithms that are related to the blocks interference algorithm (BI). The complexity of the blocks interference category algorithm depends on the choice of H_1 and H_2.

5.3.1. Reverse-interference based algorithm

The procedures that are used in the reverse-interference based algorithm is described next. The completion time of the processor pr in the period $\mathrm{SD}(j)$ is denoted by C^{j}_{pr}. The given processes are sent to the procedure $NonDecr(tab)$ to be sorted by list tab in the non-decreasing order. However, $NonIncr(tab)$ is the procedure that arranges the given processes in the list tab in the non-increasing order. The reverse-interference algorithm (RI) is described in Algorithm 3.

Once we started the execution of the processes in the period $\mathrm{SD}(j)$ and at the moment when the data of the period $\mathrm{SD}(j + 1)$ arrives, we assign the processes of $\mathrm{SD}(j + 1)$ according to the remaining of processors availability, after the period $\mathrm{SD}(j)$. The availability will be calculated based on fixing the time C^{j}_{pr} for each processor pr related to period $\mathrm{SD}(j)$. So, the control unit supposes that all the processes of $\mathrm{SD}(j)$ period are scheduled, then the control unit takes advantage of the processors availability during the period $\mathrm{SD}(j)$. To utilize the availability in each period the control unit takes some processes of $\mathrm{SD}(j + 1)$ and allocates it to the available periods of $\mathrm{SD}(j)$. As shown in Figure 7 cited in Example 4.6 when the data of process 7 arrives, the control unit places this process on processor 3 during the period SD(2) before $C^{2}_{\max}$.

Algorithm 3. Reverse-interference algorithm.

```
1: Initialize $j = 1$; $t = 0$ ; $C_{\max} = 0$
2: while $(\mathrm{SD}(j) \leq \mathrm{SD}_{\max}$ and $t < t_a)$ do
3:     Call $RetrivalDC(\mathrm{SD}(j))$
4:     $X = n_{\mathrm{SD}(j)}$
5:     Calculate $P(\mathrm{SD}(j))$
6:     Calculate $C^j_{pr}$ $\forall pr\{1 \leq pr \leq n_{pr}\}$ applying $H(P, X)$.
7:     for $(pr = 1$ to $pr = n_{pr})$ do
8:         $tab_2[pr] = C^j_{pr}$.
9:         if $(j == 1)$ then
10:            $tab_1[pr] = tab_2[pr]$;
11:        end if
12:    end for
13:    if $(j \geq 2)$ then
14:        $NonDecr(tab_1)$
15:        $NonIncr(tab_2)$
16:        for $(pr = 1$ to $pr \leq n_{pr})$ do
17:            $tab_1[pr] = tab_2[pr] + tab_1[pr]$;
18:        end for
19:    end if
20:    $j{+}{+}$.
21:    $t = timer()$.
22: end while
23: $C_{\max} = \max_{1 \leq pr \leq n_{pr}} tab_1[pr]$
24: Return $C_{\max}$
```

Table 2. Fictitious-processes.

i	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9	S_{10}
SD(1)	100	20	40	40	0	0	0	0	0	0
SD(2)	0	0	0	0	10	20	10	10	10	15

For instruction 6 in Algorithm 3, we replace $H(J, X)$ by one of the parallel processors algorithms described in Section 5.1. Thus, we denoted by $U^{\mathrm{RI}}_{\mathrm{LPT}}$, $U^{\mathrm{RI}}_{\mathrm{SPT}}$, $U^{\mathrm{RI}}_{\mathrm{SS}}$, $U^{\mathrm{RI}}_{\mathrm{MSS}}$ and $U^{\mathrm{RI}}_{\mathrm{MSK}}$ the values returned by the Algorithm 3 replacing the function $H(J, X)$ by LPT, SPT, SS, MSS and MSK, respectively.

The complexity of the Reverse-interference based algorithm depends on the choice of $H(J, X)$.

5.3.2. *Fictitious-processes based algorithm*

For this algorithm, instead of making the reverse of the completion time between increasing and decreasing, in order to enhance the final completion time, we apply a dispatching rule, which considers that all processes are executed on one processor as one fictitious process. Indeed, for SD(1) we apply a $H(P, X)$. For the second period SD(2), we apply the same $H(P, X)$. Now, the connection between SD(1) and SD(2) is as follows, for SD(2) we assume that each completion time C^2_{pr} of each processor pr is considered as a fictitious process fp^2_{pr}. Applying a dispatching rule to schedule all fp^2_{pr} on processors while taken into consideration the period SD(1). In general, for the period SD(j) the fictitious processes denoted by fp^j_{pr} has the processing time $p(fp)^j_{pr}$. The following example gives an illustration of the fictitious-processes constitution.

Example 5.2. Let $n_s = 10$ and $n_{pr} = 3$. Assume that the details of the sending data are as shown in Table 2.

After applying a given heuristic to schedule processes in Table 2 to all processors, the results are given in Figure 5.

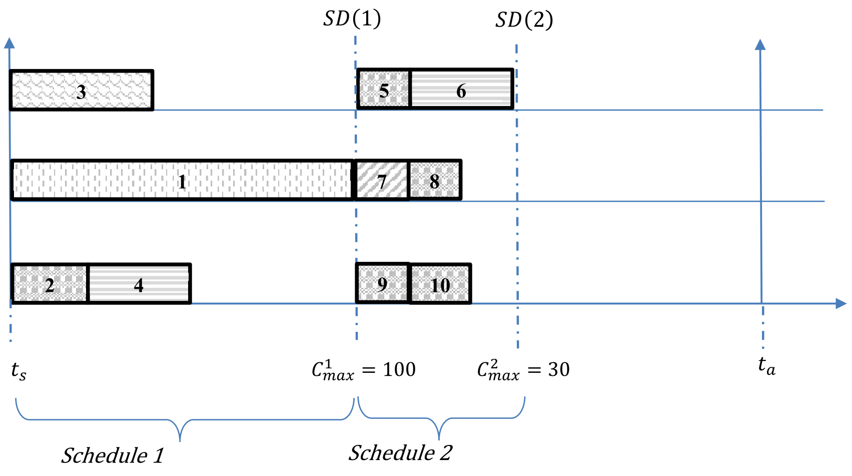

FIGURE 5. Fictitious-processes constitution.

From Figure 5, during the period SD(2) on processor 1, there are processes 5 and 6. These processes, will be treated as one fictitious process fp^2_1. The sum of processing time of processes 5 and 6 will constitute the processing time of $p(fp)^2_1$. Therefore, $p(fp)^2_1 = p^2_5 + p^2_6 = 30$. Applying the same method for processes 7 and 8 on processor 2, we have $p(fp)^2_2 = 20$ and for processes 9 and 10 on processor 3 we have $p(fp)^2_3 = 25$, (please refer to Table 2, to check processes 5, 6, 7, 8 and 9 processing time).

Now, the processes fp^2_1, fp^2_2 and fp^2_3 will be scheduled on all processors, taken into account the processes already assigned during SD(1) and the processing time $p(fp)^2_1$, $p(fp)^2_2$ and $p(fp)^2_3$.

Applying LPT rule, fp^2_1 will be assigned to the most available processor in SD(1) which is processor 1. So, the completion time on processor 1 will be 70. After that, the most available processor is processor 3, the largest fictitious processes is fp^2_3. We schedule fp^2_3 on processor 3 and the completion time of processor 3 will be 85. Next assign fp^2_2 to the most available processor which is processor 1 with minimum completion time of 70 comparing to processors 2 and 3 which have 100 and 85 as their completion time, respectively. Thus, the completion time of processor 1 will be 90. At this stage the maximum completion time is 100 in processor 3. Figure 6 illustrates the scheduling of fictitious-processes.

After constructing of fictitious-processes, scheduling of these processes will be performed by applying suitable heuristics. Thus, we denoted by $U^{\mathrm{FP}}_{\mathrm{LPT}}$, $U^{\mathrm{FP}}_{\mathrm{SPT}}$, $U^{\mathrm{FP}}_{\mathrm{RL}}$ and $U^{\mathrm{FP}}_{\mathrm{IRL}}$ the values returned by fictitious-processes based algorithms when applying LPT, SPT, RL and IRL, respectively. The RL is the randomized LPT and IRL is the iterative randomized LPT.

The complexity of the fictitious-processes based algorithm depends on the choice of $H(J, X)$.

5.3.3. Fictitious-processes with multi-subset based algorithm

For this algorithm, the first step is to construct all fictitious processes, then schedule these processes by using some of the proposed heuristics that were described in Section 5.3.2. The scheduling of these processes is the manner that is used by H_2 to merge two blocks SD(j) and SD($j+1$) described in Remark 5.1. For each block or period SD(j), we apply the heuristic MSS to schedule processes at each period. We denoted by $U^{\mathrm{FPM}}_{\mathrm{LPT}}$, $U^{\mathrm{FPM}}_{\mathrm{SPT}}$, $U^{\mathrm{FPM}}_{\mathrm{RL}}$ and $U^{\mathrm{FPM}}_{\mathrm{IRL}}$ the values returned by fictitious-processes based algorithm when we apply LPT, SPT, RL and IRL as H_1. While MSS will be applied as H_2.

The complexity of the fictitious-processes with multi-subset based algorithm depends on the choice of H_1 and H_2.

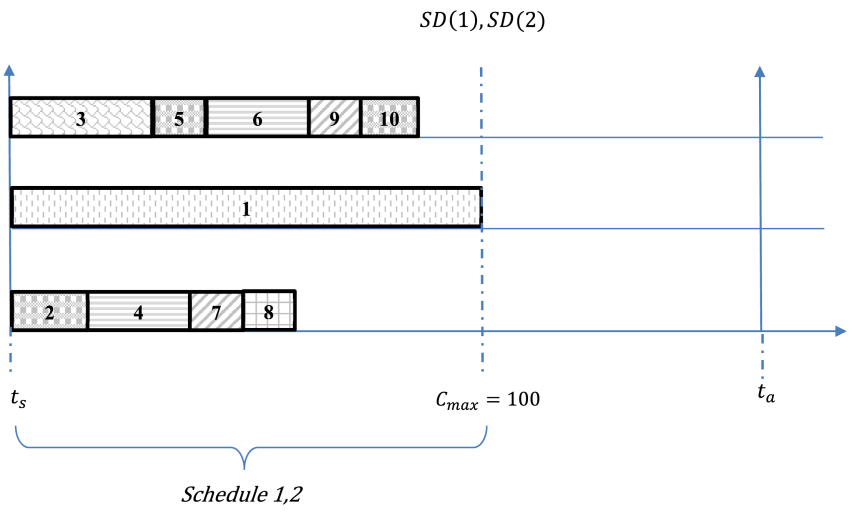

FIGURE 6. Fictitious-processes scheduling.

5.3.4. Interference Processes-blocks based algorithm

For this heuristic, instead of constructing of the fictitious processes, to ensure connection between periods; the system schedules processes of period SD$(j+1)$ with continuity of period SD(j). This means, the system schedules process by process of period SD$(j+1)$. Indeed, for the first period SD(1) we apply heuristic H_1 and for period SD(2), we apply the same heuristic, but we schedule process by process taking into account the scheduling made in period SD(1). In other words, heuristic H_1 selects the process to be scheduled on the most available processor after finishing the scheduling of processes in period SD(j).

We denoted by $U^{\text{IPB}}_{\text{LPT}}$, $U^{\text{IPB}}_{\text{SPT}}$, $U^{\text{IPB}}_{\text{RL}}$ and $U^{\text{IPB}}_{\text{IRL}}$ the values returned by interference processes-blocks based algorithm when we apply LPT, SPT, RL and IRL, respectively.

The complexity of the interference Processes-blocks based algorithm depends on the choice of H_1.

6. Sequencing-based train algorithm

This section presents the algorithm that describes the details of all running processes during a train trip, from the starting point till the train arrives at its final destination. For that purpose, we construct a sensors index, based on sensors positions $\{X(S_1) < X(S_2) < \cdots < X(S_{n_s})\}$. So the first sensor is the one nearest to the train passage. If the sensor stimulates an attention request in the control unit, this means that the variation value of vector $V^s_t(S_i)$ is not null, for which the algorithm must run a function that is responsible of estimating the needed time to complete the required intervention. This function is called $Maintenanc()$, the input of this function is the location state of the considered sensor. while $Maintenance(X(S_1))$ returns the estimated time of maintenance reserved to position $X(S_1)$ related to sensor 1.

$Maintenance(X(S_1))$ is the function that reads a file saved in the server and sent by the maintenance group, this file specifies the time required to finish maintenance process in position $X(S_i)$ in real time. In this case, when implementing the sequencing-based train algorithm in real time, we must consider this question, what are the train options if it reaches the problem location before maintenance is finished? What instructions, the control unit must execute to avoid latency or an accident? To answer these questions, the algorithm named: sequencing-based train algorithm, was proposed. Figure 7 shows the train tracking example when an intervention is required.

As Figure 7 shows, if maintenance time is greater than the time required for the train to arrive at the position of the sensor with the problem; then there are two choices:

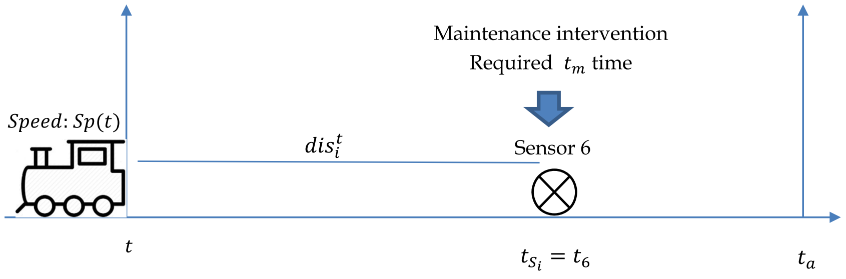

FIGURE 7. Train tracking detection.

– Decrease the train speed and announce a trip delay.
– Stop the train for a fixed time and announce a trip delay.

The remaining time to arrive at a sensor location will be denoted by $t_R = t_{S_i} - t$ (in minutes), as shown in Figure 7. If $t_m \leq t_R$ then the train can continue without any delay or any stopping. But if $t_m > t_R$ the control unit must send an alert to the driver, requesting a speed decreasing. The train delay time will be: $t_d = t_m - t_R$. The distance between the position of the train at time t and the position of sensor indexed i will be denoted by dis_i^t and will be calculated as: $dist_i^t = \frac{t_R \times Sp(t)}{60}$. While the new reduced speed that was proposed by the control unit after the maintenance alert, will be denoted by Sp_e where $Sp_e = \frac{dist_i^t \times 60}{t_m}$. Which means the train must travel with a speed (in worst case) Sp_e to arrive at time t_m with $t_m > t_R$. In the case of a delay, the control unit will activate a function that sends an alert to the driver with the following data (new speed, the modified arrival time considering the delay), this function will be denoted by $AlertDriver(Sp_e, t_d, t_a)$.

Based on the above analysis we propose the following heuristic:

7. Experimental results

This section presents the experimental results obtained by executing the implemented heuristics. The performance of the lower and upper bounds assessment is achieved after the implementing the developed heuristics in Microsoft Visual C++ (Version 2013). All experiments were executed on an Intel(R) Xeon(R) CPU E5-2687W v4 @3.00 GHz and 64 GB RAM workstation that has windows 10 with 64 bits operating system.

7.1. Test instances

The used instances are based on the selection of n_s, n_{pr}, M and $Class$. The selected values were as follows: 6 elements of $n_s \in \{3, 5, 10, 20, 50, 100\}$ and 4 elements of $n_{pr} \in \{2, 4, 6, 8\}$.

For a fixed number of sensors n_s, the total number of received data N_{rd} is randomized and does not depend on the number of the used sensors. However, the number of the received data in DC table is dependent on the number of sensors. For example, if we have 5 sensors, we can't have $N_{rd} = 12$ or $N_{rd} = 13$ because all sensors send in the same time different data. So, the number of received data in DC table is a multiple of the number of sensors. Thus, we have M times of n_s data received.

Therefore: $N_{rd} = n_s \times M$, where M is a multiplicator positive integer. $M \in \{2, 5, 10, 25, 50\}$: 5 elements. So, for example if $n_s = 5$, N_{rd} can be a multiple of 5 and M as this set: $N_{rd} = \{10, 25, 50, 125, 250\}$. So, for $n_s = 5$ the number of received data can be up to 250, this result is obtained when choosing multiplicator to be 50. So, the problem is complicated with large number of received data.

We generate 8 types of class instances of different processing times. These classes are based on the uniform distribution and normal distribution. The uniform distribution is denoted by $U[x, y]$ which gives a random number between x and y. While, the normal distribution is denoted by $N[x, y]$ with mean x and standard deviation y.

Algorithm 4. Sequencing-based train heuristic.

```
1: Initialize $j = 1$; $t = 0$;
2: while ($t < t_{\mathrm{SD}(j)}$) do
3:     $t = timer()$
4:     $RetrivalDC(\mathrm{SD}(j))$
5:     $SPJI(\mathrm{SD}(j))$
6:     while ($J(\mathrm{SD}(j)) \neq \emptyset$) do
7:         Alert to maintenance in $X(S_i)$
8:         $t_m = maintenance(X(S_i))$
9:         $t = timer()$
10:        $t_R = t_(S_i) - t$
11:        if ($t_m < t_R$) then
12:            continue;
13:        else
14:            $Sp_e = \frac{t_R \times Sp(t)}{t_m}$
15:            $t_d = t_m - t_R$
16:            $t_a = t_a + t_d$
17:            $AlertDriver(Sp_e, t_d, t_a)$
18:        end if
19:    end while
20:    $j++$;
21: end while
22: if ($\mathrm{SD}(j) \leq N_{rd}$ and $t < t_a$) then
23:     go to Step 2
24: else
25:     STOP
26: end if
```

The processing time in this paper is generated as follows:

- *Class* A, the p_i^j is in $U[1,5]$.
- *Class* B, the p_i^j is in $U[1,10]$.
- *Class* C, the p_i^j is in $U[5,10]$.
- *Class* D, the p_i^j is in $U[10,20]$.
- *Class* E, the p_i^j is in $U[10,30]$.
- *Class* F, the p_i^j is in $U[1,20]$.
- *Class* G, the p_i^j is in $N[5,2]$.
- *Class* H, the p_i^j is in $N[5,4]$.

For the variables $(n_s, M, n_{pr}, class)$, where n_s represents the number of sensors, M is the related multiplicator, n_{pr} is the number of processors and $class$ is the type of the generated processing time. The generated instances of processes were 10.

This type of generation resulted in a total number of instances equals to $6 \times 5 \times 4 \times 8 \times 10 = 9600$. The number of sent data is $n_s \times M$. For example, if $n_s = 3$ and $M = 7$, there are 21 sent data which addresses the value of N_{rd}.

7.2. Evaluation metrics

To assess the performance of the developed heuristics, several metrics are defined and presented as described next:

- A^* is the minimum value returned after the execution of all algorithms. A is the studied heuristic.
- GAP $= \frac{A - A^*}{A^*} \times 100$.

TABLE 3. Results of the separated blocks category algorithms.

	$U^{\text{SB}}_{\text{LPT}}$	$U^{\text{SB}}_{\text{SPT}}$	$U^{\text{SB}}_{\text{SS}}$	$U^{\text{SB}}_{\text{MSS}}$	$\boldsymbol{U^{\text{SB}}_{\text{MSK}}}$
Perc	38.3%	21.3%	51.5%	69.9%	**98.4%**
Time	0.005	0.005	0.023	1.687	**749.628**

Notes. The best results are given in bold.

TABLE 4. Behavior of GAP according to n_s for all SB algorithms.

n_s	$U^{\text{SB}}_{\text{LPT}}$	$U^{\text{SB}}_{\text{SPT}}$	$U^{\text{SB}}_{\text{SS}}$	$U^{\text{SB}}_{\text{MSS}}$	$\boldsymbol{U^{\text{SB}}_{\text{MSK}}}$
3	0.00	1.54	0.00	0.00	**0.00**
5	0.26	3.76	0.00	0.00	**0.00**
10	0.73	10.29	0.03	0.00	**0.00**
20	1.87	14.43	0.20	0.01	**0.00**
50	3.76	12.62	2.27	2.19	**0.00**
100	4.40	9.71	3.57	3.55	**0.00**

Notes. The best results are given in bold.

TABLE 5. Behavior of GAP according to n_{pr} for all SB algorithms.

n_{pr}	$U^{\text{SB}}_{\text{LPT}}$	$U^{\text{SB}}_{\text{SPT}}$	$U^{\text{SB}}_{\text{SS}}$	$U^{\text{SB}}_{\text{MSS}}$	$\boldsymbol{U^{\text{SB}}_{\text{MSK}}}$
2	1.75	6.91	1.06	1.05	**0.00**
4	1.89	9.08	1.01	0.96	**0.00**
6	1.84	9.42	0.99	0.92	**0.00**
8	1.86	9.50	0.99	0.91	**0.00**

Notes. The best results are given in bold.

– *Perc* is the percentage of instances when $A^* = A$ over a specific number of instances.
– *Time* is the average running time for a fixed number of instances.

Our experimental study is based on the comparison between the developed algorithms for each category. After that we consider the best heuristic in each category and compare between them. Finally we present the comparison between all algorithms.

7.3. Separated-blocks category comparison

In this category of algorithms, five heuristics were developed as was described in Section 5.2. The results of the separated blocks category algorithms are presented in Table 3. From this table we can observe that $U^{\text{SB}}_{\text{MSK}}$ is the best heuristic in the separated-blocks category in 98.4% of the cases. However, the average running time of this heuristic is very high around 749.628 s. The heuristic that has the minimum percentage is $U^{\text{SB}}_{\text{SPT}}$. Whereas $U^{\text{SB}}_{\text{MSS}}$ heuristic has the second best case value of 69.9%; if we consider the running time which is only 1.687 s or (0.0022) of the time consumed by $U^{\text{SB}}_{\text{MSK}}$ heuristic.

In Table 4, the behavior of GAP according to SB heuristics is presented, as it can be seen from the shown results, $U^{\text{SB}}_{\text{MSK}}$ heuristic has the best results with (0.00) GAP for all n_s values, while the $U^{\text{SB}}_{\text{SPT}}$ has the worst performance at $n_s = 20$. The GAP value less than 0.01 was obtained by heuristics $U^{\text{SB}}_{\text{LPT}}$ at $n_s = 3$, $U^{\text{SB}}_{\text{SS}}$ at $n_s = \{3, 5\}$, $U^{\text{SB}}_{\text{MSS}}$ at $n_s = \{3, 5, 10\}$ and $U^{\text{SB}}_{\text{MSK}}$ for all n_s values.

The behavior of GAP according to the number of processers for all SB algorithms is given in Table 5, where $U^{\text{SB}}_{\text{MSK}}$ heuristic has the best results for all n_{pr} values, while the $U^{\text{SB}}_{\text{SPT}}$ has the worst performance at $n_{pr} = 6$. The GAP value less than 0.01 was obtained only by $U^{\text{SB}}_{\text{MSK}}$ heuristic for all n_{pr} values.

Table 6. Details for SB category algorithms.

n_s	n_{pr}	$U_{\text{LPT}}^{\text{SB}}$	$U_{\text{SPT}}^{\text{SB}}$	$U_{\text{SS}}^{\text{SB}}$	$U_{\text{MSS}}^{\text{SB}}$	$\boldsymbol{U_{\text{MSK}}^{\text{SB}}}$
3	2	0.00	6.16	0.00	0.00	**0.00**
	4	0.00	0.00	0.00	0.00	**0.00**
	6	0.00	0.00	0.00	0.00	**0.00**
	8	0.00	0.00	0.00	0.00	**0.00**
5	2	9.34	1.04	0.00	0.00	**0.00**
	4	0.00	5.72	0.00	0.00	**0.00**
	6	0.00	0.00	0.00	0.00	**0.00**
	8	0.00	0.00	0.00	0.00	**0.00**
10	2	1.48	8.94	0.00	0.00	**0.00**
	4	1.41	15.00	0.12	0.00	**0.00**
	6	0.01	11.37	0.00	0.00	**0.00**
	8	0.00	5.85	0.00	0.00	**0.00**
20	2	1.03	6.12	0.03	0.02	**0.01**
	4	2.26	14.71	0.15	0.01	**0.00**
	6	2.47	18.66	0.34	0.00	**0.00**
	8	1.73	18.24	0.29	0.00	**0.00**
50	2	2.92	5.48	2.49	2.48	**0.00**
	4	3.45	10.72	2.22	2.18	**0.00**
	6	4.06	15.30	2.15	2.06	**0.00**
	8	4.60	19.00	2.23	2.04	**0.00**
100	2	4.03	5.44	3.82	3.82	**0.00**
	4	4.21	8.34	3.58	3.57	**0.00**
	6	4.50	11.18	3.46	3.43	**0.00**
	8	4.86	13.90	3.44	3.39	**0.00**

Notes. The best results are given in bold.

Table 7. Results for the reverse-interference category algorithms.

	$U_{\text{LPT}}^{\text{RI}}$	$U_{\text{SPT}}^{\text{RI}}$	$\boldsymbol{U_{\text{SS}}^{\text{RI}}}$	$U_{\text{MSS}}^{\text{RI}}$	$U_{\text{MSK}}^{\text{RI}}$
Perc	32.9%	31.0%	**84.2%**	34.5%	34.5%
Time	0.008	0.008	**0.034**	2.876	740.635

Notes. The best results are given in bold.

The results of all SB category algorithms are presented in Table 6, for comparison purposes and for more details. The worst GAP values were obtained by heuristics $U_{\text{SPT}}^{\text{SB}}$ at $n_s = 50$ and $n_{pr} = 8$, $U_{\text{LPT}}^{\text{SB}}$ at $n_s = 5$ and $n_{pr} = 2$, $U_{\text{SS}}^{\text{SB}}$ at $n_s = 100$ and $n_{pr} = 2$ and $U_{\text{MSS}}^{\text{SB}}$ at $n_s = 100$ and $n_{pr} = 2$. While all GAP values for $U_{\text{MSK}}^{\text{SB}}$ heuristic were less than 0.01 for all n_s and n_{pr} values except at $n_s = 20$ and $n_{pr} = 2$ the GAP value was 0.01.

7.4. Reverse-interference category comparison

For this category of algorithms, five heuristics were developed as described in Section 5.3.1. The results of the reverse-interference category algorithms are shown in Table 7. In this table it can be noticed that $U_{\text{SS}}^{\text{RI}}$ is the best heuristic in the reverse-interference category with 84.2% cases. However, the average running time of this heuristic is acceptable around 0.034 s. The heuristic that has the minimum percentage is $U_{\text{SPT}}^{\text{RI}}$. It is worthy to note that, MSK is the best heuristic for the parallel machines, however for the RI algorithms the utilization of SS becomes more efficient and produces better results for the studied problem.

In Table 8, the behavior of GAP according to n_s is presented for all RI category algorithms, the given results show that the $U_{\text{SS}}^{\text{RI}}$ heuristic has the best performance based on GAP values for all n_s values, while $U_{\text{SPT}}^{\text{RI}}$ heuristic

TABLE 8. Behavior of GAP according to n_s for all RI algorithms.

n_s	$U_{\text{LPT}}^{\text{RI}}$	$U_{\text{SPT}}^{\text{RI}}$	$\boldsymbol{U_{\text{SS}}^{\text{RI}}}$	$U_{\text{MSS}}^{\text{RI}}$	$U_{\text{MSK}}^{\text{RI}}$
3	0.26	0.32	**0.26**	0.26	0.26
5	0.88	1.52	**0.21**	0.84	0.84
10	3.44	5.18	**0.10**	3.05	3.05
20	3.39	4.37	**0.02**	2.93	2.92
50	0.98	1.40	**0.01**	0.88	0.88
100	0.37	0.57	**0.00**	0.34	0.34

Notes. The best results are given in bold.

TABLE 9. Details for RI category algorithms.

n_{pr}	$U_{\text{LPT}}^{\text{RI}}$	$U_{\text{SPT}}^{\text{RI}}$	$\boldsymbol{U_{\text{SS}}^{\text{RI}}}$	$U_{\text{MSS}}^{\text{RI}}$	$U_{\text{MSK}}^{\text{RI}}$
2	0.35	0.46	**0.35**	0.35	0.35
4	1.55	2.27	**0.04**	1.37	1.36
6	2.01	2.79	**0.01**	1.79	1.78
8	2.31	3.38	**0.00**	2.03	2.03

Notes. The best results are given in bold.

has the worst performance for all n_s values. $U_{\text{SPT}}^{\text{RI}}$ heuristic has the worst performance value at $n_s = 10$. The GAP values less than 0.01 was obtained by heuristic $U_{\text{SS}}^{\text{RI}}$ at $n_s = 100$. The best GAP values of 0.26, 0.32, 0.00, 0.26 and 0.26 were obtained by heuristics $U_{\text{LPT}}^{\text{RI}}$, $U_{\text{SPT}}^{\text{RI}}$, $U_{\text{SS}}^{\text{RI}}$, $U_{\text{MSS}}^{\text{RI}}$ and $U_{\text{MSK}}^{\text{RI}}$ at $n_s = 3$ except for $U_{\text{SS}}^{\text{RI}}$ at $n_s = 100$.

The behavior of GAP according to the number of processors for all RI algorithms is given in Table 9. In this table $U_{\text{SS}}^{\text{RI}}$ heuristic has the best performance, while the $U_{\text{SPT}}^{\text{RI}}$ heuristic has the worst performance. Indeed, for $U_{\text{SPT}}^{\text{RI}}$ the best GAP value is 0.46 at $n_{pr} = 2$, the worst GAP value of 3.38 at $n_{pr} = 8$. For $U_{\text{SS}}^{\text{RI}}$ the best GAP value is 0.00 at $n_{pr} = 8$, the worst GAP value of 0.35 at $n_{pr} = 2$.

For more details of the RI category algorithms, Table 10 shows all the obtained results. The worst GAP values were obtained by heuristics $U_{\text{LPT}}^{\text{RI}}$ of 7.34 at $n_s = 20$ and $n_{pr} = 8$, $U_{\text{SPT}}^{\text{RI}}$ of 9.37 at $n_s = 20$ and $n_{pr} = 8$, $U_{\text{SS}}^{\text{RI}}$ of 1.03 at $n_s = 3$ and $n_{pr} = 2$ and $U_{\text{MSS}}^{\text{RI}}$ of 6.31 at $n_s = 20$ and $n_{pr} = 8$ and $U_{\text{MSK}}^{\text{RI}}$ of 6.31 at $n_s = 20$ and $n_{pr} = 8$. Based on GAP values, the best performing heuristic was $U_{\text{SS}}^{\text{RI}}$ and its best GAP values were obtained at $n_s = 3$ and $n_{pr} = \{4, 6, 8\}$, $n_s = 5$ and $n_{pr} = \{6, 8\}$, $n_s = 20$ and $n_{pr} = \{6, 8\}$, $n_s = 50$ and $n_{pr} = \{4, 6, 8\}$, $n_s = 100$ and $n_{pr} = \{4, 6, 8\}$.

7.5. Fictitious-processes category comparison

In this category of algorithms, four heuristics were developed as described in the Section 5.3.2. Table 11 shows the results for the fictitious-processes category algorithms. From this table it can be observed that $U_{\text{LPT}}^{\text{FP}}$ has the best heuristic in the fictitious-processes category with 76.2% of cases, with a good average running time around 0.005 s. The heuristic that has the minimum percentage is $U_{\text{SPT}}^{\text{FP}}$.

In Table 12, we present the behavior of GAP according to n_s for all FP category algorithms, based on the shown results it is noticed that the performance of the given algorithms increases as the value of n_s increases. The heuristic $U_{\text{LPT}}^{\text{FP}}$ has the best performance results for all n_s values, while the $U_{\text{SPT}}^{\text{FP}}$ has the worst performance. The GAP value less than 0.01 was obtained by heuristics $U_{\text{LPT}}^{\text{FP}}$ at $n_s = \{50, 100\}$, $U_{\text{RL}}^{\text{FP}}$ at $n_s = \{50, 100\}$ and $U_{\text{IRL}}^{\text{FP}}$ at $n_s = \{50, 100\}$. It is worthy ti notice that the heuristics $U_{\text{LPT}}^{\text{FP}}$, $U_{\text{RL}}^{\text{FP}}$ and $U_{\text{IRL}}^{\text{FP}}$ obtained their best GAP values at $n_s = \{50, 100\}$. While their worst GAP values at $n_s = 3$.

TABLE 10. Details for RI category algorithms.

n_s	n_{pr}	$U^{\text{RI}}_{\text{LPT}}$	$U^{\text{RI}}_{\text{SPT}}$	$\boldsymbol{U^{\text{RI}}_{\text{SS}}}$	$U^{\text{RI}}_{\text{MSS}}$	$U^{\text{RI}}_{\text{MSK}}$
3	2	1.03	1.27	**1.03**	1.03	1.03
	4	0.00	0.00	**0.00**	0.00	0.00
	6	0.00	0.00	**0.00**	0.00	0.00
	8	0.00	0.00	**0.00**	0.00	0.00
5	2	0.69	0.89	**0.69**	0.72	0.72
	4	2.85	5.19	**0.15**	2.62	2.62
	6	0.00	0.00	**0.00**	0.00	0.00
	8	0.00	0.00	**0.00**	0.00	0.00
10	2	0.26	0.36	**0.26**	0.28	0.28
	4	4.03	5.24	**0.06**	3.39	3.39
	6	5.92	8.52	**0.06**	5.37	5.36
	8	3.55	6.60	**0.01**	3.18	3.18
20	2	0.08	0.15	**0.08**	0.07	0.07
	4	1.69	2.18	**0.01**	1.51	1.50
	6	4.44	5.76	**0.00**	3.83	3.82
	8	7.34	9.37	**0.00**	6.31	6.30
50	2	0.02	0.04	**0.02**	0.01	0.01
	4	0.51	0.72	**0.00**	0.46	0.46
	6	1.21	1.75	**0.00**	1.09	1.09
	8	2.17	3.08	**0.00**	1.96	1.96
100	2	0.01	0.01	**0.01**	0.00	0.00
	4	0.22	0.31	**0.00**	0.20	0.20
	6	0.46	0.71	**0.00**	0.43	0.43
	8	0.80	1.24	**0.00**	0.74	0.73

Notes. The best results are given in bold.

TABLE 11. Results for the fictitious-processes category algorithm.

	$\boldsymbol{U^{\text{FP}}_{\text{LPT}}}$	$U^{\text{FP}}_{\text{SPT}}$	$U^{\text{FP}}_{\text{RL}}$	$U^{\text{FP}}_{\text{IRL}}$
Perc	**76.2%**	14.6%	71.6%	71.3%
Time	**0.005**	0.005	0.015	4.634

Notes. The best results are given in bold.

TABLE 12. Behavior of GAP according to n_s for all FP category algorithms.

n_s	$\boldsymbol{U^{\text{FP}}_{\text{LPT}}}$	$U^{\text{FP}}_{\text{SPT}}$	$U^{\text{FP}}_{\text{RL}}$	$U^{\text{FP}}_{\text{IRL}}$
3	**0.54**	2.30	0.63	0.72
5	**0.28**	1.77	0.34	0.36
10	**0.07**	0.92	0.10	0.09
20	**0.01**	0.41	0.02	0.02
50	**0.00**	0.13	0.00	0.00
100	**0.00**	0.05	0.00	0.00

Notes. The best results are given in bold.

TABLE 13. Behavior of GAP according to n_{pr} for all FP category algorithms.

n_{pr}	$\boldsymbol{U^{\mathrm{FP}}_{\mathrm{LPT}}}$	$U^{\mathrm{FP}}_{\mathrm{SPT}}$	$U^{\mathrm{FP}}_{\mathrm{RL}}$	$U^{\mathrm{FP}}_{\mathrm{IRL}}$
2	**0.13**	0.76	0.18	0.15
4	**0.18**	1.01	0.21	0.24
6	**0.17**	1.00	0.19	0.24
8	**0.12**	0.96	0.14	0.16

Notes. The best results are given in bold.

TABLE 14. Details for FP category algorithms.

n_s	n_{pr}	$\boldsymbol{U^{\mathrm{FP}}_{\mathrm{LPT}}}$	$U^{\mathrm{FP}}_{\mathrm{SPT}}$	$U^{\mathrm{FP}}_{\mathrm{RL}}$	$U^{\mathrm{FP}}_{\mathrm{IRL}}$
3	2	**0.57**	2.25	0.72	0.62
	4	**0.63**	2.59	0.80	0.90
	6	**0.57**	2.30	0.62	0.90
	8	**0.37**	2.06	0.40	0.45
5	2	**0.18**	1.41	0.26	0.22
	4	**0.35**	1.91	0.37	0.44
	6	**0.35**	1.93	0.40	0.46
	8	**0.22**	1.82	0.31	0.33
10	2	**0.05**	0.56	0.06	0.06
	4	**0.06**	0.95	0.09	0.08
	6	**0.07**	1.07	0.11	0.08
	8	**0.10**	1.09	0.12	0.14
20	2	**0.01**	0.25	0.01	0.02
	4	**0.01**	0.40	0.02	0.01
	6	**0.01**	0.47	0.02	0.02
	8	**0.02**	0.51	0.03	0.03
50	2	**0.00**	0.07	0.00	0.00
	4	**0.00**	0.13	0.00	0.00
	6	**0.00**	0.16	0.00	0.00
	8	**0.00**	0.18	0.00	0.00
100	2	**0.00**	0.02	0.00	0.00
	4	**0.00**	0.05	0.00	0.00
	6	**0.00**	0.06	0.00	0.00
	8	**0.00**	0.07	0.00	0.00

Notes. The best results are given in bold.

The behavior of GAP according to the number of processers for all FP algorithms is given in Table 13. Based on the given results, increasing the number of processors does not necessarily mean an increase in the performance. For example, the GAP value for $U^{\mathrm{FP}}_{\mathrm{SPT}}$ heuristic was 0.76 at $n_{pr} = 2$ and increases up to 0.96 when $n_{pr} = 8$. While for $U^{\mathrm{FP}}_{\mathrm{LPT}}$ heuristic, the case was different as it can be seen from Table 13, the GAP values mostly decrease while n_{pr} increases.

For more details of the FP category algorithms, Table 14 is presented.

The worst GAP values were obtained by heuristics $U^{\mathrm{FP}}_{\mathrm{LPT}}$ of 0.63 at $n_s = 3$ and $n_{pr} = 4$, $U^{\mathrm{FP}}_{\mathrm{SPT}}$ of 2.59 at $n_s = 3$ and $n_{pr} = 4$, $U^{\mathrm{FP}}_{\mathrm{RL}}$ of 0.80 at $n_s = 3$ and $n_{pr} = 4$ and $U^{\mathrm{FP}}_{\mathrm{IRL}}$ of 0.90 at $n_s = 3$ and $n_{pr} = \{4, 6\}$.

For all heuristics except for $U^{\mathrm{FP}}_{\mathrm{SPT}}$, the best performance results were obtained at $n_s = 50$ and $n_{pr} = \{2, 4, 6, 8\}$, $n_s = 100$ and $n_{pr} = \{2, 4, 6, 8\}$.

Table 15. Results for the fictitious-processes with multi-subset category algorithms.

	$\boldsymbol{U_{\mathrm{LPT}}^{\mathrm{FPM}}}$	$U_{\mathrm{SPT}}^{\mathrm{FPM}}$	$U_{\mathrm{RL}}^{\mathrm{FPM}}$	$U_{\mathrm{IRL}}^{\mathrm{FPM}}$
Perc	**76.0%**	19.0%	70.7%	70.9%
Time	**1.294**	1.538	1.558	1.956

Notes. The best results are given in bold.

Table 16. Behavior of GAP according to n_s for all FPM category algorithms.

n_s	$\boldsymbol{U_{\mathrm{LPT}}^{\mathrm{FPM}}}$	$U_{\mathrm{SPT}}^{\mathrm{FPM}}$	$U_{\mathrm{RL}}^{\mathrm{FPM}}$	$U_{\mathrm{IRL}}^{\mathrm{FPM}}$
3	**0.53**	2.21	0.68	0.69
5	**0.28**	1.55	0.36	0.32
10	**0.08**	0.66	0.10	0.09
20	**0.02**	0.23	0.03	0.03
50	**0.00**	0.05	0.00	0.00
100	**0.00**	0.02	0.00	0.00

Notes. The best results are given in bold.

Table 17. Behavior of GAP according to n_{pr} for all FPM category algorithms.

n_{pr}	$\boldsymbol{U_{\mathrm{LPT}}^{\mathrm{FPM}}}$	$U_{\mathrm{SPT}}^{\mathrm{FPM}}$	$U_{\mathrm{RL}}^{\mathrm{FPM}}$	$U_{\mathrm{IRL}}^{\mathrm{FPM}}$
2	**0.12**	0.53	0.15	0.15
4	**0.19**	0.83	0.25	0.22
6	**0.17**	0.91	0.23	0.22
8	**0.13**	0.87	0.15	0.17

Notes. The best results are given in bold.

7.6. Fictitious-processes with multi-subset category comparison

In this category of algorithms, four heuristics were developed as described in the Section 5.3.3. Table 15 shows the results of the fictitious-processes with multi-subset category algorithms. From this table, it can be observed that $U_{\mathrm{LPT}}^{\mathrm{FPM}}$ is the best heuristic in the fictitious-processes with multi-subset category in 76% of the cases. However, the average running time of this heuristic is acceptable around 1.294 s. The heuristic that has the minimum percentage is $U_{\mathrm{SPT}}^{\mathrm{FPM}}$.

In Table 16, we present the behavior of GAP according to n_s for all FPM category algorithms. Based on the shown results it is noticed that the performance of the given algorithms increases as the value of n_s increases for all heuristics. Also, it is noticed that $U_{\mathrm{LPT}}^{\mathrm{FP}}$, $U_{\mathrm{RL}}^{\mathrm{FP}}$ and $U_{\mathrm{IRL}}^{\mathrm{FP}}$ heuristics were more sensitive to n_s values than $U_{\mathrm{SPT}}^{\mathrm{FP}}$ heuristics.

The behavior of GAP according to the number of processers for all FPM algorithms is given in Table 17. Based on the given results, the best obtained results for all heuristics were obtained at $n_{pr} = 2$, increasing the number of processors does not necessarily mean an increase in the performance.

For more details of the FPM category algorithms, Table 18 is presented. The worst GAP values were obtained by heuristics $U_{\mathrm{LPT}}^{\mathrm{FPM}}$ of 0.67 at $n_s = 3$ and $n_{pr} = 4$, $U_{\mathrm{SPT}}^{\mathrm{FPM}}$ of 2.63 at $n_s = 3$ and $n_{pr} = 4$, $U_{\mathrm{RL}}^{\mathrm{FPM}}$ of 0.90 at $n_s = 3$ and $n_{pr} = 4$ and $U_{\mathrm{IRL}}^{\mathrm{FPM}}$ of 0.85 at $n_s = 3$ and $n_{pr} = 4$. For all heuristics except for $U_{\mathrm{SPT}}^{\mathrm{FP}}$, the GAP value of 0.00 was obtained at $n_s = 50$ and $n_{pr} = \{2, 4, 6, 8\}$, $n_s = 100$ and $n_{pr} = \{2, 4, 6, 8\}$.

Table 18. Details for the FPM category algorithms.

n_s	n_{pr}	$\boldsymbol{U_{\mathbf{LPT}}^{\mathbf{FPM}}}$	$U_{\text{SPT}}^{\text{FPM}}$	$U_{\text{RL}}^{\text{FPM}}$	$U_{\text{IRL}}^{\text{FPM}}$
3	2	**0.51**	1.84	0.59	0.63
	4	**0.67**	2.63	0.90	0.85
	6	**0.55**	2.28	0.75	0.73
	8	**0.40**	2.08	0.49	0.54
5	2	**0.15**	0.90	0.22	0.18
	4	**0.36**	1.51	0.47	0.35
	6	**0.37**	1.94	0.49	0.42
	8	**0.24**	1.85	0.25	0.32
10	2	**0.04**	0.29	0.06	0.06
	4	**0.09**	0.59	0.09	0.09
	6	**0.10**	0.86	0.14	0.12
	8	**0.09**	0.89	0.13	0.10
20	2	**0.01**	0.09	0.01	0.01
	4	**0.02**	0.20	0.02	0.03
	6	**0.02**	0.29	0.03	0.03
	8	**0.04**	0.33	0.05	0.05
50	2	**0.00**	0.02	0.00	0.00
	4	**0.00**	0.04	0.00	0.00
	6	**0.00**	0.06	0.00	0.00
	8	**0.00**	0.08	0.00	0.00
100	2	**0.00**	0.01	0.00	0.00
	4	**0.00**	0.02	0.00	0.00
	6	**0.00**	0.02	0.00	0.00
	8	**0.00**	0.02	0.00	0.00

Notes. The best results are given in bold.

Table 19. Results for the interference processes-blocks category algorithms.

	$\boldsymbol{U_{\mathbf{LPT}}^{\mathbf{IPB}}}$	$U_{\text{SPT}}^{\text{IPB}}$	$U_{\text{RL}}^{\text{IPB}}$	$U_{\text{IRL}}^{\text{IPB}}$
Perc	**79.2%**	21.1%	76.6%	76.4%
Time	**0.006**	0.005	0.014	4.250

Notes. The best results are given in bold.

7.7. Interference processes-blocks category comparison

In this category of algorithms, five heuristics were developed as described in the Section 5.3.4. Table 19 shows the results for the interference processes-blocks category algorithms. From this table it can be observed that $U_{\text{LPT}}^{\text{IPB}}$ is the best heuristic in the interference processes-blocks category in 79.2% of the cases. However, the average running time of this heuristic is good around 0.006 s. The heuristic that has the minimum percentage is $U_{\text{SPT}}^{\text{IPB}}$.

In Table 20, we present the behavior of GAP according to n_s for all IPB category algorithms. As it can be noticed from the given results, increasing n_s values improves the values of all heuristics and that the $U_{\text{SPT}}^{\text{IPB}}$ heuristic was the least sensitive to the n_s increase for the given problem.

The behavior of GAP according to the number of processers for all IPB algorithms is given in Table 21. The given results showed that increasing the number of processors does not necessarily mean an increase in

Table 20. Behavior of GAP according to n_s for all IPB category algorithms.

n_s	$\boldsymbol{U_{\mathbf{LPT}}^{\mathbf{IPB}}}$	$U_{\mathrm{SPT}}^{\mathrm{IPB}}$	$U_{\mathrm{RL}}^{\mathrm{IPB}}$	$U_{\mathrm{IRL}}^{\mathrm{IPB}}$
3	**0.49**	2.02	0.61	0.60
5	**0.27**	1.44	0.35	0.34
10	**0.07**	0.55	0.07	0.08
20	**0.02**	0.17	0.02	0.02
50	**0.00**	0.03	0.00	0.00
100	**0.00**	0.01	0.00	0.00

Notes. The best results are given in bold.

Table 21. Behavior of GAP according to n_{pr} for all IPB category algorithms.

n_{pr}	$\boldsymbol{U_{\mathbf{LPT}}^{\mathbf{IPB}}}$	$U_{\mathrm{SPT}}^{\mathrm{IPB}}$	$U_{\mathrm{RL}}^{\mathrm{IPB}}$	$U_{\mathrm{IRL}}^{\mathrm{IPB}}$
2	**0.12**	0.37	0.11	0.15
4	**0.17**	0.75	0.23	0.20
6	**0.16**	0.86	0.20	0.18
8	**0.12**	0.84	0.16	0.17

Notes. The best results are given in bold.

Table 22. Details for the IPB category algorithms.

n_s	n_{pr}	$\boldsymbol{U_{\mathbf{LPT}}^{\mathbf{IPB}}}$	$U_{\mathrm{SPT}}^{\mathrm{IPB}}$	$U_{\mathrm{RL}}^{\mathrm{IPB}}$	$U_{\mathrm{IRL}}^{\mathrm{IPB}}$
3	2	**0.48**	1.24	0.42	0.56
	4	**0.59**	2.55	0.84	0.68
	6	**0.55**	2.29	0.70	0.53
	8	**0.34**	2.02	0.50	0.61
5	2	**0.18**	0.70	0.19	0.25
	4	**0.33**	1.28	0.46	0.41
	6	**0.35**	1.93	0.43	0.46
	8	**0.23**	1.84	0.32	0.25
10	2	**0.05**	0.20	0.05	0.05
	4	**0.07**	0.49	0.07	0.08
	6	**0.06**	0.69	0.07	0.09
	8	**0.09**	0.83	0.10	0.13
20	2	**0.01**	0.05	0.01	0.01
	4	**0.01**	0.15	0.01	0.01
	6	**0.02**	0.22	0.02	0.03
	8	**0.03**	0.27	0.03	0.03
50	2	**0.00**	0.01	0.00	0.00
	4	**0.00**	0.03	0.00	0.00
	6	**0.00**	0.04	0.00	0.00
	8	**0.00**	0.06	0.00	0.00
100	2	**0.00**	0.00	0.00	0.00
	4	**0.00**	0.01	0.00	0.00
	6	**0.00**	0.01	0.00	0.00
	8	**0.00**	0.01	0.00	0.00

Notes. The best results are given in bold.

TABLE 23. Best categories comparison.

	$U^{\text{SB}}_{\text{MSK}}$	$U^{\text{RI}}_{\text{SS}}$	$U^{\text{FP}}_{\text{LPT}}$	$\boldsymbol{U^{\text{FPM}}_{\text{LPT}}}$	$U^{\text{IPB}}_{\text{LPT}}$
Perc	10.9%	31.6%	47.6%	**48.7%**	47.9%
Time	749.628	0.034	0.005	**1.294**	0.006

Notes. The best results are given in bold.

TABLE 24. Details for best categories comparison.

n_s	n_{pr}	$U^{\text{SB}}_{\text{MSK}}$	$U^{\text{RI}}_{\text{SS}}$	$U^{\text{FP}}_{\text{LPT}}$	$\boldsymbol{U^{\text{FPM}}_{\text{LPT}}}$	$U^{\text{IPB}}_{\text{LPT}}$
3	2	32.22	3.15	0.41	**0.41**	0.49
	4	126.96	22.79	0.07	**0.07**	0.07
	6	224.59	75.64	0.00	**0.00**	0.00
	8	312.89	121.17	0.00	**0.00**	0.00
5	2	16.29	1.79	0.23	**0.21**	0.25
	4	75.83	9.30	0.31	**0.29**	0.39
	6	152.88	32.53	0.00	**0.00**	0.00
	8	227.66	71.76	0.00	**0.00**	0.00
10	2	5.33	0.56	0.06	**0.07**	0.05
	4	29.18	0.88	0.83	**0.83**	0.79
	6	65.63	5.37	0.26	**0.25**	0.26
	8	109.01	17.45	0.07	**0.07**	0.10
20	2	1.73	0.18	0.01	**0.01**	0.01
	4	9.42	0.13	0.65	**0.65**	0.64
	6	22.15	0.23	1.18	**1.18**	1.18
	8	39.09	0.82	0.85	**0.84**	0.83
50	2	0.26	2.30	2.26	**2.26**	2.26
	4	1.32	1.22	1.50	**1.50**	1.50
	6	3.06	0.21	0.79	**0.79**	0.78
	8	6.45	0.03	0.92	**0.92**	0.92
100	2	0.10	3.73	3.72	**3.72**	3.72
	4	0.52	3.17	3.33	**3.33**	3.33
	6	1.05	2.66	2.96	**2.96**	2.95
	8	1.73	2.14	2.60	**2.60**	2.60

Notes. The best results are given in bold.

the performance for all the given heuristics. For the given problem in this research, increasing the number of processors is not significant and the best GAP results were obtained at $n_{pr} = 2$ for all heuristics.

For more details of the IPB category algorithms, Table 22 is presented.

The worst GAP values were obtained by heuristics $U^{\text{IPB}}_{\text{LPT}}$ of 0.59 at $n_s = 3$ and $n_{pr} = 4$, $U^{\text{IPB}}_{\text{SPT}}$ of 2.55 at $n_s = 3$ and $n_{pr} = 4$, $U^{\text{IPB}}_{\text{RL}}$ of 0.84 at $n_s = 3$ and $n_{pr} = 4$ and $U^{\text{IPB}}_{\text{IRL}}$ of 0.68 at $n_s = 3$ and $n_{pr} = 4$.

For all heuristics except for $U^{\text{IPB}}_{\text{SPT}}$, the GAP value of 0.00 was obtained at $n_s = 50$ and $n_{pr} = \{2, 4, 6, 8\}$, $n_s = 100$ and $n_{pr} = \{2, 4, 6, 8\}$. While the heuristic $U^{\text{IPB}}_{\text{SPT}}$ obtained its best GAP value of 0.00 at $n_s = 100$ and $n_{pr} = 2$.

7.8. Comparison between best category algorithms

The best algorithms were $U^{\text{SB}}_{\text{MSK}}$, $U^{\text{RI}}_{\text{SS}}$, $U^{\text{FP}}_{\text{LPT}}$, $U^{\text{FPM}}_{\text{LPT}}$ and $U^{\text{IPB}}_{\text{LPT}}$ for the categories SB, RI, FP, FPM, and IPB respectively.

TABLE 25. Overall comparison between all heuristics.

	$U^{\mathrm{SB}}_{\mathrm{SPT}}$	$U^{\mathrm{SB}}_{\mathrm{LPT}}$	$U^{\mathrm{SB}}_{\mathrm{SS}}$	$U^{\mathrm{SB}}_{\mathrm{MSS}}$	$U^{\mathrm{SB}}_{\mathrm{MSK}}$	$U^{\mathrm{RI}}_{\mathrm{LPT}}$	$U^{\mathrm{RI}}_{\mathrm{SPT}}$	$U^{\mathrm{RI}}_{\mathrm{SS}}$	$U^{\mathrm{RI}}_{\mathrm{MSS}}$	$U^{\mathrm{RI}}_{\mathrm{MSK}}$	$U^{\mathrm{FP}}_{\mathrm{LPT}}$
Perc	0.0%	0.0%	0.0%	0.0%	10.9%	2.3%	1.7%	30.7%	2.2%	2.2%	33.1%
Time	0.005	0.005	0.023	1.687	749.628	0.008	0.008	0.034	2.876	740.635	0.005
	$U^{\mathrm{FP}}_{\mathrm{SPT}}$	$U^{\mathrm{FP}}_{\mathrm{RL}}$	$U^{\mathrm{FP}}_{\mathrm{IRL}}$	$\mathbf{U^{FPM}_{LPT}}$	$U^{\mathrm{FPM}}_{\mathrm{SPT}}$	$U^{\mathrm{FPM}}_{\mathrm{RL}}$	$U^{\mathrm{FPM}}_{\mathrm{IRL}}$	$U^{\mathrm{IPB}}_{\mathrm{LPT}}$	$U^{\mathrm{IPB}}_{\mathrm{SPT}}$	$U^{\mathrm{IPB}}_{\mathrm{RL}}$	$U^{\mathrm{IPB}}_{\mathrm{IRL}}$
Perc	10.8%	31.1%	30.4%	**34.2%**	11.4%	32.2%	32.3%	33.7%	11.9%	32.5%	32.3%
Time	0.005	0.015	4.634	**1.294**	1.538	1.558	1.956	0.006	0.005	0.014	4.250

Notes. The best results are given in bold.

In this subsection, we compare these heuristics to find the best heuristic between all algorithms developed in this paper.

Table 23 presents the comparison results of the best category's heuristics. The given results indicates that the best heuristic is $U^{\mathrm{FPM}}_{\mathrm{LPT}}$ with 48.7% and an average running time of 1.294 s. While the worst heuristic was $U^{\mathrm{SB}}_{\mathrm{MSK}}$ with 10.9% and an average running time of 749.628 s.

Table 24 represents the details of best category's comparison. Based on the shown results, the best performing heuristics were $U^{\mathrm{FPM}}_{\mathrm{LPT}}$ and $U^{\mathrm{FP}}_{\mathrm{LPT}}$. While the worst performing heuristic was $U^{\mathrm{SB}}_{\mathrm{MSK}}$. Heuristics that obtained GAP values less than 0.01 were, $U^{\mathrm{FP}}_{\mathrm{LPT}}$, $U^{\mathrm{IPB}}_{\mathrm{LPT}}$ at $n_s = 3$, $n_{pr} = \{6, 8\}$ and at $n_s = 5$, $n_{pr} = \{6, 8\}$.

The comparison results between all algorithms are given in Table 25. The shown results indicates that there aren't any dominance heuristic between the used algorithms, and that the best heuristic is $U^{\mathrm{FPM}}_{\mathrm{LPT}}$ in 34.2% of cases with an average time of 1.294 s. Four algorithms $U^{\mathrm{SB}}_{\mathrm{SPT}}$, $U^{\mathrm{SB}}_{\mathrm{LPT}}$, $U^{\mathrm{SB}}_{\mathrm{SS}}$, and $U^{\mathrm{SB}}_{\mathrm{MSS}}$ have a percentage less than 0.1%. The second best heuristic is $U^{\mathrm{FP}}_{\mathrm{LPT}}$ with percentage of 33.1% with an average time of 0.005 s.

8. CONCLUSION

This research utilizes operational research in railway systems to address identical parallel processors scheduling problem. Because of trip constraints and time schedule limitations, railway system must complete all the given requests in a limited time. This is an NP-Hard problem, and the main goal is to minimize the maximum completion time of the received requests. This problem was solved by developing new heuristics based on two categories of algorithms, the separated blocks category algorithm (SB) and the blocks interference category algorithm. The first category is composed of 5 heuristics. While the second category is composed of four algorithm types, which are, Reverse-interference based algorithm (RI), Fictitious-processes based algorithm (FP), Fictitious-processes with multi-subset based algorithm (FPM) and Interference Processes-blocks based algorithm (IPB). Each algorithm type is also composed of different heuristics, RI algorithm has 5 different heuristics, FP algorithm has 4 different heuristics, FPM algorithm has 4 different heuristics and IPB algorithm has 4 different heuristics. The total of the developed heuristics are 22. The performance assessments of the developed heuristics were performed for the execution time based on the average relative gap; the given results showed that there isn't any dominance between the algorithms and the best heuristic for the proposed problem was $U^{\mathrm{FPM}}_{\mathrm{LPT}}$ heuristics in 34.2% of the cases. The presented solution was capable of reading data from DC table before finishing the execution of all previous processes, which enables the control unit of the railway monitoring system, to derive the proper instructions for each request type, to ensure that there will be no latency or cancelation for any of the scheduled trips.

In this work, the maximum percentage is 34.2% which is relatively small, the next objective of this work will be to enhance the presented heuristics to achieve a percentage of no less than 50%. The expected enhancement will be reached by considering three aspects. The first issue will be to search for a metaheuristic that use all the given heuristics. The second consideration is to apply a variable neighborhood search (VNS)to enhance the given heuristics. Finally, derive a lower bound for the studied problem to be compared with the results obtained by the given heuristics.

Acknowledgements. The authors would like to thank the Deanship of Scientific Research at Majmaah University for supporting this work under Project Number No. R-2022-188.

References

[1] I. Agrebi, M. Jemmali, H. Alquhayz and T. Ladhari, Metaheuristic algorithms for the two-machine flowshop scheduling problem with release dates and blocking constraint. *J. Ch. Inst. Eng.* **44** (2021) 573–582.

[2] F. al Fayez, L.K.B. Melhim and M. Jemmali, Heuristics to optimize the reading of railway sensors data. In: 2019 6th International Conference on Control, Decision and Information Technologies (CoDIT). IEEE (2019) 1676–1681.

[3] M. Alharbi and M. Jemmali, Algorithms for investment project distribution on regions. *Comput. Intell. Neurosci.* **2020** (2020). DOI: 10.1155/2020/3607547.

[4] M. Alkhelaiwi, W. Boulila, J. Ahmad, A. Koubaa and M. Driss, An efficient approach based on privacy-preserving deep learning for satellite image classification. *Remote Sens.* **13** (2021) 2221.

[5] H. Alquhayz and M. Jemmali, Max–min processors scheduling. *Inf. Technol. Control* **50** (2021) 5–12.

[6] A. Altay and M. Baykal-Gürsoy, Imperfect rail-track inspection scheduling with zero-inflated miss rates. *Transp. Res. Part C Emerg. Technol.* **138** (2022) 103608.

[7] H. Amdouni, M. Jemmali, M. Mrad and T. Ladhari, An exact algorithm minimizing the makespan for the twomachine flowshop scheduling under release dates and blocking constraints. *Int. J. Ind. Eng.* **28** (2021) 631–643.

[8] A. Bakhtiary, J.A. Zakeri and S. Mohammadzadeh, An opportunistic preventive maintenance policy for tamping scheduling of railway tracks. *Int. J. Rail Transp.* **9** (2021) 1–22.

[9] W. Boulila, A top-down approach for semantic segmentation of big remote sensing images. *Earth Sci. Inf.* **12** (2019) 295–306.

[10] W. Boulila, M. Sellami, M. Driss, M. Al-Sarem, M. Safaei and F.A. Ghaleb, Rs-dcnn: a novel distributed convolutional-neural-networks based-approach for big remote-sensing image classification. *Comput. Electron. Agri.* **182** (2021) 106014.

[11] R.L. Burdett and E. Kozan, Performance profiling for predictive train schedules. *J. Rail Transp. Planning Manage.* **4** (2014) 98–114.

[12] C. Dao, R. Basten and A. Hartmann, Maintenance scheduling for railway tracks under limited possession time. *J. Transp. Eng. Part A Syst.* **144** (2018) 04018039.

[13] M. Dell'Amico, M. Iori, S. Martello and M. Monaci, Heuristic and exact algorithms for the identical parallel machine scheduling problem. *INFORMS J. Comput.* **20** (2008) 333–344.

[14] H. Dong, H. Zhu, Y. Li, Y. Lv, S. Gao, Q. Zhang and B. Ning, Parallel intelligent systems for integrated high-speed railway operation control and dynamic scheduling. *IEEE Trans. Cybern.* **48** (2018) 3381–3389.

[15] F. Donzella, M. del Cacho Estil-les, C. Bersani, R. Sacile and L. Zero, Train scheduling and rescheduling model based oncustomer satisfaction. Application to genoa railway network. In: 2018 13th Annual Conference on System of Systems Engineering (SoSE). IEEE (2018) 593–600.

[16] J. García, P. Moraga, M. Valenzuela, B. Crawford, R. Soto, H. Pinto, A. Peña, F. Altimiras and G. Astorga, A db-scan binarization algorithm applied to matrix covering problems. *Comput. Intell. Neurosci.* **2019** (2019). DOI: 10.1155/2019/3238574.

[17] H. Ghandorh, W. Boulila, S. Masood, A. Koubaa, F. Ahmed and J. Ahmad, Semantic segmentation and edge detection – approach to road detection in very high resolution satellite images. *Remote Sens.* **14** (2022) 613.

[18] M. Haouari and M. Jemmali, Tight bounds for the identical parallel machine-scheduling problem: Part II. *Int. Trans. Oper. Res.* **15** (2008) 19–34.

[19] M. Haouari, A. Gharbi and M. Jemmali, Tight bounds for the identical parallel machine scheduling problem. *Int. Trans. Oper. Res.* **13** (2006) 529–548.

[20] M. Jemmali, An optimal solution for the budgets assignment problem. *RAIRO: Oper. Res.* **55** (2021) 873–897.

[21] M. Jemmali, Projects distribution algorithms for regional development. *ADCAIJ* **10** (2021). http://hdl.handle.net/10366/147245.

[22] M. Jemmali, Intelligent algorithms and complex system for a smart parking for vaccine delivery center of covid-19. *Complex Intell. Syst.* **8** (2022) 597–609.

[23] M. Jemmali and A. Alourani, Mathematical model bounds for maximizing the minimum completion time problem. *J. Appl. Math. Comput. Mech.* **20** (2021) 43–50.

[24] M. Jemmali and H. Alquhayz, Equity data distribution algorithms on identical routers. In: International Conference on Innovative Computing and Communications. Springer (2020) 297–305.

[25] M. Jemmali, L.K.B. Melhim and M. Alharbi, Randomized-variants lower bounds for gas turbines aircraft engines. In: World Congress on Global Optimization. Springer (2019) 949–956.

[26] M. Jemmali, L.K.B. Melhim, S.O.B. Alharbi and A.S. Bajahzar, Lower bounds for gas turbines aircraft engines. *Commun. Math. App.* **10** (2019) 637–642.

[27] M. Jemmali, M.M. Otoom and F. al Fayez, Max–min probabilistic algorithms for parallel machines. In: Proceedings of the 2020 International Conference on Industrial Engineering and Industrial Management. ACM (2020) 19–24.

[28] M. Jemmali, L. Hidri and A. Alourani, Two-stage hybrid flowshop scheduling problem with independent setup times. *Int. J. Simul. Model. (IJSIMM)* **21** (2022) 5–16.

[29] M. Jemmali, L.K.B. Melhim, M.T. Alharbi, A. Bajahzar and M.N. Omri, Smart-parking management algorithms in smart city. *Sci. Rep.* **12** (2022) 1–15.

[30] B. Johannes, Scheduling parallel jobs to minimize the makespan. *J. Scheduling* **9** (2006) 433–452.
[31] S. Jütte, D. Müller and U.W. Thonemann, Optimizing railway crew schedules with fairness preferences. *J. Scheduling* **20** (2017) 43–55.
[32] D. Kovenkin and V. Podverbnyy, Issues of planning work on the current maintenance of the railway track. *Transp. Res. Proc.* **61** (2022) 636–640.
[33] D. Laha and J.N. Gupta, An improved cuckoo search algorithm for scheduling jobs on identical parallel machines. *Comput. Ind. Eng.* **126** (2018) 348–360.
[34] J. Lan, Y. Jiang, G. Fan, D. Yu and Q. Zhang, Real-time automatic obstacle detection method for traffic surveillance in urban traffic. *J. Signal Process. Syst.* **82** (2016) 357–371.
[35] S. Li and Y. Zhang, On-line scheduling on parallel machines to minimize the makespan. *J. Syst. Sci. Complexity* **29** (2016) 472–477.
[36] L.K.B. Melhim, M. Jemmali and M. Alharbi, Intelligent real-time intervention system applied in smart city. In: 2018 21st Saudi Computer Society National Computer Conference (NCC). IEEE (2018) 1–5.
[37] H. Mezni, M. Driss, W. Boulila, S.B. Atitallah, M. Sellami and N. Alharbi, Smartwater: a service-oriented and sensor cloud-based framework for smart monitoring of water environments. *Remote Sens.* **14** (2022) 922.
[38] M. Movaghar and S. Mohammadzadeh, Bayesian monte carlo approach for developing stochastic railway track degradation model using expert-based priors. *Struct. Infrastruct. Eng.* **18** (2022) 145–166.
[39] D. Pisinger, Dynamic programming on the word ram. *Algorithmica* **35** (2003) 128–145.
[40] J. Pradeep, M. Harikrishnan and K. Vijayakumar, Automatic railway detection and tracking inspecting system. In: Proceedings of International Conference on Data Science and Applications. Springer (2022) 309–318.
[41] Z. Qi, Y. Tian and Y. Shi, Efficient railway tracks detection and turnouts recognition method using hog features. *Neural Comput. App.* **23** (2013) 245–254.
[42] B. Roy and A.K. Sen, Meta-heuristic techniques to solve resource-constrained project scheduling problem. In: International Conference on Innovative Computing and Communications. Springer (2019) 93–99.
[43] M. Sedghi, O. Kauppila, B. Bergquist, E. Vanhatalo and M. Kulahci, A taxonomy of railway track maintenance planning and scheduling: a review and research trends. *Reliab. Eng. Syst. Saf.* **215** (2021) 107827.
[44] L.P. Veelenturf, D. Potthoff, D. Huisman and L.G. Kroon, Railway crew rescheduling with retiming. *Transp. Res. Part C Emerg. Technol.* **20** (2012) 95–110.
[45] X. Xu, K. Li, L. Yang and Z. Gao, An efficient train scheduling algorithm on a single-track railway system. *J. Scheduling* **22** (2019) 85–105.
[46] C. Zhang, Y. Gao, L. Yang, U. Kumar and Z. Gao, Integrated optimization of train scheduling and maintenance planning on high-speed railway corridors. *Omega* **87** (2019) 86–104.