Applied Soft Computing Journal 82 (2019) 105575

Contents lists available at ScienceDirect

Applied Soft Computing Journal

journal homepage: www.elsevier.com/locate/asoc

# Scheduling software updates for connected cars with limited availability

Carlos E. Andrade [a,*], Simon D. Byers [b], Vijay Gopalakrishnan [b], Emir Halepovic [b], David J. Poole [b], Lien K. Tran [b], Christopher T. Volinsky [b]

[a] AT&T Labs Research, 200 South Laurel Avenue, Middletown, NJ 07748, USA
[b] AT&T Labs Research, 1 AT&T Way, Bedminster, NJ 07921, USA

## HIGHLIGHTS

- We study a problem of scheduling Firmware-Over-The-Air updates for connected cars.
- We present a new problem called Time- and Machine-Dependent Scheduling Problem.
- We propose several metaheuristics and an integer programming model.
- Tests were performed on synthetic and real scenarios with massive number of cars.
- Tests suggest that the biased random-key genetic algorithm has better performance.

## ARTICLE INFO

*Article history:*
Received 6 December 2018
Received in revised form 6 May 2019
Accepted 10 June 2019
Available online 27 June 2019

*Keywords:*
Connected cars
Internet of Things
Scheduling
Time- and machine-dependent scheduling
Biased random-key genetic algorithm

## ABSTRACT

The current and the new generation of Internet of Things (IoT) devices present several challenges, among them the software update of legacy and new devices using wireless connections. In this paper, we study a problem of scheduling massive Firmware-Over-The-Air updates for millions of connected cars. We model this problem as a new generic problem called Time- and Machine-Dependent Scheduling Problem (TMDSP) that resembles project scheduling problems with variable-intensity activities. In the TMDSP, jobs, machine utilization, and production rates vary over time. In each period, a given job can be executed by a subset of machines with different capacities and production rates. The objective is to deploy a feasible schedule with minimum total completion time. We explore solving the problem using several approaches among than Biased Random-Key Genetic Algorithm (BRKGA), Iterated Local Search (ILS), Simulating Annealing (SA), two variations of Tabu Search (TB), and traditional genetic algorithms (TGA), in addition to a Mixed Integer Programming (MIP) model. We test the proposed approaches on a synthetic benchmark and very large real instances in IoT space. While using a commercial solver and the MIP model, we are able to solve only a few real instances (with, at most, 1,976 cars and 9,116 sectors); on the other hand, BRKGA presents significantly better results when compared to ILS, MIP solver and a simple multi-start heuristic.



## 1. Introduction

The number of wireless connected devices has been increasing exponentially in recent years. A large number of such devices belongs to a class called Internet of Things (IoT). IoT devices use the network not only for their primary purpose, such as to communicate or transmit collected data, but also for maintenance purposes, such as software or firmware updates. Because IoT devices are software-driven, it is inevitable that software updates will be required to maintain the device, address security issues, and to keep up with technological evolution. Given the widespread deployment and intended form-factor, over-the-air (OTA) updates offer the fastest, most convenient, and cost effective way of delivering software updates to these devices. For a very large and growing segment of IoT devices, such as connected cars or monitoring and control systems for intelligent transport containers, OTA is only available via cellular networks. Furthermore, these special classes of IoT devices exhibit specific network behaviors including being inherently mobile, having reduced availability on the network (1 hour/day), connecting to different access points in the network on different days, and could also be absent from the network for an extended period [1]. These

* Corresponding author.
*E-mail addresses:* cea@research.att.com (C.E. Andrade), byers@research.att.com (S.D. Byers), gvijay@research.att.com (V. Gopalakrishnan), emir@research.att.com (E. Halepovic), poole@research.att.com (D.J. Poole), lien@research.att.com (L.K. Tran), volinsky@research.att.com (C.T. Volinsky).

https://doi.org/10.1016/j.asoc.2019.105575
1568-4946/

traits represent a significant challenge for a network operator when delivering OTA software updates to these IoT devices. The software updates must be delivered in a way that maximizes the update success rate, makes efficient use of radio spectrum, minimizes the need for network capacity expansion, and ensures minimal impact to all network users. From the network operator's perspective, the OTA problem for IoT becomes a scheduling problem.

To model this OTA scheduling problem for IoT, we present the *Time- and Machine-Dependent Scheduling Problem* (TMDSP), which operates on massive software downloads to millions of devices connected to radio networks. One of the main challenges of this problem is that the devices are *mobile* and typically connect to *several access points* in the network *over time*. In addition, the server and network usually cannot control the full download process and only may suggest the start time of the download. This problem also includes other constraints such as network and server capacities, and device limitations.

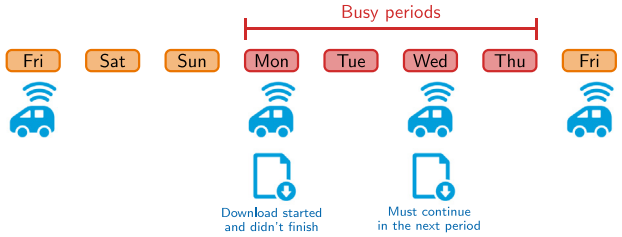
In this paper, we describe TMDSP as a generic scheduling problem aiming to model several scenarios with similar characteristics to those described earlier. However, we focus our attention to Firmware-Over-The-Air updates for connected cars. We present a Mixed Integer Programming (MIP) model, a simple multi-start heuristic, an Iterated Local Search (ILS) algorithm, a Biased Random-Key Genetic Algorithm (BRKGA) an Simulating Annealing (SA) algorithm, two variations of Tabu Search (TB), and an traditional genetic algorithm (TGA) to solve it. Our approaches were tested on the sets of synthetic and real instances of considered IoT scenarios, which usually comprise massive amounts of cars and access points. The results suggest that the BRKGA produces significantly better results when compared with the other algorithms. With commercial MIP solver, we can find solutions for instances up to 1976 cars and 9116 sectors, BRKGA can solve instances with more than 33,000 cars and 24,000 sectors, in six wall-clock hours of computation.

The structure of this paper is as follows. Section 2 describes the OTA update use case. In Section 3, TMDSP is formally described, examples are given, and a MIP model is proposed. Section 4 discusses related literature. Section 5 proposes a BRKGA to solve the TMDSP. Sections 6 and 7 describes the proposed ILS, SA, TS, and TGA. Section 8 reports the computational experiments and Section 9 concludes the paper.

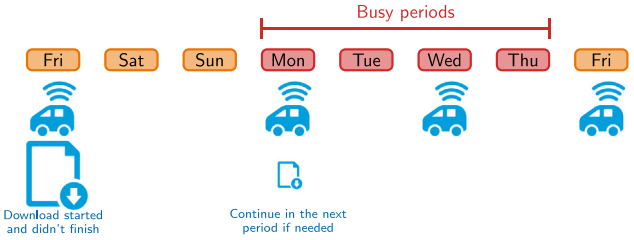
## 2. Understanding firmware-over-the-air updates for connected cars

Modern vehicles are heavily dependent on software and, due to this, more than five million manufacturer recalls made in the United States in 2015 were for software only [2]. Assuming that, at the car dealerships, the per-service cost is roughly $100, the cost for software-only issues reached half of a billion dollars only in 2015. Moreover, specialists foresee that 90% the cars will be connected by the year 2020 [3]. Connected cars are, in general, very different from other connected devices such as smartphones. While a smartphone is typically connected to a network all the time, connected cars rarely appear on the network, on average of one hour per day, when their engines are running [4]. Such behavior drastically limits the opportunities to download software. Furthermore, connected cars usually have hardware with limited networking capabilities [1]. Given that cars last about 11 years on average [5], in contrast to smartphones whose average age is 4.7 years [6], it is safe to assume that manufacturers and network operators must deal with a substantial number of legacy hardware.

Manufacturers are trying alternative ways to update software and address recalls other than at the dealership. Some manufacturers provide USB sticks with software to the customers, but




(a) Undesirable download periods.




(b) Desirable download periods.

**Fig. 1.** Illustration of Firmware-Over-The-Air (FOTA) download over a busy week assuming that the car appears in the network only a few days. Monday to Thursday are considered busy days (in red) for the network. The size of the download icon indicates the volume of download in the day.

there are reports that this approach often fails due to reliance on human intervention [7]. OTA using home Wi-Fi is also implemented, but it fails when the car is out-of-range, such as in underground garages or remote parking lots, as well as reliance on owners setting up Wi-Fi connections. The use of satellite is an expensive option that requires additional hardware and may not offer adequate control and download verification. Therefore, these alternative strategies usually open a substantial liability gap that the manufacturers do not want to take when addressing critical recalls.

A viable approach is the Firmware-Over-The-Air (FOTA) over cellular networks. Cellular networks are already widespread, often covering all inhabited areas and traffic corridors. It is estimated that $35 billion in annual savings will be realized by 2022 using FOTA [8]. With the expected arrival of 5G, cellular networks will be able to provide high data rates allowing quick updates in some areas. However, since existing connected cars on the roads have legacy hardware, they will not be able to take advantage of such bandwidth, and will have to use 4G cellular infrastructure.

Other two critical issues for FOTA are the availability of the car in the network, and the very restrictive scenario for updating. Most manufacturers require that FOTA updates be only carried out when the *engine is turned on*, limiting the time for download drastically. It happens that, in general, *cars appear in the cellular network during rush hours when the network is also highly loaded* [1].

Fig. 1 illustrates a car that appears in the network on Mondays, Wednesdays, and Fridays only, assuming that the busy periods occur from Monday to Thursday only. In Fig. 1(a), the car starts to download the update on Monday but cannot complete it, so it continues on the next available period which is Wednesday.

Note that the car will resume the download automatically when the network becomes available again, the network cannot postpone or otherwise control the started download. Hence, in this example, the car does not skip Wednesday, since it appears in the network that day. Note that both download periods occur while the network is busy, which reduces the download speed, and consequently, the completion rate of an update campaign, while also competing with regular user traffic. This situation

not only degrades the network performance but also raises the liability of the manufacturer if not completing an urgent update in a timely manner, for example. Fig. 1(b) shows a more desirable situation, where the car is allowed to start the download Friday, a non-busy network day, and almost finishes there. It may happen that a small amount of download remaining must be downloaded in the next period, which is a busy Monday. However, the volume of data is small which impacts the network less and increases the probability of completion.

The amount of data downloaded varies according to time and network conditions. First, the time spent under a cellular radio varies per car and period, i.e., a car spends different amounts of time under the same cellular radio at different times of the day. Second, the amount of data delivered by the cellular radios varies with network conditions, such as resource capacity and load. Resource usage efficiency in 4G LTE is measured using Physical Resource Block (PRB) utilization level and the data volume in a given period. The PRB is the amount of physical spectrum used to transport data for a scheduling unit of time, making a total number of PRBs the capacity of the radio cell. The load is usually given as a percentage of the total available number of PRBs. Therefore, a user or group of users cannot use more than 100% of the available PRBs, and the level of about 80% is often considered a maximum desirable long-term average [9]. Since the car is traveling through a radio cell for a given time, the amount of data it downloads is directly related to the amount of PRBs it consumes from the cellular network. For more details on the LTE network measures and modeling, see [1,10].

Therefore, considering all the aforementioned constraints, the goal is to create a download schedule that will minimize overall completion and the time downloads happen during the busy network hours. To achieve that, we base the solution on known network utilization patterns and the historical data of each car's activity in the network. The outcome is the schedule comprising of download start time for each connected car.

## 3. Formal definition and models

Since the download scheduling problem described in Section 2 can also be applied to other scenarios such as software update for mobile industrial equipment or smart containers, we model it as a generic scheduling problem called the *Time- and Machine-Dependent Scheduling Problem* (TMDSP). In TMDSP model, we consider that the end devices are the jobs that want to use some capacity production from the radio cells in the cellular network, modeled as server machines.

Let $J$ be a set of jobs and $M$ be a set of machines. Consider $T = \{0, \ldots, t^{final}\}$ as the set of periods. For each machine $m \in M$ and period $t \in T$, $c_{mt} \in \mathbb{R}^+$ gives the maximum capacity and $\eta_{mt} \in \mathbb{R}^+$ gives the utilization level for machine $m$ in period $t$. We consider that $c_{mt}$ is an intrinsic metric of the machines, and the unique piece of information we can obtain from them per period. In this sense, $\eta_{mt}$ describes how much of the resource $c_{mt}$ can be transformed into production units. In the FOTA context, $c_{mt}$ is percentage of physical resource blocks available in a cell tower sector and $\eta_{mt}$ is the number of bytes per physical resource block. In such case, if $c_{mt} = 30$ and $\eta_{mt} = 10$, the yield is 300 bytes. To generalize this model, we consider two metrics: a generic *machine utilization unit* (muu) and a generic *production unit* (pu). In FOTA, the machine utilization unit is given in percentage of physical resource blocks (PRB%), and production unit is given in bytes (or megabytes, MB). The maximum capacity $c_{mt}$ is given in muu (PRB%). Therefore, $\eta_{mt}$ is given in pu/muu (MB/PRB%). A global resource capacity $r^{max} \in \mathbb{R}^+$ (given in pu (MB)) must be enforced per period $t$ across all machines, i.e., the total machine capacity utilization cannot exceed this threshold. In FOTA, this limit constrains the volume of data downloaded in a period. This bound is usually given per intermediary Internet of Things (IoT) control systems. Also, a maximum number of jobs per period $k^{max} \in \mathbb{N}$ is enforced.

**Table 1**
Sets and constants definitions.

| | |
|---|---|
| $J$ | Set of jobs/cars |
| $M$ | Set of machines/radio cells or sectors |
| $T$ | Set of periods |
| $c_{mt}$ | Maximum capacity per machine (muu; PRB%) |
| $\eta_{mt}$ | Utilization rate (pu/muu; MB/PRB%) |
| $q_{jmt}$ | Capacity consumption (muu; PRB%) |
| $d_j$ | Total demand (pu; MB) |
| $r^{max}$ | Global resource capacity (pu; MB) |
| $r^{min}$ | Minimum resource consumption (pu; MB) |
| $k^{max}$ | Maximum number of jobs per period |

Each job $j$ has a capacity consumption $q_{jmt} \in \mathbb{R}^+$ required from machine $m$ in period $t$ (given in muu or PRB%). However, job $j$ must use, at least, $r^{min} \in \mathbb{R}^+$ of the global resources (given in pu). In FOTA, this lower bound sets a minimum amount of data that a car must download when connected to the network. Note that $q_{jmt}$ is closely related to the processing time in classical scheduling problems, although it does not reflect time itself. It is worth stressing that in the TMDSP, the capacity consumption is strictly required by the jobs. In other words, let $c'_{mt} \leq c_{mt}$ be the remaining capacity on machine $m$ in period $t$. Job $j$ can be scheduled in a period $t$ if and only if job $j$ consumes *exactly* $q_{jmt}$, implying that $q_{jmt} \leq c'_{mt}$ for all machines $m$ (with the exception of a last period where the remaining demand can be smaller than these limits). Such restriction comes directly from the connected cars and other IoT devices. The network cannot control how much the car can download per period, since this is controlled by capacity and transport protocols. Since the car appears in the network with a given probability, the network must provide enough capacity for the download. Indeed, $q_{jmt}$ is computed using historical car data, as described in [1]. Each job also has a total demand $d_j \in \mathbb{R}^+$ to be fulfilled given in pu (i.e., download size given in MB). Table 1 summarizes these definitions.

In some cases, $r^{max}$, $r^{min}$, and $k^{max}$ can vary according to the period and must be indexed accordingly. However, these capacities are constants, or described as very simple functions, which do not significantly change the model presented in this paper. Indeed, all techniques proposed here can be applied to non-scalars $r^{max}$, $r^{min}$, and $k^{max}$.

In a real network, a job can appear to multiple machines and multiple times within a period (as a driving pattern of cars), as shown in Fig. 2(a). Each appearance is defined as a *session*. For instance, job A has two sessions using 60 and 40 muu from machine M1, and one session on machine M2 using 30 muu, both in period 1. In period 2, job A can use only 60 muu from machine M2. In period 3, job A has ten small sessions on machine M1. Although such configurations can exist, the order that a job is served in within a period does not matter for most applications. Therefore, we assume that we can pre-process the instance, merging sessions in the same machine within a period, for each job, as shown in Fig. 2(b).

A job may require several periods to have its demand satisfied. For example, suppose job A has a demand of 250 pu. Note that, even if we assign machines M1 and M2 exclusively to serve job A, it is not possible to serve all demand with three periods since the machines can serve, at most, 200 pu for job A. In such cases, additional periods are required by cycling the previous periods. In the last example, job A has to use the fourth period to finish. We consider that period 4 mimics period 1, period 5 mimics period 2, and so forth. Such model is known as *rolling horizons*.

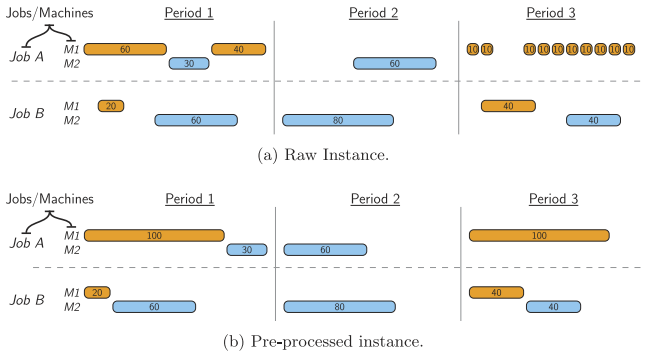



**Fig. 2.** An example of an input instance. The orange blocks (first and third lines with boxes) represent sessions on machine M1. The blue blocks (second and fourth lines with boxes) represents sessions on machine M2. The number inside the blocks describes the capacity consumption $q_{jmt}$. We consider that the periods have the same duration although they are not depicted using the same scale.

The machines are shared resources and can be used by several jobs simultaneously within a period if the total capacity is respected. Not all machines are available in all periods. Again, simple pre-processing can reduce the size of the problem by disregarding job sessions for periods that a particular machine is not available.

A critical aspect of this problem is that the jobs have a large degree of autonomy, and that a fine-grained control over its execution is not possible. Once a job starts, it cannot be stopped or controlled in terms of which period to execute in and, therefore, this model can be considered *non-preemptive*. *Also, a job must be able to use all capacity it requires from the machines in a particular period.* In other words, the residual capacity in the machines should be larger than the capacity utilization demanded by the job in each period. This fact models the car admission in the network, and the "continuous" download process explained earlier in FOTA example of Section 2.

Note that the problem also defines a minimum resource consumption $r^{min}$, which should be respected for all jobs. However, it is possible that the capacity consumed by a job, given a machine and a period, be less than $r^{min}$. When that happens, we cannot schedule such jobs. In other words, a job $j$ cannot be scheduled if $q_{jmt} < r^{min}/\eta_{mt}$, for all $m \in M$ and $t \in T$.

Algorithm 1 brings a high-level pseudocode that shows the main constraints to schedule jobs. Assume that $y_{imt} \in \mathbb{R}^+$ is the amount of capacity used by job $j$ from machine $m$ on period $t$. Also assume that $a_{jt} \in \{0, 1\}$ indicates that job $j$ is scheduled (active) on period $t$. Note that, when schedule, job $j$ will consume $q_{jmt}$ capacity of machine $m$ on period $t$ satisfying $q_{jmt} \times \eta_{mt}$ of its demand.

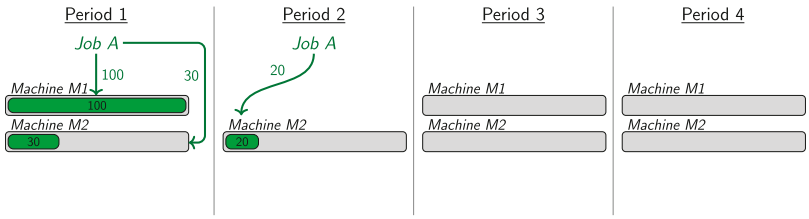
With respect to the machine utilization, we have two models in practice: in the first model, a job can use only one machine in each period as in other classic scheduling problems. We call this variation *Time- and Machine-Dependent Scheduling Problem with Single Task Period* (TMDSP-ST). The other model, which occurs in practice frequently, is the one where a job can use multiple machines in the same period. Such variation is called *Time- and Machine-Dependent Scheduling Problem with Multi-Task Period* (TMDSP-MT). Modeling FOTA as either TMDSP-ST or TMDSP-MT depends on the granularity of the time used to track the connections. In practice, a car can only connect to one radio cell at a time, for a few seconds or minutes, configuring the TMDSP-ST model. However, such time granularity is too small, and hard to work in practice. Therefore, periods (bins) of 15 or 30 min are used instead [1]. In such periods, a car can hand off between several cells, and download data from them. Therefore, in this paper, we model the FOTA problem as TMDSP-MT.

Consider an example of a solution for TMDSP-ST. The jobs, machines, and sessions are shown in Fig. 2. We assume that both machines have capacity of 100 muu in all periods, job A demands 150 pu, and job B demands 200 pu. Suppose that the utilization rate is 1 pu/muu for all machines in all periods. Fig. 3 shows a schedule where job A is scheduled before job B. The schedule starts with the assignment of 100 muu of machine M1 and 30 muu from machine M2 for job A (Fig. 3(a)). Note that this is strictly necessary since $q_{\text{A,M1,1}} = 100$ and $q_{\text{A,M2,1}} = 30$. One must remember that it is not allowed to assign less capacity that required per the job: we cannot assign only 80 muu on machine M1, or 25 muu on machine M2, or even use either M1 or M2 alone. Since the total demand is not fulfilled yet (100 muu $\times$ 1 pu/muu + 30 muu $\times$ 1 pu/muu = 130 pu), the schedule for job A extends to period 2, where the remaining demand (20 pu/1 pu/muu = 20 muu) is assigned to machine M2. Once job A is scheduled, it is time to schedule job B (Fig. 3(b)). The first step tries to schedule job B on period 1. However, this is not possible: job B requires $q_{\text{B,M1,1}} = 20$ from machine M1 but M1 capacity is already depleted by job A. Even though job B could use the spare capacity on machine M2, the problem states that a job must use all capacity it requires on the period. *Therefore, the inability to schedule job B on machine M1 impairs its scheduling on M2 on period 1.* Therefore, we skip for period 2 where we can schedule the required capacity ($q_{\text{B,M2,2}} = 80$) on machine M2. From this point, we need to use more two periods to fulfill job B demand. On period 3, all capacity from machines M1 and M2 are used. On period 4, both machines are used, although we could use only machine M2 to fulfill the demand since M2 enough room. Fig. 4(a) shows the final schedule when scheduling job A before job B.

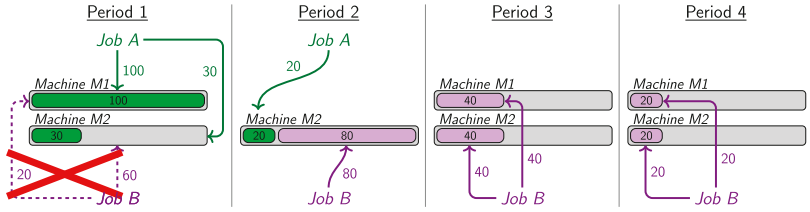
When scheduling job B before job A, we have similar behavior (Fig. 4(b)). On period 1, even job B does not use all capacity from machines M1 and M2, the spare capacity on M1 is not enough to accommodate job A requirements. On period 2, job B depletes machine M2 which also blocks job A to be scheduled there. Therefore, job A only finds room on period 3 on machine M1. This is only possible because job B is scheduled on machine M2 on this period. If job B is scheduled on machine M1, job A would be pushed to periods already. We can see that the results are susceptible to the scheduling and allocation order due to the unique constraints of the problem.

**Algorithm 1:** General conditions for scheduling on TMDSP.

```
for each job j ∈ J do
    Starting from period t = 0;
    while d_j is not fulfilled do
        if the following conditions are true:
            ▷ All capacity required per job j: for each
              machine m ∈ M, q_jmt ≤ Σ_{i∈J} y_imt;
            ▷ Machine capacity: for each machine m ∈ M,
              Σ_{i∈J} y_imt ≤ c_mt;
            ▷ Maximum global resource utilization:
              Σ_{i∈J, m∈M} y_imt + q_jmt ≤ r^max;
            ▷ Maximum number of jobs per period:
              Σ_{i∈J} a_it + 1 ≤ r^max;
        then schedule d' demand of job j on period t such
             that: d' = Σ_{m∈M} η_mt q_jmt;
        else go to next period t = t + 1;
```

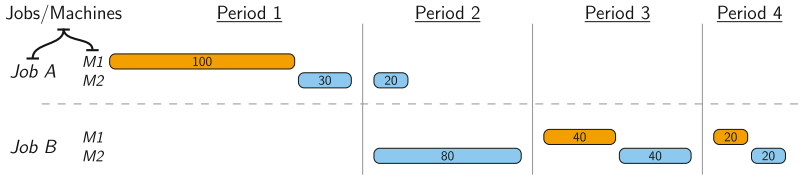



(a) Job A is scheduled first and depleted machine M1 on period 1.

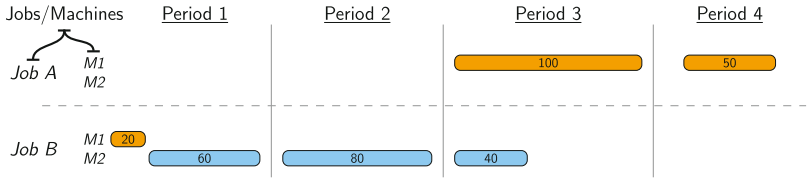



(b) Job B is scheduled in sequence. Note that job B can be scheduled in period 1 since it cannot use machine M1.

**Fig. 3.** Scheduling job A before job B. Assume that: both machines have capacity of 100 muu and utilization rate of 1 pu/muu in all periods; job A and job B demands 150 pu and 200 pu, respectively; jobs' capacity consumptions ($q_{jmt}$) are described on Fig. 2(b).




(a) Scheduling first job A, then job B.




(b) Scheduling first job B, then job A.

**Fig. 4.** Two different schedules based on the order that the jobs are scheduled. Again, we consider that the periods have the same duration although they are not depicted using the same scale.

In real applications, some jobs may appear very late, and it is not uncommon that a job defines the makespan which is the time when all jobs have finished. Due to this reason, the metric to minimize in the TMDSP is the total completion time, which is the sum of completion times for all jobs (equivalent to minimizing the average completion time in this case). In the previous example, both schedules 4(a) and 4(b) have makespan of 4. However, the total completion time for schedule 4(a) is 6 while for schedule 4(b) is 7. Section 8.4 discusses the relationship between the total completion time and the makespan for FOTA instances.

### 3.1. Mixed integer programming model

We now model the TMDSP as a mixed integer program (MIP). We deliberately chose to use set notation in our MIP model, which is not common in the scheduling literature. The reason is that TMDSP and its variants are fairly sparse problems and the jobs are not available for each machine and each time, i.e., matrix $q$ is fairly sparse. Matrix $c$ may be sparse as well in some cases.

Let $t^{max}$ be a valid upper bound of the makespan. Let $\tilde{T} = \{0, \ldots, t^{max}\}$ be a set of periods. Consider the following decision variables:

- $x_{jmt} \in \{0, 1\}$ for $j \in J$, $m \in M$, and $t \in \tilde{T}$: $x_{jmt} = 1$ indicates that job $j$ is scheduled to be executed on machine $m$ in period $t$; $x_{jmt} = 0$ otherwise;
- $y_{jmt} \in \mathbb{R}^+$ for $j \in J$, $m \in M$, and $t \in \tilde{T}$: amount of capacity from machine $m$ consumed by job $j$ in period $t$;
- $s_{jt} \in \{0, 1\}$ for $j \in J$ and $t \in \tilde{T}$: $s_{jt} = 1$ if job $j$ starts in period $t$;
- $a_{jt} \in \{0, 1\}$ for $j \in J$ and $t \in \tilde{T}$: $a_{jt} = 1$ if job $j$ is scheduled/active in period $t$;
- $f_{jt} \in \{0, 1\}$ for $j \in J$ and $t \in \tilde{T}$: $f_{jt} = 1$ if job $j$ (pseudo-)finishes in period $t$, i.e., this variable may not reflect the actual finish time as explained later. One can argue that $f_{jt}$ is not necessary. However, as explained later, $f_{jt}$ is essential to avoid over-utilization of the machines;
- $0 \leq C_j \leq t^{max}$ for $j \in J$: describes the completion time of each job.

Since the number of periods in a solution $t^{max}$ can be larger than the number of periods in the job sessions $t^{final}$ (rolling horizon), we define $\hat{t} = t \mod t^{final}$ for all $t \in \tilde{T}$ to be used in the following MIP:

$$\min \quad \sum_{j \in J} C_j \tag{1a}$$

$$\text{s.t.} \quad C_j \geq t a_{jt} \qquad \forall j \in J, t \in \tilde{T} \tag{1b}$$

$$\sum_{m \in M} x_{jmt} \leq 1 \qquad \forall j \in J, t \in \tilde{T} \tag{1c}$$

$$\sum_{t \in \tilde{T}} s_{jt} = 1 \qquad \forall j \in J \tag{1d}$$

$$a_{jt} \geq \sum_{m \in M} x_{jmt} / |M| \qquad \forall j \in J, t \in \tilde{T} \tag{1e}$$

$$a_{j0} = s_{j0} \qquad \forall j \in J \tag{1f}$$

$$a_{jt} \leq s_{jt} + a_{jt'} \qquad \forall j \in J, t \in \tilde{T} \setminus \{0\}, \; t' = \max\{t'' < t : q_{jmt''} > 0\} \tag{1g}$$

$$f_{jt^{max}} = a_{jt^{max}} \qquad \forall j \in J \tag{1h}$$

$$f_{jt} \leq 1 - \sum_{t' > t} a_{jt'} / |\tilde{T}| \qquad \forall j \in J, t \in \tilde{T} \setminus \{t^{max}\} \tag{1i}$$

$$y_{jmt} \leq q_{jm\hat{t}} x_{jm\hat{t}} \qquad \forall j \in J, m \in M, t \in \tilde{T} \tag{1j}$$

$$y_{jmt} \geq q_{jm\hat{t}} (x_{jmt} - f_{jt}) \qquad \forall j \in J, m \in M, t \in \tilde{T} \tag{1k}$$

$$\sum_{j \in J} y_{jmt} \leq c_{m\hat{t}} \qquad \forall m \in M, t \in \tilde{T} \tag{1l}$$

$$\sum_{m \in M} \sum_{j \in J} \eta_{m\hat{t}} y_{jmt} \leq r^{max} \qquad \forall t \in \tilde{T} \tag{1m}$$

$$\sum_{j \in J} a_{jt} \leq k^{max} \qquad \forall t \in \tilde{T} \tag{1n}$$

$$\sum_{m \in M} \sum_{t \in T} \eta_{m\hat{t}} y_{jmt} = d_j \qquad \forall j \in J. \tag{1o}$$

Objective Function (1a) computes the total completion time given by the sum of individual completion times computed by Constraint (1b). Constraints (1c)–(1g) are classic constraints for scheduling problems. Constraint (1c) ensures that a job uses only one machine at time. This constraint is used on TMDSP-ST but should be dropped in TMDSP-MT. Constraint (1d) guarantees that each job has only one start time, and Constraint (1e) ensures that a job is active when scheduled on a machine. Constraints (1f) and (1g) use the activation variable to avoid preemption. Constraint (1f) is a simple trick to avoid cycling on the activation sequence. Constraint (1g) allows activating job $j$ in period $t$ if $t$ is the starting time, or $j$ is active on the previous period $t'$. Note that the notation is a little bit different from the conventional form that uses $t-1$ instead of $t'$ as defined. For the most scheduling problems in the literature, no preemption means to skip no periods during the full execution. In TMDSP, no preemption means that when scheduled, a job must use the full capacity it requires. A job may not be available for each period, and $t-1$ may not reflect the correct period. Therefore, $t'$ is the largest period before $t$ such that job $j$ is available and can be executed.

The remaining constraints are, in some sense, more specific to TMDSP. First, it is necessary to identify the final period to correctly account for the capacity utilization. Constraints (1h) and (1i) act similarly to previous constraints. Constraints (1h) tie the last active period to the last possible finish time. For all other periods, Constraint (1i) allows $f_{jt}$ be the one only when the job has no activity after $t$. Constraint (1j) allows capacity utilization only when the job is scheduled in the machine in the given period. Constraint (1k) ensures that there is enough room for the job to be executed using its maximum capacity consumption for given machine and period, except the last period. Note that in the last period, the job may use less than the required capacity to finish. If we do not use this strategy (and variables $f_{jt}$ which are used just for this purpose), we may lose optimality, since job $j$ may use more capacity than necessary, and may push forward other jobs that could use this spare capacity otherwise. There is a caveat with $f_{jt}$: if the remaining demand is exactly $y_{jmt}$, it is not necessary that $f_{jt} = 1$ since Constraint (1k) can be satisfied without that. Therefore, $f_{jt}$ may not represent the actual finish time. For this reason, we call it pseudo-finishing time and we do not use $f_{jt}$ in the objective function. Note that we cannot merge Constraints (1j) and (1k) in an equality due to the last period since a job may require more than zero but less than $q_{jmt}$ to finish as commented earlier. Constraints (1l)–(1n) specify capacity restrictions. Constraint (1l) limits the machine utilization per period and Constraint (1m) ensures that the global resource usage per period is respected. Constraint (1n) limits the number of jobs per period. Finally, Constraint (1o) guarantees that the job demand is fulfilled. Note that MIP (1a) does not incorporate constraints over the minimum resource consumption $r^{min}$. The jobs that do not respect such limit are filtered out of the input, and marked as non-schedulable, as commented earlier.

Some works in the literature [11] choose to penalize the resource over-utilization instead of computing the correct amount as we do using variables $f_{jt}$ and Constraints (1j) and (1k). While this approach reduces the number of constraints and eliminates the need for $f_{jt}$, it causes significant side effects for FOTA scheduling. In our preliminary experiments, the total completion time was more than 30% longer in the model without $f_{jt}$ than compared to the model presented here.

Note that Constraint (1i) is very dense, slowing down the optimization and causing high memory use with large instances. Therefore, we can substitute it by telescopic sums. For each job $j$ and period $t$, we introduce a variable $\sigma_{jt}$ to accumulate the sums. Instead of Constraint (1i), we introduce the following constraints:

$$\sigma_{jt^{max}} = a_{jt^{max}} \qquad \forall j \in J, t \in \tilde{T} \tag{1i'}$$

$$\sigma_{jt} = a_{jt} + \sigma_{j,t+1} \qquad \forall j \in J, t \in \tilde{T} \setminus t^{max} \tag{1i''}$$

$$f_{jt} \leq 1 - \sigma_{jt} / |T| \qquad \forall j \in J, t \in \tilde{T}. \tag{1i'''}$$

Although the number of variables and constraints increases, Constraints (1i′)–(1i‴) are much sparser and easier to handle that Constraint (1i).

The proposed MIP is a time-indexed formulation which may not be as effective as formulations using positional variables according to Della Croce et al. [12]. However, since each job execution is time dependent, permutation-only-based formulations may be not appropriate.

## 4. Literature review

In the Time- and Machine-Dependent Scheduling Problem (TMDSP), although the machines perform the same functions, they operate independently, characterizing a problem with unrelated parallel machines. Using the standard notation introduced by Graham et al. [13], we can describe TMDSP as $R|q_{jmt}|\sum C_i$, where $R$ stands for unrelated parallel machines, $q_{jmt}$ for time- and machine-dependent capacity consumption, and $\sum C_i$ for minimizing the total completion time.

TMDSP is a variation of the Resource Constrained Project Scheduling Problem (RCPSP) and shares characteristics with scheduling problems with time-dependent processing times and scheduling problems with deterministic machine availability constraints. RCPSP is a widely studied problem proposed by Blazewicz et al. [14], where it is proven to be $\mathcal{NP}$-hard. Since RCPSP can be reduced in polynomial time to TMDSP, one can argue that TMDSP is also $\mathcal{NP}$-hard. The literature about RCPSP is extensive and a survey of its application can be found in [15].

TMDSP shares some components with RCPSP with variable intensity activities. In this problem, one wants to allocate continuously divisible resources to activities over the time varying their quantities such that a set of temporal and resource constraints are satisfied, and an objective function is minimized [11,16,17]. The main difference between RCPSP with variable intensity activities and TMDSP is that in the RCPSP, the activities have a strict order to obey, usually given by a precedence graph. In TMDSP, such an order does not exist. Indeed, the jobs in TMDSP can use any machine available according to the period and consumption requirements. Therefore, TMDSP has a solution space much larger than RCPSP with variable intensity activities. This RCPSP variation also considers a cost of over-using a resource. As explained in Section 3.1, such a strategy increases the total completion time drastically for the FOTA scheduling. Therefore, TMDSP explicitly avoids the over-use of resources.

Other literature about scheduling problems with time-dependent processing times assumes that the processing time is a well-defined function such as linear or piecewise function, but even for such cases, the variations, for the most part, are $\mathcal{NP}$-hard. Note that in TMDSP, each job has a different processing requirement (capacity consumption $q_{jmt}$) for each machine and timebin, which introduces high non-linearity and differentiates TMDSP from other problems proposed previously. Kong et al. [18] studied a scheduling problem with non-linear processing times but on a single and unrelated parallel machines. Although non-linear, the processing times are described using a well-defined exponential function. To solve such a problem, the authors proposed a hybrid approach with shuffle frog leap algorithm and variable neighborhood search algorithm. Liu et al. [19] investigated a similar problem where the job processing time is a decreasing function of its starting time. They proposed a hybrid algorithm combining cuckoo search algorithm and self-adaptive differential evolution. Woo and Kim [20] proposed a matheuristic combining simulating annealing, genetic algorithms, and mixed integer programming for a scheduling problem with time-dependent deterioration and multiple rate-modifying activities. The different from the previous works is that each machine has a particular but yet well-defined deterioration rate. Moreno-Camacho et al. [21] presented an interesting problem where the actual processing time is a function of the scheduling sequence to represent the decline in workers' performance. Therefore, instead to have a rigid deterioration function as the previous works, here the scheduling order dictates the pace of the worker/machine performance. The authors compared several mixed integer programming models through extensive computational experiments.

From the resource consumption perspective, TMDSP requires a renewable resource whose availability varies over time (the global resource capacity $r^{max}$). Kuschel and Bock [22] investigated the Weighted Capacitated Planned Maintenance Problem with presents such global resource constraints. For some variants a constant number of maintenance activities, the authors presented strongly-polynomial time algorithms. For general versions, they used a multi-state tabu search to solve the problems. In [23], the authors applied a simulation approach for RCPSP for renewable resource-constrained change propagations. Their strategy is based on a forward-propagate-later-rollback simulation used to compute the fitness of individuals by a genetic algorithm during the optimization. Muritiba et al. [24] presented a path-relink approach for Multi-mode RCPSP renewable and nonrenewable resource. They approach also includes lower and upper bound computations for efficient local neighborhood searches.

The TMDSP also has similarities with scheduling problems with deterministic machine availability constraints. In such problems, the machines are not available during specific periods for reasons like maintenance. Ma et al. [25] presented a survey about such problems. The reader may refer to [26–28] for recent works. The main difference between TMDSP and other problems with availability constraints is that in the latter, when a machine is unavailable, no job can use it. In TMDSP, the machines may be unavailable for a subset of jobs in a given period. Indeed, the interpretation is the opposite way: in given period of time, the jobs can use subset of machines only.

## 5. Solution using Biased Random-Key Genetic Algorithm

Because of the large-scale nature of practical instances, it can be difficult to solve the TMDSP using exact approaches. Therefore, we use the Biased Random-Key Genetic Algorithm (BRKGA) as the optimization framework. The BRKGA was proposed by Gonçalves and de Almeida [29] and Ericsson et al. [30], and has been shown to be useful for a wide variety of combinatorial problems such as scheduling [31,32], parcel packing [33], vehicle routing [34,35], hub location [36], clustering [37] and machine learning [38], winner determination in combinatorial auctions [39], placement of virtual machines in data centers [40], and complex network design [41]. BRKGA has also been used to check the feasibility for instances from mixed integer programming problems for which feasible solutions are challenging to find [42].

The BRKGA works with a set of solutions $\mathcal{P}$ called *population*, where each solution (also called *individual*) is represented by a vector $v \in [0, 1]^n$, called *chromosome*, for a given $n$. The population is partitioned into the *elite set* $\mathcal{P}_e$ and the *non-elite set*. The solutions in the elite set are the best of the whole population according to some performance metric (*fitness*). The non-elite set contains the remaining individuals ($\mathcal{P} \setminus \mathcal{P}_e$). On each BRKGA iteration (generation), the elite set is copied to the next generation population and a small percentage of random solutions (called *mutants*) is also included in this new population. Considering that the number of elite individuals plus the number of mutants are less than the size of the population, the latter is completed with offspring from the combination (*mating*) of a randomly chosen individual from the elite set and a randomly chosen individual from the non-elite set. The mating is done using biased uniform crossover where a component is taken from the elite individual with probability $\rho$, otherwise the component is taken from the non-elite individual. Since an individual is a vector $v \in [0, 1]^n$, to obtain a solution, it is necessary to apply a function $f : [0, 1]^n \to \mathcal{S}$ that maps a vector $v$ to a solution in the space $\mathcal{S}$, which is the space of solutions of the problem to be solved. Typically, $f$ also computes the fitness of the individual. Function $f$ is called *decoder*. For a detailed description, see Gonçalves and Resende [43].

### 5.1. Decoding the TMDSP

Consider that the jobs are indexed by $\{1, \ldots, |J|\}$. Each allele/key $v_j$ of the chromosome $v \in [0, 1]^n$, where $n = |J|$, is paired with the job $j$. The main idea is to allocate the jobs in a given order, into slots in the current scheduling horizon. A horizon is a set of periods $\{1, \ldots, t^{final}\}$ within which jobs can be scheduled,

and multiple horizons may be necessary to complete the entire schedule.

Assume that for each job $j$, the non-decreasing period-capacity ordered list $PC_j = \{(t, m)\}_{t\in T, m\in M}$ in such way that $(t, m) \leq (t', m')$ if and only if

- either $t < t'$;
- or $t = t'$ and $q_{jmt} > q_{jm't'}$;
- or $t = t'$, $q_{jmt} = q_{jm't'}$ and $m < m'$.

In other words, we sort first by the shortest periods, then by the largest capacity consumptions, and finally by the smallest machine IDs. We disregard pairs $(t, m)$ for which $q_{jmt} = 0$.

For each horizon $h$, period $t$, and machine $m$, we assume that

- $\mathsf{MachineLoad}_{htm}$ is the load of machine $m$ (in mmu);
- $\mathsf{PeriodLoad}_{ht}$ is the total load of period $t$ (in pu);
- $\mathsf{NumJobs}_{ht}$ is the number of jobs in period $t$.

Algorithm 2 describes a decoder procedure that translates a chromosome to a schedule, and Fig. 5 depicts the first steps of this algorithm. First, we order the jobs according to the value of their keys (line 1). Then, MachineLoad, PeriodLoad, and NumJobs are initialized to zero (line 2). For each job in such order, we create a temporary schedule JobSched initializing it as an empty list (line 4). JobSched holds tuples $\langle h, t, m, \ell\rangle$ that describe the horizon, the time, the machine, and the used capacity. We also start from the first horizon and the full demand (lines 5–6).

The main loop (lines 3–26) consists of finding available slots to schedule the job and distribute the demand among these slots. For each possible slot in the period-capacity ordered list $PC$, we compute the maximum possible load (maxload) which is the minimum among the remaining demand (rdemand), the remaining global capacity for that period, and the capacity that should be used on that machine (line 9). Next, we compute the residual capacity of the machine (residual) assuming that the job is scheduled in that slot (line 10). Note that rdemand is given in production units (pu) and residual is given in machine utilization units (muu, see Section 3).

In lines 11–15, we verify if any of the problem constraints are violated. If the residual is negative, we have no room to accommodate the job demand in that particular machine and period. Note that if we assume preemption, this constraint can be removed. Moreover, the job cannot be scheduled when the maximum number of scheduled jobs per period is reached, or when the global resource capacity is exceed, or the remaining demand is larger than the global minimum resource consumption but the amount to be used (maxload) is smaller than this value. In all these cases, the temporary schedule for the current job is reset and the next period is attempted. Otherwise, the algorithm schedules the job by adding a scheduled slot in JobSched and adjusting the remaining demand (lines 17–18). If rdemand $> 0$, the algorithm iterates over the next periods trying to satisfying the remaining demand. If there are no more periods in the current horizon that can accommodate the demand, the next horizon is used (line 21). Once all demands have been satisfied, the schedule for the current job is committed by updating all matrices (lines 22–25) and adding to the final schedule (line 26).

Algorithm 2 creates "left-shifted" schedules which are schedules where jobs starts as soon as possible when capacity is available [44]. With the final schedule, we may compute the makespan or the total completion time. As discussed in Section 3, the total completion time is more accurate for the applications that TMDSP models, and we use this metric as performance metric in the BRKGA.

Algorithm 2 must iterate over all jobs, and in the worst case, each job must use all its available slots (maybe multiple times)

**Algorithm 2:** Building a schedule – decoder.

**Input**: Chromosome/vector $v \in [0, 1]^n$ where $n$ is the number of jobs; All data and parameters defined on Section 3.

**Output**: Feasible schedule and the total completion time.

1 Let $\pi$ to be a permutation of jobs induced by the non-increasing order of corresponding keys in $v$;
2 Initialize MachineLoad, PeriodLoad, and NumJobs as zero-matrices;
3 **foreach** $j \in \pi$ *in the given order* **do**
  /* JobSched holds tuples $\langle h, t, m, \ell\rangle$ that describe the horizon, the time, the machine, and the used capacity. */
4  JobSched $\leftarrow$ empty list;
5  $h \leftarrow 1$;
6  rdemand $\leftarrow d_j$;
7  **while** rdemand $> 0$ **do**
8  **for** $(t, m) \in PC_j$ *in the given order* **do**
9  maxload $\leftarrow \min(\text{rdemand}, \eta_{mt} \times q_{jmt})$;
10  residual $\leftarrow$ $c_{mt} - \mathsf{MachineLoad}_{htm} - (\text{maxload}/\eta_{mt})$;
11  **if** (residual $< 0$) **or** ($\mathsf{NumJobs}_{ht} = k^{max}$) **or**
12  (maxload $+ \mathsf{PeriodLoad}_{ht} > r^{max}$) **or**
13  (rdemand $> r^{min}$ **and** maxload $< r^{min}$) **then**
14  rdemand $\leftarrow d_j$;
15  JobSched $\leftarrow$ empty list;
16  **else**
17  rdemand $\leftarrow$ rdemand $-$ maxload;
18  Append tuple $\langle h, t, m, \text{maxload}\rangle$ to JobSched;
19  **if** rdemand $= 0$ **then**
20  Go to line 22;
21  $h \leftarrow h + 1$;
22  **for** $\langle h, t, m, \ell\rangle \in$ JobSched **do**
23  $\mathsf{MachineLoad}_{htm} \leftarrow \mathsf{MachineLoad}_{htm} + \ell$;
24  $\mathsf{PeriodLoad}_{ht} \leftarrow \mathsf{PeriodLoad}_{ht} + \ell$;
25  $\mathsf{NumJobs}_{ht} \leftarrow \mathsf{NumJobs}_{ht} + 1$;
26  Add JobSched to the final schedule;
27 **return** *The final schedule and the total completion time.*

to ensure that its demand is fulfilled. Therefore, the running-time complexity depends on the capacity of each slot and the demand itself. Assuming that the total slot capacity and demand are polynomially related (i.e., the number of slots need to fulfill the demand is not exponential), the running-time complexity is $O\left(|J|\frac{D}{C}\right)$, where

$$D = \sum_{j\in J} d_j \tag{2}$$

and

$$C = \sum_{j\in J} \sum_{\substack{m\in M,\\ t\in T:\\ q_{jmt}>0}} \eta_{mt} c_{mt}. \tag{3}$$

Note that $C$ is the total machine capacity that can serve the job demands. We assume that, in the worst case, each job must use the machine exclusively, increasing the total number of slots and horizons needed to fulfillment.

Note that Algorithm 2 is a straightforward construction method based on the permutation of the jobs and it is completely independent of the metaphor of BRKGA. Indeed, this method can be easily adapted to other general optimization frameworks, as shown in the next section.

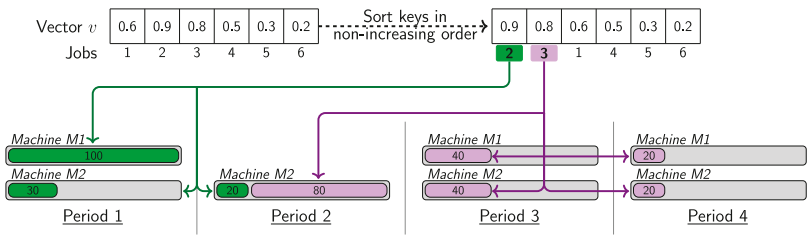



**Fig. 5.** Decoding process where jobs are sorted according the chromosome keys and then scheduled in sequence.

## 6. Solution using Iterated Local Search

The Iterated Local Search (ILS) is a procedure that consists of building a sequence of solutions by iteratively applying perturbations and local search methods from an initial solution. It consists of a perturbation phase, used to escape from local optima, and a local search phase used for the exploitation of the perturbed solution. ILS has been used with great success in several scheduling problems. Stützle and Ruiz [45] details this method and depicts a rich list of recent references.

For most scheduling problem, the local search explores the "interchange" and "insertion" neighborhoods [46]. In the interchange neighborhood, a neighbor is obtained by swapping two jobs from their original positions. The insert neighborhood moves a job to a new position trying to improve the solution cost. Note that the search in both neighbors is on the order of $O(n^2)$. When the search finds local optima in such neighborhoods, the perturbation step is applied, changing the job sequence randomly through an intensity parameter.

Algorithm 3 depicts the basic ILS procedure applied to the TMDSP. Note that all perturbations and searches are done in the vectors $v \in [0, 1]^n$ (where $n$ is the number of jobs), leveraging the Decoder 2 to build the solutions. The algorithm starts from a random vector (line 1) and extracts an initial solution from it (lines 2–3). The main loop (lines 4–16) starts with the perturbation step (lines 5–10), where two swaps between the vector keys are applied. The first move swaps two neighbors jobs chosen uniformly random (lines 6–7). In the second move, the algorithm swaps two uniformly chosen jobs (lines 8–10). Once perturbed, $v$ is decoded to extract a temporary solution (line 11). In lines 12–16, the algorithm applies the local searches in the insertion and interchange neighborhoods. This loop only stops when no improved solution can be found, and then, the best solution is returned. Note that function val(.) returns the total completion time of a given solution.

Note that both insertion and interchange neighborhood searches are done in $v$, i.e., the moves are applied to the keys in $v$ and then, $v$ is decoded to extract solution $\mathcal{S}$. However, to keep the correct semantic of the insertion and interchange neighborhoods, we do not exchange pairs of keys directly. Before that, the algorithm rebuilds the job sequence and applies either the insertion or the interchange move in the job sequence. After that, the corresponding keys of such jobs are exchanged in $v$. To clarify that, assume we have jobs $\{1, \ldots, 4\}$ and the corresponding $v = \{0.3, 0.1, 0.4, 0.2\}$. Therefore, $v$ induces the job sequence $(2, 4, 1, 3)$. A valid interchange move can switch the first pair $\langle 2, 4\rangle$ generating the sequence $(4, 2, 1, 3)$ and inducing $v = \{0.3, 0.2, 0.4, 0.1\}$. In other words, the moves are done in the solution space, not in the binary representation space.

**Algorithm 3:** Iterated Local Search for the TMDSP.

**Input**: Perturbation intensity $\rho$.
**Output**: Feasible schedule and the total completion time.

1 Let $v \in [0, 1]^n$ be a random vector where $n$ is the number of jobs;
2 $\mathcal{S} \leftarrow$ `decode`$(v)$;
3 $\mathcal{S}^\star \leftarrow \mathcal{S}$;
4 **while** *a stopping criterion is not reached* **do**
5  **for** $k \leftarrow 1$ **to** $\rho$ **do**
6  $i \leftarrow$ rand$(1, n-1)$;
7  swap$(v_i, v_{i+1})$;
8  $i \leftarrow$ rand$(1, n)$;
9  $j \leftarrow$ rand$(1, n)$;
10  swap$(v_i, v_j)$;
11  $\mathcal{S} \leftarrow$ `decode`$(v)$;
12  **do**
13  **if** val$(\mathcal{S}) <$ val$(\mathcal{S}^\star)$ **then**
14  $\mathcal{S}^\star \leftarrow \mathcal{S}$;
15  $\mathcal{S} \leftarrow$ insert_neighborhood$(\mathcal{S})$;
16  $\mathcal{S} \leftarrow$ interchange_neighborhood$(\mathcal{S})$;
17  **while** val$(\mathcal{S}) <$ val$(\mathcal{S}^\star)$;
18 **return** $\mathcal{S}^\star$;

## 7. Solution using traditional algorithms

Since BRKGA and ILS are relatively new approaches to solve optimization problems, it is important to compare those approaches with traditional optimization heuristics. For that, we also proposed a tabu search, a simulated annealing, and a traditional genetic algorithm for the TMDSP.

### 7.1. Simulating annealing

The simulating annealing is a search methodology that, starting from a given solution, explores its neighborhood and it may accept as the incumbent, solutions worse than the current given a probability. Such a probability is reduced over time, mimicking the temperature cool down from the metallurgy industry. Simulating annealing algorithms have been used for several scheduling problems such as [47,48] and more recently [49]. A survey of other application can be found in [50].

Traditional simulating annealing algorithms only explore one neighborhood of a solution. Since in the TMDSP it is too computationally expensive to explore a full neighborhood, we chose to explore partially two neighborhoods, to increase the probability to find good solutions. Algorithm 4 shows such a procedure. The algorithm starts with a random solution $\mathcal{S}^\star$ on line 1. The temporary solution $\mathcal{S}$, the temperature $t$, and the iteration and reset counters *iter* and *lri* are also initialized (lines 2–4). Then, the algorithm enters in the main loop (lines 5–27) iterating until a stopping criterion is reached. In the first neighborhood, we explore solutions that interchange neighboring jobs. The algorithm uniformly draws position $i$ in the interval $[1, n-1]$ (line 6), where $n$ is the number of jobs, and swaps jobs $\mathcal{S}_i$ and $\mathcal{S}_{i+1}$ (line 7). In line 8, the acceptance probability $p$ is computed using the

traditional Napierian exponential as a function of the difference between the value of the best solution found so far and the current solution, and the current temperature $t$. As before, function val(.) returns the total completion time of the given schedule. The current solution $\mathcal{S}$ is accepted either when it is better than the best solution found so far $\mathcal{S}^\star$ or the acceptance probability $p$ is greater than a random toss in the interval [0, 1] (line 9). *lb* holds the value of such a solution. Otherwise, the swap move is undone, and *lb* will hold the value of the best solution. Lines 13–19 depicts the same procedure as before, but for the second neighborhood where we swap two random jobs. In line 20, we hold the best solution. Lines 21–22 adjust the temperature. Note that, when the temperature is around the minimum one, the algorithm only accepts moves that improve the best solution. It is very likely that the algorithm converges fast, and when that happens, we restart/reset the search from a random solution. Such restart is controlled by the reset counter *lri*, and the parameter *rt*. Finally, the algorithm returns the best solution on line 28.

**Algorithm 4:** Simulating annealing for the TMDSP.

**Input**: Initial temperature $st$; attenuation factor $\alpha$; restart/reset offset $rt$.
**Output**: Feasible schedule.

1 Let $\mathcal{S}^\star$ be an initial random solution;
2 $\mathcal{S} \leftarrow \mathcal{S}^\star$;
3 $t \leftarrow st$;
4 $iter \leftarrow 1$; $lri \leftarrow 1$;
5 **while** *a stopping criterion is not reached* **do**
  // 1st neighborhood.
6  $i \leftarrow \text{rand}(1, n-1)$;
7  swap($\mathcal{S}_i, \mathcal{S}_{i+1}$);
8  $p \leftarrow e^{-\frac{\text{val}(\mathcal{S})-\text{val}(\mathcal{S}^\star)}{t}}$;
9  **if** val($\mathcal{S}$) < val($\mathcal{S}^\star$) **or** $p >$ rand(0, 1) **then**
10  $lb \leftarrow \text{val}(\mathcal{S})$;
11  **else**
12  swap($\mathcal{S}_i, \mathcal{S}_{i+1}$); $lb \leftarrow \text{val}(\mathcal{S}^\star)$;
  // 2nd neighborhood.
13  $i \leftarrow \text{rand}(1, n-1)$; $j \leftarrow \text{rand}(1, n-1)$;
14  swap($\mathcal{S}_i, \mathcal{S}_j$);
15  $p \leftarrow e^{-\frac{\text{val}(\mathcal{S})-\text{val}(\mathcal{S}^\star)}{t}}$;
16  **if** val($\mathcal{S}$) < val($\mathcal{S}^\star$) **or** $p >$ rand(0, 1) **then**
17  $lb \leftarrow \text{val}(\mathcal{S})$;
18  **else**
19  swap($\mathcal{S}_i, \mathcal{S}_j$); $lb \leftarrow \text{val}(\mathcal{S}^\star)$;
20  **if** $lb <$ val($\mathcal{S}^\star$) **then** $\mathcal{S}^\star \leftarrow \mathcal{S}$ ;
21  $t \leftarrow \alpha t$;
22  **if** $t < 1$ **then** $t \leftarrow 1$ ;
  // Restart/Reset.
23  **if** $t = 1$ **and** $iter - lri > rt$ **then**
24  Let $\mathcal{S}$ be an random solution;
25  $t \leftarrow st$;
26  $lri \leftarrow iter$;
27  $iter \leftarrow iter + 1$;
28 **return** $\mathcal{S}^\star$;

### 7.2. Tabu search

Proposed by Glover [51], the tabu search keeps a record of visited solutions called tabu list or table. This list is used to skip solutions already visited in the past, allowing new solution (maybe worse ones) to be accepted as the incumbent, and therefore, have its neighborhood visited. Tabu search has been successful to solve several problems and a comprehensive survey can be found in [52]. More recent work of taby search on scheduling problems can be found in [53,54].

Traditionally, tabu search explores the full neighborhood of a solution to then, move to the best neighbor that is not in the tabu table. This can be too computationally costly to TMDSP real-large instances. Therefore, we tested two approaches: in the first one, we sweep the full neighborhood as in the traditional tabu search; in the second approach, we explore the neighborhood partially through a parameter. As in the other algorithms discussed in this paper, we use the "interchange" neighborhood (discussed in Section 6).

Algorithm 5 brings a partial neighborhood tabu search for the TMDSP. The algorithm starts from a random solution $\mathcal{S}^\star$, and initialize the candidate solution $\mathcal{S}^{cand}$, an empty tabu list, and the iteration and last reset counters *iter* and *lri* (lines 1–4). Then, we reach the main loop (lines 5–21) that stops when a stopping criterion is reached. Lines 6–12 explores *maxneigh* random neighbors in the interchange neighborhood by drawing to random jobs $i$ and $j$ (line 8), swap their position (line 9), and accepting the new solution when it is better than the candidate solution, and it is not in the tabu list (lines 10–11). Note that line 12 restores the original solution. Once this search is done, the algorithm adds the candidate solution $\mathcal{S}^{cand}$ to the tabu list (line 13) and holds the best solution (line 14). The tabu tenure is based on the maximum size *mt* of the tabu list: if the tabu list is larger than *mt*, we remove the oldest solution in that list. Following, if the algorithm reaches a local minimum and stays there for *rt* iterations, the algorithm cleans the tabu list and restart/reset the search from a new solution. The algorithm ends retuning the best solution found on line 22.

Note that Algorithm 5 explores a limited space in the neighborhood of each solution. To fully explore the neighborhood, one must replace lines 7 and 8 by a (nested) loop that iterates over all pairs of jobs. As commented before, the computational complexity of such a search is $O(n^2)$ where $n$ is the number of jobs.

### 7.3. Traditional genetic algorithm

Since the focus of this paper is the TMDSP and its solution through BRKGA algorithms, we also have implemented a traditional genetic algorithm [55] for comparison. On each generation, we re-create the whole population contrasting the elitism of BRKGA that always copy the top performing chromosomes across generations.

The new individuals are created from a single point crossovers: a crossover point is uniformly chosen such that the chromosome parents A and B are sectioned in two parts, e.g., AA′ and BB′. Then, two children are generated by exchanging those parts between the parents, e.g., AB′ and BA′. We leverage the random-key mechanism of BRKGA here, which guarantee that any crossover point is chosen, we always have a valid solution.

The parent's selection uses the roulette method. The algorithm assigns to each parent a probability based on their fitness, such that we create a cumulative probability distribution used during the parent selection. Therefore, parents with better fitness (small total completion times) have more probability to be picked than less fit individuals.

After generating the new individuals, the algorithm applies a mutation with a small probability on each gene of each chromosome. Such mutations "invert" the value allele/key. For instance, $v_i = 0.3$ is chosen for mutation, the new value will be $v_i =$

**Algorithm 5:** Tabu search for the TMDSP.

**Input**: Maximum tabu list size $mt$; Maximum number of neighbors to be visited $maxneigh$; restart/reset offset $rt$.
**Output**: Feasible schedule.

Let $\mathcal{S}^\star$ be an initial random solution;
$\mathcal{S}^{cand} \leftarrow \mathcal{S}^\star$;
$tabulist \leftarrow \emptyset$;
$iter \leftarrow 1$; $lri \leftarrow 1$;
**while** *a stopping criterion is not reached* **do**
  $\mathcal{S}^{temp} \leftarrow \mathcal{S}^{cand}$;
  **for** $k \leftarrow 1$ **to** $maxneigh$ **do**
    $i \leftarrow \text{rand}(1, n-1)$; $j \leftarrow \text{rand}(1, n-1)$;
    swap($\mathcal{S}_i^{temp}, \mathcal{S}_j^{temp}$);
    **if** val($\mathcal{S}^{temp}$) < val($\mathcal{S}^{cand}$) **and** $\mathcal{S}^{temp} \notin tabulist$ **then**
      $\mathcal{S}^{cand} \leftarrow \mathcal{S}^{temp}$;
    swap($\mathcal{S}_i^{temp}, \mathcal{S}_j^{temp}$); // Restore previous.
  $tabulist \leftarrow tabulist \cup \{\mathcal{S}^{cand}\}$;
  **if** val($\mathcal{S}^{cand}$) < val($\mathcal{S}^\star$) **then** $\mathcal{S}^\star \leftarrow \mathcal{S}^{cand}$ ;
  **if** $|tabulist| > mt$ **then**
    Remove the oldest solution from $tabulist$;
  // Restart/Reset.
  **if** $iter - lri > rt$ **then**
    Let $\mathcal{S}^{cand}$ be an random solution;
    $tabulist \leftarrow \{\mathcal{S}^{cand}\}$;
    $lri \leftarrow iter$;
  $iter \leftarrow iter + 1$;
**return** $\mathcal{S}^\star$;

0.7. Again, such mutations do not generate invalid solutions but change the position of a job in the solution.

As for BRKGA, we have also tested the island model [56] where the algorithm evolves multiples populations in parallel and, once a while, exchanges a given number of individuals among the population.

The parameters for the traditional genetic algorithm are: population size $p$ and mutation probability $\mu$. And, as we did for BRKGA, we also use during tuning, parameters to control the number of populations, migration between populations, and reset/restart when trapped on local minima. We use the same stopping criteria we use for the other algorithms.

## 8. Experimental results and discussion

### 8.1. Instances

We tested two sets of instances. The first set contains 60 synthetic instances named SY. These instances are small, and they are intended to assert the MIP model and algorithms correctness. They are divided into two subsets. In the Loose subset, the machines are available 80% of the time, on average, their capacity varies from 30 to 80, and we have no global capacity. In the Tight subset, the machines' capacities are tighter (30 to 40), they have 30% of chance to appear in a period, and the global capacity is set to 100. The jobs distributions over time use a bi-normal distribution to mimic both rush hours in a regular work day. All other parameters and distributions are described in Appendix A. The instance naming follows the pattern sy_<subset>_j<N>_m<N>_p<N>_<N> where <subset> is either loose or tight, and j, m, and p are followed by the number of jobs, machines, and periods, respectively. Lastly, a suffix identifies different instances generated from the same set of parameters. For example, sy_loose_j20_m05_p010_01 is an instance with 20 jobs on 5 machines with high availability during 10 periods.

**Table 2**
Basic statistics of FU instances.

| Subset | Inst. | Cars | | | Sectors | | |
|---|---|---|---|---|---|---|---|
| | | Min. | Max. | Median | Min. | Max. | Median |
| Common | 71 | 505 | 33,132 | 5514 | 1463 | 21,913 | 5167 |
| Rare | 82 | 107 | 16,844 | 1456 | 1262 | 24,956 | 5030 |

The second instance set is compounded by 153 instances based on a real-world Internet of Things application of firmware updates over-the-air for connected cars [1,10], and we name them FU instances. As in the previous sections, we consider that a car is a job, and an LTE sector[1] is a machine, and we use for these instances the former nomenclature. The set is divided into two subsets: in the first that contains 71 instances, the cars appear very frequently in several sectors during the schedule window. We call this subset Common. The second subset contains 82 instances with a considerable number of rare cars, i.e., cars that appear in a few sectors and time slots. This subset is called Rare. Table 2 shows some basic statistics about FU instances. The first column describes the subset and the second column the number of instances in that set. Following we have the minimum, maximum, and median number of cars and sectors, respectively. For detailed information, see Appendix A. Note that we have a fixed number of 672 periods for all these instances (15-minutes bins for a week), and they are not shown in the table. The nomenclature follows the same pattern of SY instances, but the suffix is dropped, since the instances have unique distributions. Both sets of instances are available at https://github.com/ceandrade/tmdsp_instances.[2]

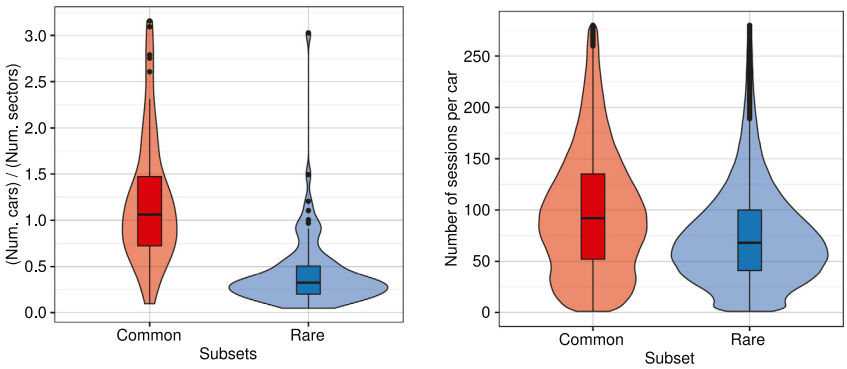
While the number of sectors is relatively close between the two subsets, the number of cars is much larger in Common subset than in the Rare subset on average. Indeed, Rare instances have relatively fewer cars than sectors, as shown in Fig. 6(a). The car-per-sector ratios for Rare are $0.43 \pm 0.40$. Common instances have the opposite characteristic: $1.47 \pm 1.14$ cars per sector on average. Fig. 6(b) also shows that the number of sessions per car is significantly larger in the Common subset that the Rare subset. On average, cars from the Common instances have $115 \pm 96$ sessions (max = 3432) and cars from Rare instances have $83 \pm 69$ sessions (max = 1984). Such difference is significant at a 95% confidence level. Therefore, cars from Common instances should have higher probability to be scheduled than cars from Rare instances.

### 8.2. Computational environment and parameter settings

The computational experiments are performed on a cluster of identical machines with a Intel Xeon E5530 CPU at 2.40 GHz (4 cores/8 threads) and 32 GB of RAM running CentOS Linux. The algorithms are implemented in C++ and built using the GNU GCC 7.2. The BRKGA code is a variation from [57]. The MIP formulation was solved with IBM ILOG CPLEX 12.8 solver. Running times reported are UNIX real wall-clock times in seconds, excluding the effort to read the instance.

[1] A sector is a set of several radio cells or frequency carriers covering the same direction from the base station. A car connects to one of the cells in the same sector, and sector capacity is the sum of cell capacities.

[2] The instances are in the release process at this time.




(a) Ratio between number of cars and sectors.

(b) Car sessions.

**Fig. 6.** Instances' distributions.

We use the traditional acronyms for BRKGA, ILS, and CPLEX. The simulating annealing results are under acronym SA. We use TS-FULL and TS-RD for the full and partial neighborhood searches in the tabu search algorithms, respectively. The traditional genetic algorithm is referenced as TGA.

For all algorithms except ILS and CPLEX, due to the large number of control parameters, we performed parameter tuning using the irace package [58]. For details, refer to Appendix B.

Assuming $n = |\mathcal{J}|$ as the number of jobs, the following parameter values were suggested by irace:

- BRKGA: population size of $p = 180$; elite size $p_e = \lfloor 0.26 \times 180 \rfloor = 46$; number of mutants introduced at each generation $p_m = \lfloor 0.10 \times 180 \rfloor = 18$; probability of biased crossover $\rho = 0.58$; one population; reset interval of 500 iterations without improvements in the best solution;
- SA: start temperature $st = 450$; attenuation factor $\alpha = 0.95$; reset interval $rt = 4900$;
- TS-FULL: maximum tabu list size $mt = n$; reset interval $rt = 170$;
- TS-RD: maximum tabu list size $mt = 0.75n$; reset interval $rt = 450$; maximum number of neighbors to be visited $maxneigh = 0.015n$;
- TGA: population size of $p = 60$; mutation probability $\mu = 0.027$; one population; reset interval of 500 iterations without improvements in the best solution.

Since ILS only has the perturbation parameter to be tuned, we performed a simple design of experiments with 20% of the FU instances chosen randomly, and 10 independent runs for each. The perturbation parameter was varied from 2 to 10. Setting the perturbation parameter to 4 produced better results on average.

For CPLEX, we used the default parameter setting but limited the working memory to 30 GB. Since MIP (1) requires a valid makespan upper bound, we run BRKGA for 10% of the allowed time, and use its solution to bound such makespan. This BRKGA solution is also used to warm-start CPLEX as an initial incumbent solution.

For all algorithms, we allowed four parallel threads, and for the stopping criteria, we use one wall-clock hour for SY instances and six wall-clock hours for FU instances. Except for CPLEX, we used an additional stopping criteria: the optimization is ended after 1000 iterations without improvement in the best solution found so far during that run. For each instance, 30 independent runs (using different seeds) were performed per algorithm, but only one run for CPLEX.

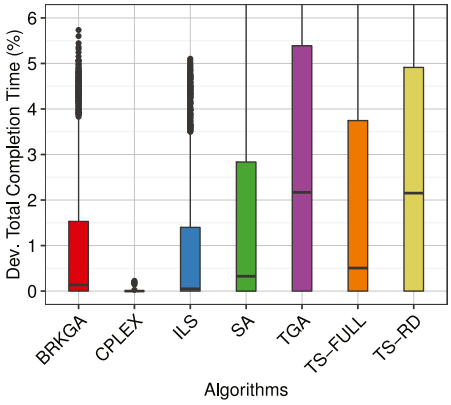



**Fig. 7.** Distributions of the proportional deviations for the total completion time.

### 8.3. Analysis of the synthetic instances

First, we analyze the synthetic instances, since they are smaller and carefully crafted, allowing the assessment of the correctness of our approaches. For that, we defined proportional difference or deviation as $(\mathcal{A}(\mathcal{I}) - B_{\mathcal{I}})/B_{\mathcal{I}}$ where, for a given instance $\mathcal{I}$, $B_{\mathcal{I}}$ is the value of the best solution known for $\mathcal{I}$, and $\mathcal{A}(\mathcal{I})$ is the value of a solution obtained by algorithm $\mathcal{A}$ for $\mathcal{I}$.

Fig. 7 depicts the average deviation from the best results for the synthetic instances. We can note that CPLEX presented solid results, primarily concentrated close to zero. The average deviation for CPLEX was only $0.01 \pm 0.05\%$. Then we have ILS and BRKGA with averages of $0.88 \pm 1.36\%$ and $0.92 \pm 1.38\%$, respectively. The other algorithms presented higher deviations with also more jitter: SA with $1.55 \pm 2.17\%$, TGA with $2.99 \pm 3.22\%$, TS-FULL with $19.3 \pm 2.55\%$, and TS-RD with $2.72 \pm 2.96\%$.

To confirm these results, we tested the algorithms' distributions for normality using the Shapiro test, which revealed that these are not normally distributed within 95% confidence interval. Therefore, we applied pairwise comparisons using Wilcoxon rank sum test with Hommel's $p$-value adjusting which results are shown on Table 3. The structure of this table is as follows. Each row and column is indexed by one algorithm. Each element in the diagonal (bold) is the median of the percentage deviation for the corresponding algorithm. The upper-right diagonal elements are medians of the differences in location statistics for each pair of algorithms. Larger negative differences indicates that the "row algorithm" has its location statistics *better* than the "column

**Table 3**
Difference in median location for cost distributions for synthetic instances. The main diagonal shows the median of the results of each algorithm. The upper-right elements are the difference between the medians. The bottom-left elements are the *p*-values. We use a confidence interval of 95% and omit *p*-values are less than 0.05.

| | BRKGA | CPLEX | ILS | SA | TGA | TS-FULL | TS-RD |
|---|---|---|---|---|---|---|---|
| BRKGA | **0.14** | 0.13 | 0.00 | 0.00 | -0.88 | -0.12 | -0.64 |
| CPLEX | | **0.00** | -0.03 | -0.31 | -2.16 | -0.47 | -2.13 |
| ILS | | | **0.05** | -0.02 | -0.90 | -0.15 | -0.66 |
| SA | | | | **0.33** | -0.46 | 0.00 | -0.26 |
| TGA | | | | | **2.17** | 0.20 | 0.00 |
| TS-FULL | | | | | | **0.51** | -0.01 |
| TS-RD | | | | | | | **3.15** |

**Table 4**
Algorithm performance on 50 synthetic instances with unknown optimal solutions.

| Algorithm | # Best | % Best | % Run | Prop. diff. | |
|---|---|---|---|---|---|
| | | | | % | $\sigma$ |
| BRKGA | 19.00 | 38.00 | 26.60 | 1.50 | 1.49 |
| CPLEX | 43.00 | 86.00 | 86.00 | 0.16 | 0.07 |
| ILS | 22.00 | 44.00 | 36.00 | 1.65 | 1.48 |
| SA | 17.00 | 34.00 | 21.80 | 2.38 | 2.29 |
| TGA | 10.00 | 20.00 | 15.53 | 4.25 | 3.07 |
| TS-FULL | 11.00 | 22.00 | 20.00 | 2.90 | 2.64 |
| TS-RD | 10.00 | 20.00 | 19.93 | 4.08 | 2.76 |

algorithm". The bottom-left diagonal elements are the *p*-values of each test. We omit all $p < 0.05$ values, that indicate that the difference is statistically significant for those pairs. We also omitted confidence intervals since for all tests the values lie in these intervals and they are very narrow. Within a confidence interval of 95%, CPLEX delivered significantly better results than the other algorithms. ILS appears in second place, and even though its median is very close to BRKGA and SA ones (less than 0.01), its results are yet statistically better. The same situation occurs when we compare BRKGA and SA that are ranked in third and fourth, respectively. TS-FULL follows overcoming TS-RD and TGA. It is interesting to note that TS-FULL got significantly better results than TS-RD, probably because it performs a whole neighborhood search. In another hand, SA also performs a partial neighborhood search as TS-RD does, but the former can obtain better results than the latter.

CPLEX found an optimal solution for 10 from the 60 synthetic instances. Indeed, all other algorithms also found, in at least one run, an optimal solution for each of such cases. For the remaining 50 instances, Table 4 describes the summary of the results. The first column brings the name of the algorithm. Column "# Best" represents the number of instances for which the algorithm found the best solution known; column "% Best" shows a percentage of the number of best solutions found; and column "% Run" shows a percentage of the number of runs on which the algorithm found the best solution. The two columns under label "Prop. diff." show, respectively, the average of the proportional difference between the best solution value and the achieved value (%), and its corresponding standard deviation ($\sigma$). Not surprisingly, CPLEX found a large number of best solutions, presenting minimal deviations for the ones it could not reach. BRKGA and ILS presented similar results, and even they fell short in the number of best solutions found, their deviations are less than 2% from the best ones. SA also presented similar results, although its deviation is higher than the previous algorithms. Tabu search variations and TGA fell short presenting no more than half of the best solutions found by CPLEX. Per-instance results are shown in Appendix C.

Fig. 8 shows performance profiles for all algorithms where *X*-axis shows the time needed to reach a target solution value while the *Y*-axis shows the cumulative probability to reach a target solution value for the given time in the *X*-axis. In Fig. 8(a), the target are the values of the median solution for each instance. For Fig. 8(b), the values of the best solutions for each instance are used. Note that for the average cases, BRKGA is really fast to find such solutions, presenting higher empirical probability than all other algorithms, in the first five seconds. ILS also presented a good empirical probability that grows linearly from 60% to 100% after 10 seconds. CPLEX uses a lot of time polishing the solutions which appear at the end of the optimization. For the best solutions, BRKGA and ILS have similar behavior, although ILS is able to find more best solutions as described earlier in Table 4. SA, TS-FULL, and TS-RD have similar results for the best solution, but SA is much faster on find average solutions. TGA fell short in both situations due to premature convergence, even though with restart strategy enabled.

### 8.4. Analysis of the real-world instances

Since our main interest is to solve real-world instances, we turn to the analysis of the firmware-over-the-air updates for connected cars, previously named as FU instances. It turns out that such instances are much harder to solve than the synthetic ones, not only due to their size, but also due to the heterogeneity among the cars, sectors, and loads distributions. Recall that we use six wall-clock hours for each experiment in this section. From the 153 FU instances, CPLEX was able to solve only one instance from the Common subset and 29 instances from the Rare subset (totaling only 19% of the FU instances). Although CPLEX found feasible solutions for 30 instances, only for fu_rare_j00218_m03201_p672, CPLEX found a better solution than the incumbent (but with a small marginal difference of 0.01%). Therefore, we focus our attention to the proposed heuristics further.

Fig. 9(a) depicts the distribution of the percentage deviation from the best solutions for each algorithm for the Common subset. BRKGA presented minimal variation with an average deviation of $3.46 \pm 2.63\%$. The other algorithms show more jitter in their results and significantly higher average deviations: $7.34 \pm 4.26\%$ for TGA; $7.67 \pm 3.54\%$ for SA; $8.09 \pm 4.21\%$ for TS-RD; $8.75 \pm 4.89\%$ for TS-FULL; and $9.39 \pm 5.07\%$ for ILS. It is interesting to note that the distributions of all algorithms presented a guitar-shape form, with smaller density around the median for all algorithms but BRKGA. This fact indicates that we have several good and bad solutions, but not much of then at the medium point. BRKGA is an exception, with large density at the median, and the first quartile with small deviations.

Fig. 9(b) shows the distribution of the percentage deviation of makespans. These values are the makespans obtained from solutions for which the total completion time is minimized, as defined in the problem. Note the solutions obtained for both algorithms have similar distributions, and they do not differ statistically (*p*-value > 0.05, confidence interval of 95%). The average makespan deviation for BRKGA is $1.48 \pm 7.81\%$, for ILS is $2.03 \pm 10.27\%$, for SA is $1.87 \pm 10.28\%$, for TGA is $2.11 \pm 9.90\%$, for TS-FULL is $2.23 \pm 10.60\%$, and for TS-RD is $2.25 \pm 10.17\%$. It is interesting to note that the minimum and the first quartile for all algorithms are negative numbers. These results suggest that the correlation between total completion time and makespan in the TMDSP may be weak. Therefore, we may have a solution with a very good makespan but substantial total completion time, which may indicate an imbalance in the job distribution along the schedule. The average percentage deviation of total completion times between the best solution (with respect to the total completion time) and

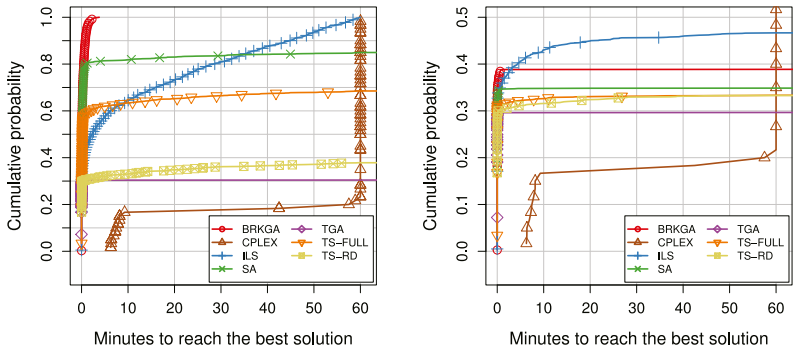



**Fig. 8.** Running time distributions to the best solutions found. The markers correspond to 5% of the points plotted for each algorithm.

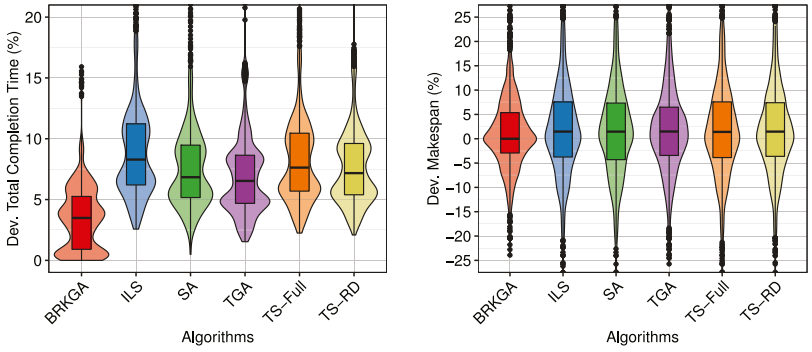



**Fig. 9.** Distributions for the FU Common subset.

the solution with the smallest makespan is: 3.46 ± 2.07% with minimum 0.00% for BRKGA; 9.42 ± 5.26% with minimum 2.80% for ILS; 7.75 ± 3.69% with minimum 2.44% for SA; 7.30 ± 4.25% with minimum 1.99% for TGA; 8.71±4.71% with minimum 2.44% for TS-FULL; and 8.06 ± 4.20% with minimum 2.63% for TS-RD.

Table 5 shows the results for Wilcoxon U test between all pairs of algorithms. Its description is similar to Table 4. One can note that all tests result in p-values less than 0.05, indicating that all algorithms differ among themselves. Differing from results for synthetic instances, BRKGA produced the best results followed by TGA, SA, TS-RD, and TS-FULL. Note that when ILS presented the best results for small instances, it cannot solve the real large instances as well as other algorithms. Also, the populational algorithms (BRKGA and TGA) were significantly better than the other strategies.

The results for Rare instances have similar shapes to those for the Common subset, as we see in Fig. 10(a), although the deviations are slightly higher than for the Common subset. On average, BRKGA deviation is 4.96±4.05%. When compared to the results from Fig. 9(a), BRKGA presents significantly more variation on Rare than Common instances ($p$-value < 0.05, Wilcoxon rank test, 95% confidence interval). The same occurs for the other algorithms such averages are: 16.39±4.63% for SA; 17.17±4.97% for TGA; 18.33±4.87% for TS-RD; 19.88±6.08% for TS-FULL; and 20.75 ± 6.44% for ILS. As before, BRKGA presented significantly better results than all other algorithms, at a 95% confidence level. However, while the difference is about 5% on the common subset, in the rare subset the difference is about 13%. Table 6 shows the results of the Wilcoxon U test for the Rare subset.

Fig. 10(b) depicts the distributions of makespan deviations for the Rare set. The average percentage deviation between the total

**Table 5**
Difference in median location for cost distributions for FU Common instance subset. The main diagonal shows the median of the results of each algorithm. The upper-right elements are the difference between the medians. The bottom-left elements are the $p$-values. We use a confidence interval of 95% and omit $p$-values are less than 0.05.

| | BRKGA | ILS | SA | TGA | TS-FULL | TS-RD |
|---|---|---|---|---|---|---|
| BRKGA | **3.49** | -5.30 | -4.01 | -3.44 | -4.69 | -4.22 |
| ILS | | **8.29** | 1.27 | 1.84 | 0.60 | 1.06 |
| SA | | | **6.84** | 0.58 | -0.67 | -0.21 |
| TGA | | | | **6.53** | -1.23 | -0.79 |
| TS-FULL | | | | | **7.63** | 0.67 |
| TS-RD | | | | | | **7.17** |

**Table 6**
Difference in median location for cost distributions for FU Rare instance subset. The main diagonal shows the median of the results of each algorithm. The upper-right elements are the difference between the medians. The bottom-left elements are the $p$-values. We use a confidence interval of 95% and omit $p$-values are less than 0.05.

| | BRKGA | ILS | SA | TGA | TS-FULL | TS-RD |
|---|---|---|---|---|---|---|
| BRKGA | **4.10** | -16.30 | -11.66 | -12.44 | -15.36 | -13.58 |
| ILS | | **21.95** | 4.77 | 3.89 | 0.93 | 2.78 |
| SA | | | **16.92** | -0.89 | -3.85 | -1.98 |
| TGA | | | | **18.11** | -2.96 | -1.11 |
| TS-FULL | | | | | **20.95** | 2.13 |
| TS-RD | | | | | | **19.04** |

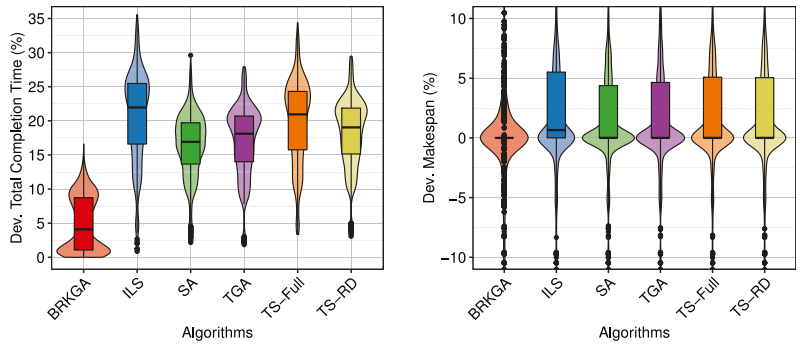



**Fig. 10.** Distributions for the FU Rare subset.

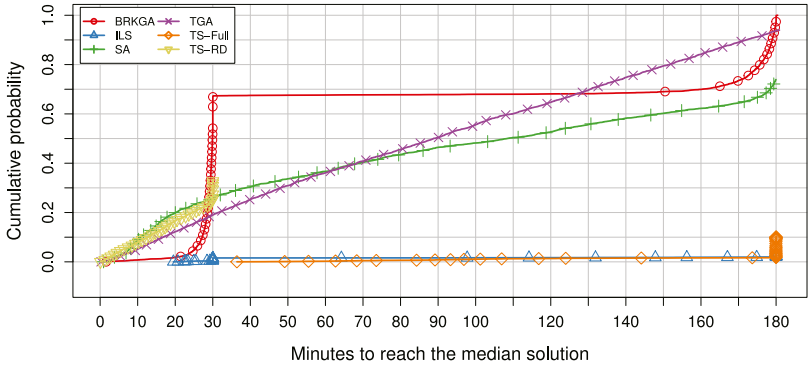



**Fig. 11.** Running time distributions to the median solutions. The markers correspond to 1% of the points plotted for each algorithm.

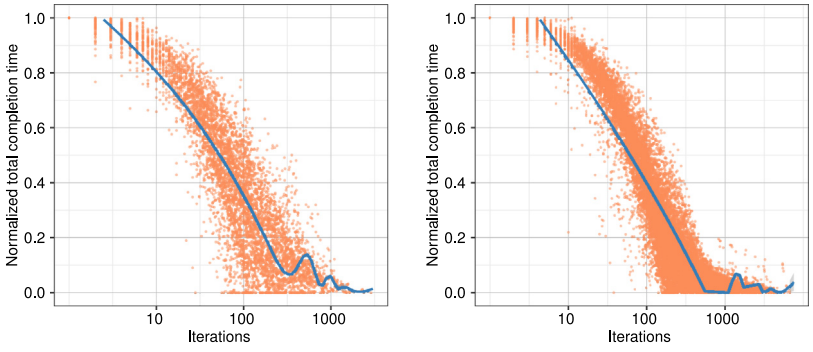



**Fig. 12.** Evolution of the total completion time over BRKGA iterations (5% of the total solutions are shown).

completion times from the best solution (with respect to the total completion time) and the solution with the smallest makespan is: 3.57 ± 3.79% with minimum 0.00% for BRKGA; 20.55 ± 6.51% with minimum 1.26% for ILS; 16.42±4.41% with minimum 3.98% for SA; 17.25 ± 5.00% with minimum 2.52% for TGA; 19.74 ± 5.94% with minimum 4.55% for TS-FULL; and 20.14±5.98% with minimum 4.57% for TS-RD.

As commented before, FU instances are hard to solve. BRKGA was the only algorithm to reach the best solution for each instance, and it could do it in only 3% of the runs. No other tested algorithm found the best solutions. For this reason, Fig. 11 shows the performance profiles with targets set to the values of the median solution for each instance. We also tried to plot against the 1st quartile values of such solutions. However, neither algorithms found solutions better then the 1st quartile solutions. One can note that BRKGA (red lines with circles) presented a high probability to find a median solution in the first 30 minutes of optimization, but stalls 150 min (2.5 h) when it momentum starts again. TGA (purple lines with crosses) has a linear behavior and outpasses BRKGA with 2 hours of optimization, although BRKGA surpasses it in the long run. SA (green lines with plus signals) also presented a linear behavior presenting better cumulative probability than TGA in the first 60 minutes, although its probability is smaller than BRKGA and TGA in the long run. TS-RD (golden

lines with upside-down triangles) is competitive until 30 minutes but cannot find good solutions after that. Lastly, ILS (blue line with upside triangles) and TS−FULL (orange line with diamonds) presented very low probability to find even a median solution, although TS−FULL is a little bit better for long runs. Per-instance results are shown in Appendix C.

Although Fig. 11 suggests that BRKGA has "big jumps" in the search space, this observation is not accurate. Fig. 12 shows the BRKGA evolution process. For each instance and each run, we normalize the total completion time, making the initial solution proportional to 1.0 and the best solution proportional to 0.0 on the $Y$-axis. The $X$-axis depicts the number of iterations on the log scale. Note that the evolution process improves the solution exponentially until iteration 1000. From that point, improvements are rare and present slight variability. We can note the BRKGA converges faster in Rare instances than Common instances. Fig. 12 also suggests more dispersion on Common instances.

## 9. Conclusion

In this paper, we present a new scheduling problem called Time- and Machine-Dependent Scheduling Problem (TMDSP), including some variants. The main characteristic that differentiates TMDSP from other scheduling problems is the temporal relationship between each job and each machine. In each period, the capacity and production rate of each machine varies. The amount of capacity a job requires from a machine also varies with the machine and in time. TMDSP is used to model highly heterogeneous scenarios such as firmware-over-the-air downloads for connected cars and other Internet of Things devices. We also present a Mixed Integer Programming (MIP) formulation, and developed classical and modern metaheuristics to solve the TMDSP, among them simulating annealing, two versions of tabu search, traditional genetic algorithm, biased random-key genetic algorithm, and iterated local search. We tested these approaches on a collection of instances based on a real-world application of firmware updates over-the-air for devices with limited capacity.

TMDSP presents as a very challenging problem to solve due to the heterogeneity of the capacities and productions rates and also due to the large size of practical instances. A MIP commercial solver is able to solve only 21% of the instances from the proposed benchmark, even when allowed a large amount of memory and running time. BRKGA presents significantly better results when compared with the MIP solver and the other tested heuristics. BRKGA is able to find solutions within 20% of the best found using 100 evolution iterations and closes the gap to 5% within 1000 iterations.

TMDSP and its variations open a new research path where further investigations are warranted. It would be interesting to identify structural properties of this problem with an aim to design better approaches to solve it, such as new heuristics, lower bounds, valid cuts, and decomposition strategies. Other interesting analyses may be to understand the relationship between the total completion time and the makespan, by optimizing the latter instead the former metric, in both TMDSP-ST and TMDSP-MT variations.

## Acknowledgments

We thank the anonymous reviewers for providing comments that improved this paper, and AT&T IoT business unit, in special Barbara Roselle, for provide us valuable information and help in this project.

## Declaration of competing interest

One or more of the authors of this paper have disclosed potential or pertinent conflicts of interest, which may include receipt of payment, either direct or indirect, institutional support, or association with an entity in the biomedical field which may be perceived to have potential conflict of interest with this work. For full disclosure statements refer to https://doi.org/10.1016/j.asoc.2019.105575.

## Appendix A. Details for FU instances

Tables A.7 and A.8 show details about the Common and Rare instances, respectively. The first column shows the name of the instance, followed by the number of jobs, number of machines, and the ratio between jobs and machines. The last three columns depict the average demand, average number of sessions per job, and the average density. The density describes the average number of periods of consumption of a job over all possible slots. Let $\beta_j$ be the number of periods where job $j$ appears on a machine, i.e.,

$$\beta_j = |\{q_{jmt} > 0, \forall m \in M, t \in T\}|.$$

The density is computed using the equation

$$\left(\sum_{j \in J} \frac{\beta_j}{|J||M||T|}\right) / |J|. \tag{A.1}$$

The density is useful to analyze the number of non-zeros elements in the MIP model, and usually is correlated with the hardness in solve the MIP.

## Appendix B. Parameter settings

For tuning, we chose randomly a set of 20 instances which sizes vary from 1557 jobs and 5266 machines to 25,312 jobs and 8184 machines from both Rare and Common subsets.

Most of the tested algorithms have a larger number of control parameters which makes difficult to perform a full factorial design. Therefore, we used the *iterated racing* to tune the parameters. This method consists of sampling configurations from a particular distribution, evaluating them using a statistical method, and refining the sampling distribution with repeated applications of F-Race. We used the irace package [58], for such task. We used a budget of 3000 experiments in the tuning procedure over the instances mentioned earlier in this section. Each experiment was limited to three hours.

To tune the BRKGA parameters, we used the following ranges: population size $p \in [50, 700]$; percentage of elite individuals $p_e \in [0.1, 0.5]$; percentage of mutants introduced at each generation $p_m \in [0.1, 0.5]$; probability on biased crossover $\rho \in [0.5, 0.8]$; number of populations in $[1, 2]$; interval for individual exchange in $[50, 100]$ iterations; number of elite individuals exchanged in $[1, 2]$; and number of iterations without improvement in the best solution until population reset in $[200, 500]$.

Table B.9 shows the suggestion made by irace The first four columns represent the parameters as described before, column "# pop." stands for number of populations, column "Int. exch." stands for interval for individual exchange, column "# ind. exch." represents number of exchanged individuals, and column "Reset Int." represents the reset interval. We picked the values of the

**Table A.7**
Details of the 71 Common instances. The number of jobs and machines are absolute numbers. Demands, number of sessions per job, and density are averages.

| Instance name | Jobs | Machines | Ratio | Average | | |
|---|---|---|---|---|---|---|
| | | | | Demand | Sessions | Density |
| fu_common_j00505_m01902_p672 | 505 | 1902 | 0.27 | 3261.78 | 125.14 | 2e-07 |
| fu_common_j00586_m01497_p672 | 586 | 1497 | 0.39 | 3283.00 | 103.48 | 2e-07 |
| fu_common_j00888_m03941_p672 | 888 | 3941 | 0.23 | 3292.25 | 81.09 | 3e-08 |
| fu_common_j01562_m02572_p672 | 1562 | 2572 | 0.61 | 3302.23 | 112.26 | 4e-08 |
| fu_common_j01954_m02112_p672 | 1954 | 2112 | 0.93 | 3361.47 | 105.48 | 4e-08 |
| fu_common_j02102_m21913_p672 | 2102 | 21913 | 0.10 | 3391.06 | 118.57 | 4e-09 |
| fu_common_j02225_m02480_p672 | 2225 | 2480 | 0.90 | 3382.72 | 128.50 | 3e-08 |
| fu_common_j02304_m02242_p672 | 2304 | 2242 | 1.03 | 3364.72 | 87.33 | 3e-08 |
| fu_common_j02381_m02148_p672 | 2381 | 2148 | 1.11 | 3341.99 | 102.16 | 3e-08 |
| fu_common_j02441_m02307_p672 | 2441 | 2307 | 1.06 | 3357.12 | 103.86 | 3e-08 |
| fu_common_j02657_m04023_p672 | 2657 | 4023 | 0.66 | 3324.11 | 117.34 | 2e-08 |
| fu_common_j02754_m04811_p672 | 2754 | 4811 | 0.57 | 3294.35 | 101.30 | 1e-08 |
| fu_common_j02912_m02817_p672 | 2912 | 2817 | 1.03 | 3301.65 | 128.63 | 2e-08 |
| fu_common_j03211_m03220_p672 | 3211 | 3220 | 1.00 | 3362.09 | 113.94 | 2e-08 |
| fu_common_j03267_m04576_p672 | 3267 | 4576 | 0.71 | 3320.82 | 123.01 | 1e-08 |
| fu_common_j03298_m01645_p672 | 3298 | 1645 | 2.00 | 3381.35 | 107.42 | 3e-08 |
| fu_common_j03310_m02915_p672 | 3310 | 2915 | 1.14 | 3408.15 | 108.95 | 2e-08 |
| fu_common_j03320_m04583_p672 | 3320 | 4583 | 0.72 | 3340.82 | 124.85 | 1e-08 |
| fu_common_j03426_m04741_p672 | 3426 | 4741 | 0.72 | 3311.15 | 103.63 | 9e-09 |
| fu_common_j03593_m02974_p672 | 3593 | 2974 | 1.21 | 3295.79 | 109.79 | 2e-08 |
| fu_common_j03696_m04574_p672 | 3696 | 4574 | 0.81 | 3343.72 | 105.77 | 9e-09 |
| fu_common_j03716_m02145_p672 | 3716 | 2145 | 1.73 | 3366.20 | 81.74 | 2e-08 |
| fu_common_j03734_m05485_p672 | 3734 | 5485 | 0.68 | 3333.69 | 102.18 | 7e-09 |
| fu_common_j03873_m05287_p672 | 3873 | 5287 | 0.73 | 3305.22 | 114.80 | 8e-09 |
| fu_common_j03897_m02588_p672 | 3897 | 2588 | 1.51 | 3339.72 | 119.23 | 2e-08 |
| fu_common_j03918_m04835_p672 | 3918 | 4835 | 0.81 | 3335.66 | 116.47 | 9e-09 |
| fu_common_j03978_m10039_p672 | 3978 | 10039 | 0.40 | 3307.71 | 106.88 | 4e-09 |
| fu_common_j04143_m06535_p672 | 4143 | 6535 | 0.63 | 3323.16 | 129.28 | 7e-09 |
| fu_common_j04241_m03139_p672 | 4241 | 3139 | 1.35 | 3344.08 | 118.35 | 1e-08 |
| fu_common_j04412_m05650_p672 | 4412 | 5650 | 0.78 | 3315.47 | 111.59 | 7e-09 |
| fu_common_j04525_m06837_p672 | 4525 | 6837 | 0.66 | 3264.25 | 109.44 | 5e-09 |
| fu_common_j04574_m03788_p672 | 4574 | 3788 | 1.21 | 3362.03 | 110.20 | 9e-09 |
| fu_common_j04604_m04813_p672 | 4604 | 4813 | 0.96 | 3326.50 | 129.78 | 9e-09 |
| fu_common_j04888_m08869_p672 | 4888 | 8869 | 0.55 | 3285.70 | 113.43 | 4e-09 |
| fu_common_j04961_m06153_p672 | 4961 | 6153 | 0.81 | 3300.01 | 111.77 | 5e-09 |
| fu_common_j05514_m05088_p672 | 5514 | 5088 | 1.08 | 3354.31 | 112.42 | 6e-09 |
| fu_common_j05710_m10019_p672 | 5710 | 10019 | 0.57 | 3255.76 | 116.10 | 3e-09 |
| fu_common_j05783_m07209_p672 | 5783 | 7209 | 0.80 | 3310.14 | 110.15 | 4e-09 |
| fu_common_j05822_m01844_p672 | 5822 | 1844 | 3.16 | 3353.84 | 108.01 | 1e-08 |
| fu_common_j06167_m05038_p672 | 6167 | 5038 | 1.22 | 3347.96 | 116.28 | 6e-09 |
| fu_common_j06274_m05911_p672 | 6274 | 5911 | 1.06 | 3294.92 | 109.86 | 4e-09 |
| fu_common_j06407_m02296_p672 | 6407 | 2296 | 2.79 | 3265.35 | 118.34 | 1e-08 |
| fu_common_j06410_m04644_p672 | 6410 | 4644 | 1.38 | 3343.83 | 103.54 | 5e-09 |
| fu_common_j06809_m20225_p672 | 6809 | 20225 | 0.34 | 3314.88 | 130.46 | 1e-09 |
| fu_common_j06919_m05493_p672 | 6919 | 5493 | 1.26 | 3252.24 | 115.35 | 5e-09 |
| fu_common_j07233_m05554_p672 | 7233 | 5554 | 1.30 | 3347.13 | 127.25 | 5e-09 |
| fu_common_j07426_m04291_p672 | 7426 | 4291 | 1.73 | 3314.88 | 106.23 | 5e-09 |
| fu_common_j08501_m06284_p672 | 8501 | 6284 | 1.35 | 3308.99 | 118.64 | 3e-09 |
| fu_common_j08647_m09919_p672 | 8647 | 9919 | 0.87 | 3267.82 | 125.35 | 2e-09 |
| fu_common_j08871_m10045_p672 | 8871 | 10045 | 0.88 | 3265.06 | 117.18 | 2e-09 |
| fu_common_j08971_m06674_p672 | 8971 | 6674 | 1.34 | 3320.94 | 119.82 | 3e-09 |
| fu_common_j09587_m01463_p672 | 9587 | 1463 | 6.55 | 3333.23 | 85.05 | 9e-09 |
| fu_common_j09706_m06043_p672 | 9706 | 6043 | 1.61 | 3300.95 | 122.38 | 3e-09 |
| fu_common_j11000_m10075_p672 | 11000 | 10075 | 1.09 | 3321.11 | 106.11 | 1e-09 |
| fu_common_j12578_m03430_p672 | 12578 | 3430 | 3.67 | 3273.65 | 121.64 | 4e-09 |
| fu_common_j13895_m10263_p672 | 13895 | 10263 | 1.35 | 3306.92 | 124.62 | 1e-09 |
| fu_common_j14066_m07701_p672 | 14066 | 7701 | 1.83 | 3288.88 | 101.67 | 1e-09 |
| fu_common_j14843_m10088_p672 | 14843 | 10088 | 1.47 | 3242.35 | 136.96 | 1e-09 |
| fu_common_j15594_m05984_p672 | 15594 | 5984 | 2.61 | 3276.64 | 103.36 | 2e-09 |
| fu_common_j16667_m13944_p672 | 16667 | 13944 | 1.20 | 3289.29 | 145.97 | 9e-10 |
| fu_common_j17114_m08130_p672 | 17114 | 8130 | 2.11 | 3328.85 | 125.19 | 1e-09 |
| fu_common_j18611_m03805_p672 | 18611 | 3805 | 4.89 | 3328.21 | 110.82 | 2e-09 |
| fu_common_j18633_m08225_p672 | 18633 | 8225 | 2.27 | 3309.08 | 109.04 | 1e-09 |
| fu_common_j18948_m05167_p672 | 18948 | 5167 | 3.67 | 3349.06 | 108.11 | 2e-09 |
| fu_common_j22029_m11659_p672 | 22029 | 11659 | 1.89 | 3298.54 | 128.75 | 7e-10 |
| fu_common_j22112_m11575_p672 | 22112 | 11575 | 1.91 | 3290.13 | 104.46 | 6e-10 |
| fu_common_j24247_m10486_p672 | 24247 | 10486 | 2.31 | 3324.35 | 121.27 | 7e-10 |
| fu_common_j25312_m08184_p672 | 25312 | 8184 | 3.09 | 3320.86 | 121.18 | 9e-10 |
| fu_common_j29454_m08194_p672 | 29454 | 8194 | 3.59 | 3335.97 | 113.83 | 7e-10 |
| fu_common_j32335_m11748_p672 | 32335 | 11748 | 2.75 | 3334.53 | 119.46 | 5e-10 |
| fu_common_j33132_m09916_p672 | 33132 | 9916 | 3.34 | 3327.66 | 99.17 | 4e-10 |

**Table A.8**
Details of the 82 `Rare` instances. The number of jobs and machines are absolute numbers. Demands, number of sessions per job, and density are averages.

| Instance name | Jobs | Machines | Ratio | Average | | |
|---|---|---|---|---|---|---|
| | | | | Demand | Sessions | Density |
| fu_rare_j00107_m01676_p672 | 107 | 1676 | 0.06 | 3072.90 | 85.40 | 7e-07 |
| fu_rare_j00159_m01831_p672 | 159 | 1831 | 0.09 | 3257.36 | 64.33 | 3e-07 |
| fu_rare_j00175_m01262_p672 | 175 | 1262 | 0.14 | 3275.89 | 71.15 | 5e-07 |
| fu_rare_j00218_m03201_p672 | 218 | 3201 | 0.07 | 3296.88 | 55.37 | 1e-07 |
| fu_rare_j00359_m02338_p672 | 359 | 2338 | 0.15 | 3333.26 | 78.16 | 1e-07 |
| fu_rare_j00476_m01905_p672 | 476 | 1905 | 0.25 | 3273.28 | 65.16 | 1e-07 |
| fu_rare_j00507_m01909_p672 | 507 | 1909 | 0.27 | 3320.87 | 91.13 | 1e-07 |
| fu_rare_j00535_m10742_p672 | 535 | 10742 | 0.05 | 3339.66 | 80.40 | 2e-08 |
| fu_rare_j00554_m01849_p672 | 554 | 1849 | 0.30 | 3301.08 | 78.96 | 1e-07 |
| fu_rare_j00555_m03789_p672 | 555 | 3789 | 0.15 | 3313.87 | 82.20 | 6e-08 |
| fu_rare_j00635_m01960_p672 | 635 | 1960 | 0.32 | 3314.65 | 80.23 | 1e-07 |
| fu_rare_j00662_m03869_p672 | 662 | 3869 | 0.17 | 3343.08 | 73.37 | 4e-08 |
| fu_rare_j00666_m01887_p672 | 666 | 1887 | 0.35 | 3354.23 | 69.88 | 8e-08 |
| fu_rare_j00669_m02526_p672 | 669 | 2526 | 0.26 | 3341.35 | 80.31 | 7e-08 |
| fu_rare_j00695_m02541_p672 | 695 | 2541 | 0.27 | 3338.36 | 80.95 | 7e-08 |
| fu_rare_j00712_m02322_p672 | 712 | 2322 | 0.31 | 3407.19 | 76.90 | 7e-08 |
| fu_rare_j00722_m01444_p672 | 722 | 1444 | 0.50 | 3390.58 | 76.40 | 1e-07 |
| fu_rare_j00744_m03940_p672 | 744 | 3940 | 0.19 | 3357.85 | 90.98 | 5e-08 |
| fu_rare_j00749_m03927_p672 | 749 | 3927 | 0.19 | 3310.87 | 82.56 | 4e-08 |
| fu_rare_j00756_m03882_p672 | 756 | 3882 | 0.19 | 3341.80 | 79.82 | 4e-08 |
| fu_rare_j00763_m04120_p672 | 763 | 4120 | 0.19 | 3360.42 | 61.55 | 3e-08 |
| fu_rare_j00784_m03994_p672 | 784 | 3994 | 0.20 | 3333.06 | 69.96 | 3e-08 |
| fu_rare_j00858_m02332_p672 | 858 | 2332 | 0.37 | 3347.32 | 83.63 | 6e-08 |
| fu_rare_j00861_m02582_p672 | 861 | 2582 | 0.33 | 3319.86 | 77.72 | 5e-08 |
| fu_rare_j00961_m05095_p672 | 961 | 5095 | 0.19 | 3310.72 | 81.81 | 2e-08 |
| fu_rare_j00963_m05633_p672 | 963 | 5633 | 0.17 | 3350.70 | 90.45 | 2e-08 |
| fu_rare_j00963_m09116_p672 | 963 | 9116 | 0.11 | 3302.18 | 77.34 | 1e-08 |
| fu_rare_j00977_m01884_p672 | 977 | 1884 | 0.52 | 3341.66 | 60.98 | 5e-08 |
| fu_rare_j00978_m02822_p672 | 978 | 2822 | 0.35 | 3361.31 | 85.69 | 5e-08 |
| fu_rare_j01024_m04159_p672 | 1024 | 4159 | 0.25 | 3311.88 | 91.55 | 3e-08 |
| fu_rare_j01049_m03104_p672 | 1049 | 3104 | 0.34 | 3384.40 | 77.45 | 4e-08 |
| fu_rare_j01077_m04918_p672 | 1077 | 4918 | 0.22 | 3295.23 | 83.87 | 2e-08 |
| fu_rare_j01082_m05883_p672 | 1082 | 5883 | 0.18 | 3283.99 | 82.18 | 2e-08 |
| fu_rare_j01117_m07460_p672 | 1117 | 7460 | 0.15 | 3289.81 | 81.35 | 1e-08 |
| fu_rare_j01125_m04020_p672 | 1125 | 4020 | 0.28 | 3307.66 | 92.51 | 3e-08 |
| fu_rare_j01133_m05012_p672 | 1133 | 5012 | 0.23 | 3297.86 | 82.77 | 2e-08 |
| fu_rare_j01198_m04023_p672 | 1198 | 4023 | 0.30 | 3360.53 | 82.59 | 3e-08 |
| fu_rare_j01326_m02236_p672 | 1326 | 2236 | 0.59 | 3284.22 | 79.74 | 4e-08 |
| fu_rare_j01372_m09337_p672 | 1372 | 9337 | 0.15 | 3276.27 | 89.38 | 1e-08 |
| fu_rare_j01399_m03783_p672 | 1399 | 3783 | 0.37 | 3387.22 | 75.11 | 2e-08 |
| fu_rare_j01408_m04054_p672 | 1408 | 4054 | 0.35 | 3357.95 | 88.35 | 2e-08 |
| fu_rare_j01504_m05047_p672 | 1504 | 5047 | 0.30 | 3321.49 | 81.29 | 2e-08 |
| fu_rare_j01524_m01720_p672 | 1524 | 1720 | 0.89 | 3392.97 | 86.60 | 5e-08 |
| fu_rare_j01557_m05266_p672 | 1557 | 5266 | 0.30 | 3324.44 | 81.41 | 1e-08 |
| fu_rare_j01579_m06379_p672 | 1579 | 6379 | 0.25 | 3362.43 | 82.79 | 1e-08 |
| fu_rare_j01588_m04460_p672 | 1588 | 4460 | 0.36 | 3389.02 | 87.66 | 2e-08 |
| fu_rare_j01664_m03940_p672 | 1664 | 3940 | 0.42 | 3376.92 | 78.72 | 2e-08 |
| fu_rare_j01721_m05457_p672 | 1721 | 5457 | 0.32 | 3334.34 | 85.81 | 1e-08 |
| fu_rare_j01769_m14235_p672 | 1769 | 14235 | 0.12 | 3361.27 | 87.33 | 5e-09 |
| fu_rare_j01957_m08911_p672 | 1957 | 8911 | 0.22 | 3326.64 | 88.66 | 8e-09 |
| fu_rare_j01976_m05570_p672 | 1976 | 5570 | 0.35 | 3379.76 | 90.69 | 1e-08 |
| fu_rare_j01982_m05463_p672 | 1982 | 5463 | 0.36 | 3375.82 | 86.65 | 1e-08 |
| fu_rare_j02113_m01415_p672 | 2113 | 1415 | 1.49 | 3419.06 | 60.55 | 3e-08 |
| fu_rare_j02113_m08312_p672 | 2113 | 8312 | 0.25 | 3362.57 | 91.51 | 8e-09 |
| fu_rare_j02542_m03331_p672 | 2542 | 3331 | 0.76 | 3334.19 | 85.92 | 2e-08 |
| fu_rare_j02704_m08309_p672 | 2704 | 8309 | 0.33 | 3394.20 | 75.36 | 5e-09 |
| fu_rare_j02869_m07184_p672 | 2869 | 7184 | 0.40 | 3337.69 | 75.74 | 5e-09 |
| fu_rare_j02873_m12267_p672 | 2873 | 12267 | 0.23 | 3295.06 | 93.09 | 4e-09 |
| fu_rare_j03008_m09551_p672 | 3008 | 9551 | 0.31 | 3324.89 | 91.87 | 5e-09 |
| fu_rare_j03082_m24956_p672 | 3082 | 24956 | 0.12 | 3326.36 | 79.15 | 2e-09 |
| fu_rare_j03416_m08906_p672 | 3416 | 8906 | 0.38 | 3375.46 | 93.37 | 5e-09 |
| fu_rare_j03440_m05758_p672 | 3440 | 5758 | 0.60 | 3347.63 | 77.36 | 6e-09 |
| fu_rare_j03751_m11593_p672 | 3751 | 11593 | 0.32 | 3335.52 | 99.95 | 3e-09 |
| fu_rare_j04062_m07353_p672 | 4062 | 7353 | 0.55 | 3365.59 | 89.94 | 4e-09 |
| fu_rare_j04365_m07548_p672 | 4365 | 7548 | 0.58 | 3346.14 | 79.67 | 4e-09 |
| fu_rare_j04580_m03800_p672 | 4580 | 3800 | 1.21 | 3359.58 | 80.84 | 7e-09 |
| fu_rare_j04632_m04780_p672 | 4632 | 4780 | 0.97 | 3379.90 | 79.54 | 5e-09 |
| fu_rare_j04702_m09247_p672 | 4702 | 9247 | 0.51 | 3309.77 | 76.06 | 3e-09 |
| fu_rare_j05066_m10200_p672 | 5066 | 10200 | 0.50 | 3322.35 | 91.47 | 3e-09 |
| fu_rare_j05253_m09421_p672 | 5253 | 9421 | 0.56 | 3347.94 | 85.40 | 3e-09 |
| fu_rare_j05902_m07910_p672 | 5902 | 7910 | 0.75 | 3340.16 | 82.51 | 3e-09 |

(*continued on next page*)

**Table A.8** (*continued*).

| Instance name | Jobs | Machines | Ratio | Average | | |
|---|---|---|---|---|---|---|
| | | | | Demand | Sessions | Density |
| fu_rare_j06269_m14379_p672 | 6269 | 14379 | 0.44 | 3304.26 | 90.68 | 1e-09 |
| fu_rare_j06348_m07034_p672 | 6348 | 7034 | 0.90 | 3319.87 | 79.90 | 3e-09 |
| fu_rare_j06568_m07266_p672 | 6568 | 7266 | 0.90 | 3355.96 | 82.11 | 3e-09 |
| fu_rare_j06954_m17338_p672 | 6954 | 17338 | 0.40 | 3334.14 | 103.48 | 1e-09 |
| fu_rare_j07103_m10437_p672 | 7103 | 10437 | 0.68 | 3345.58 | 85.61 | 2e-09 |
| fu_rare_j08013_m15179_p672 | 8013 | 15179 | 0.53 | 3324.62 | 99.47 | 1e-09 |
| fu_rare_j08265_m09084_p672 | 8265 | 9084 | 0.91 | 3333.40 | 76.11 | 2e-09 |
| fu_rare_j09730_m08806_p672 | 9730 | 8806 | 1.10 | 3349.61 | 59.92 | 1e-09 |
| fu_rare_j10179_m10123_p672 | 10179 | 10123 | 1.01 | 3314.10 | 78.78 | 1e-09 |
| fu_rare_j11445_m23107_p672 | 11445 | 23107 | 0.50 | 3289.63 | 81.45 | 5e-10 |
| fu_rare_j16844_m05561_p672 | 16844 | 5561 | 3.03 | 3317.33 | 78.92 | 1e-09 |

**Table B.9**
`irace` results for BRKGA.

| $p$ | % $p_e$ | % $p_m$ | $\rho$ | # pop. | Int. exch. | # ind. exch. | Reset Int. |
|---|---|---|---|---|---|---|---|
| 181 | 0.26 | 0.10 | 0.58 | 1 | 92 | 1 | 500 |
| 142 | 0.28 | 0.10 | 0.50 | 1 | 93 | 1 | 500 |
| 140 | 0.43 | 0.11 | 0.63 | 1 | 93 | 1 | 480 |

**Table B.10**
`irace` results for SA.

| $st$ | $\alpha$ | $rt$ |
|---|---|---|
| 450 | 0.9500 | 4900 |
| 562 | 0.9723 | 4144 |
| 385 | 0.9463 | 4795 |

**Table B.11**
`irace` results for TS-FULL and TS-RD.

| TS-FULL | | TS-RD | | |
|---|---|---|---|---|
| $mt$ | $rt$ | $mt$ | $rt$ | $maxneigh$ |
| 1.00 | 170 | 0.75 | 450 | 0.015 |
| 1.06 | 137 | 1.00 | 207 | 0.015 |
| 1.03 | 168 | 0.78 | 489 | 0.010 |

first line of each table, rounded up to two digits for real values, and round to the next multiple of 10, in the case of integer values.

For the simulating annealing SA, we use the following ranges: start temperature $st \in [10, 1000]$; attenuation factor $\alpha \in [0.700, 0.999]$; reset interval $rt \in [500, 5000]$. Table B.10 shows the suggestion made by `irace`.

For the tabu searches TS-FULL and TS-RD, we use the following ranges: tabu percentage size (with respect to the number of jobs) $mt \in [0.5, 2]$; reset interval $rt \in [100, 5000]$; (only for TS-RD) maximum number of neighbors to be visited (with respect to the number of jobs) $maxneigh \in [0.01, 0.30]$. Table B.11 shows the suggestion made by `irace`.

And finally, for the traditional genetic algorithm TGA, we use the following ranges: population size $p \in [50, 700]$; mutation probability $\mu \in [0.01, 0.05]$; number of populations in $[1, 2]$; interval for individual exchange in $[50, 100]$ iterations; number of elite individuals exchanged in $[1, 2]$; and number of iterations without improvement in the best solution until population reset in $[200, 500]$. Table B.12 shows the suggestion made by `irace`.

## Appendix C. General results for TMDSP-MT

Tables C.13–C.16 show detailed results for each instance and each algorithm considering the TMDSP-MT variation. The first column describes the instance. The second column depicts the lower bound obtained by CPLEX and the third column is the total completion time of the best solution found among the algorithms. The following column shows the integrality gap between the lower bound and the best solution found. Note that for the FU instances, we have only a few of lower bounds and gaps, since CPLEX could not solve most of such instances due to memory problems as explained on Section 8.4. The following columns show the percentage deviation from the average total completion time of the solutions found by a given algorithm to the best solution found. A star ($\star$) indicates that the algorithm found the best solution in at least one run.

## Appendix D. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.asoc.2019.105575.

**Table B.12**
`irace` results for TGA.

| $p$ | % $\mu$ | # pop. | Int. exch. | # ind. exch. | Reset Int. |
|---|---|---|---|---|---|
| 60 | 0.0272 | 1 | 92 | 1 | 500 |
| 51 | 0.0246 | 1 | 95 | 1 | 489 |
| 58 | 0.0235 | 1 | 99 | 1 | 493 |
| 68 | 0.0289 | 1 | 90 | 1 | 495 |

**Table C.13**
Best results for each synthetic loose instance, and the percentage deviation of the best solution found for each algorithm. The table also shows the best lower bound and the integrality gap between the lower bound and the best solution.

| Instance | Lower bound | Best solution | Gap% | Average deviation % | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | BRKGA | CPLEX | ILS | SA | TGA | TS-FULL | TS-RD |
| sy_loose_j20_m05_p010_01 | 4208.04 | 5568 | 32.32 | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ |
| sy_loose_j20_m05_p010_02 | 3666.19 | 5630 | 53.57 | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ |
| sy_loose_j20_m05_p010_03 | 4051.82 | 5018 | 23.85 | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ |
| sy_loose_j20_m05_p010_04 | 4132.98 | 4962 | 20.06 | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ |
| sy_loose_j20_m05_p010_05 | 4679.01 | 6171 | 31.89 | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ |
| sy_loose_j20_m05_p010_06 | 4565.89 | 6338 | 38.81 | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ |
| sy_loose_j20_m05_p010_07 | 3432.17 | 4555 | 32.71 | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ |
| sy_loose_j20_m05_p010_08 | 5974.66 | 8471 | 41.78 | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ |
| sy_loose_j20_m05_p010_09 | 4093.66 | 5067 | 23.78 | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ |
| sy_loose_j20_m05_p010_10 | 5953.78 | 9483 | 59.28 | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ |
| sy_loose_j50_m10_p050_01 | 3997.62 | 6600 | 65.10 | 1.92 | ⋆ | 2.03 | 2.09 | 6.89 | 2.48 | 6.00 |
| sy_loose_j50_m10_p050_02 | 3675.80 | 7543 | 105.21 | 0.12 | ⋆ | ⋆ | 0.68 | 3.49 | 0.74 | 3.25 |
| sy_loose_j50_m10_p050_03 | 4439.00 | 8216 | 85.09 | ⋆ | 0.19 | 0.17 | 0.52 | 3.50 | 1.41 | 3.64 |
| sy_loose_j50_m10_p050_04 | 4934.54 | 7646 | 54.95 | 1.40 | ⋆ | 1.39 | 1.43 | 4.26 | 2.13 | 3.70 |
| sy_loose_j50_m10_p050_05 | 3993.95 | 7435 | 86.16 | 1.12 | ⋆ | 0.73 | 1.17 | 4.34 | 1.69 | 4.20 |
| sy_loose_j50_m10_p050_06 | 5418.89 | 8649 | 59.61 | 1.43 | ⋆ | 1.48 | 2.17 | 4.17 | 2.40 | 3.67 |
| sy_loose_j50_m10_p050_07 | 4654.69 | 7664 | 64.65 | 0.82 | ⋆ | 0.80 | 1.06 | 4.84 | 2.40 | 4.55 |
| sy_loose_j50_m10_p050_08 | 4404.33 | 7414 | 68.33 | 2.62 | ⋆ | 2.58 | 2.93 | 6.00 | 3.29 | 5.66 |
| sy_loose_j50_m10_p050_09 | 3402.40 | 6880 | 102.21 | 3.08 | ⋆ | 3.17 | 3.95 | 6.56 | 3.55 | 6.77 |
| sy_loose_j50_m10_p050_10 | 4660.89 | 8715 | 86.98 | 1.15 | ⋆ | 1.29 | 2.97 | 5.63 | 3.01 | 5.14 |
| sy_loose_j50_m30_p100_01 | 1979.09 | 2382 | 20.36 | 0.34 | ⋆ | 0.08 | 0.34 | 4.11 | 1.72 | 4.28 |
| sy_loose_j50_m30_p100_02 | 1214.30 | 2137 | 75.99 | 3.79 | ⋆ | 3.79 | 4.63 | 7.53 | 4.91 | 6.97 |
| sy_loose_j50_m30_p100_03 | 1481.99 | 1980 | 33.60 | 2.32 | ⋆ | 2.32 | 2.32 | 6.92 | 3.43 | 6.72 |
| sy_loose_j50_m30_p100_04 | 1862.28 | 2446 | 31.34 | 1.92 | ⋆ | 1.96 | 2.41 | 6.17 | 3.48 | 5.85 |
| sy_loose_j50_m30_p100_05 | 1428.02 | 2403 | 68.27 | 2.21 | ⋆ | 2.00 | 2.95 | 5.95 | 3.04 | 4.99 |
| sy_loose_j50_m30_p100_06 | 1625.45 | 2116 | 30.18 | 1.13 | ⋆ | 1.23 | 1.37 | 4.02 | 1.23 | 3.97 |
| sy_loose_j50_m30_p100_07 | 1526.46 | 2232 | 46.22 | 4.08 | ⋆ | 4.26 | 4.57 | 8.20 | 6.09 | 7.21 |
| sy_loose_j50_m30_p100_08 | 1439.65 | 2351 | 63.30 | 4.17 | ⋆ | 4.34 | 4.42 | 9.10 | 5.32 | 8.51 |
| sy_loose_j50_m30_p100_09 | 1160.34 | 2156 | 85.81 | 4.04 | ⋆ | 4.04 | 4.50 | 8.26 | 5.10 | 6.96 |
| sy_loose_j50_m30_p100_10 | 1451.53 | 2167 | 49.29 | 3.23 | ⋆ | 3.23 | 3.92 | 6.88 | 4.61 | 6.69 |

**Table C.14**
Best results for each synthetic tight instance, and the percentage deviation of the best solution found for each algorithm. The table also shows the best lower bound and the integrality gap between the lower bound and the best solution.

| Instance | Lower bound | Best solution | Gap% | Average deviation % | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | BRKGA | CPLEX | ILS | SA | TGA | TS-FULL | TS-RD |
| sy_tight_j20_m05_p010_01 | 7526.00 | 7526 | – | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ |
| sy_tight_j20_m05_p010_02 | 11307.00 | 11307 | – | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ |
| sy_tight_j20_m05_p010_03 | 15853.00 | 15853 | – | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ |
| sy_tight_j20_m05_p010_04 | 9496.00 | 9496 | – | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ |
| sy_tight_j20_m05_p010_05 | 8722.00 | 8722 | – | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ |
| sy_tight_j20_m05_p010_06 | 9916.00 | 9916 | – | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ |
| sy_tight_j20_m05_p010_07 | 9496.00 | 9496 | – | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ |
| sy_tight_j20_m05_p010_08 | 7887.00 | 7887 | – | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ |
| sy_tight_j20_m05_p010_09 | 13445.00 | 13445 | – | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ |
| sy_tight_j20_m05_p010_10 | 7982.00 | 7982 | – | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ | ⋆ |
| sy_tight_j50_m10_p050_01 | 7632.76 | 11844 | 55.17 | 0.68 | ⋆ | 0.68 | 0.68 | 0.94 | 0.69 | 0.95 |
| sy_tight_j50_m10_p050_02 | 8338.93 | 11717 | 40.51 | 0.32 | ⋆ | 0.32 | 0.32 | 0.73 | 0.32 | 0.46 |
| sy_tight_j50_m10_p050_03 | 9539.72 | 12091 | 26.74 | 0.05 | ⋆ | 0.05 | 0.05 | 0.22 | 0.05 | 0.24 |
| sy_tight_j50_m10_p050_04 | 6968.42 | 9848 | 41.32 | ⋆ | ⋆ | ⋆ | ⋆ | 0.09 | ⋆ | 0.09 |
| sy_tight_j50_m10_p050_05 | 8261.06 | 10464 | 26.67 | 2.46 | ⋆ | 2.46 | 2.46 | 2.46 | 2.46 | 2.46 |
| sy_tight_j50_m10_p050_06 | 8177.55 | 11360 | 38.92 | 0.15 | ⋆ | 0.15 | 0.15 | 0.55 | 0.15 | 0.46 |
| sy_tight_j50_m10_p050_07 | 10436.19 | 13948 | 33.65 | 0.27 | ⋆ | 0.27 | 0.27 | 0.36 | 0.28 | 0.35 |
| sy_tight_j50_m10_p050_08 | 10386.82 | 11898 | 14.55 | 0.44 | ⋆ | 0.44 | 0.44 | 0.44 | 0.44 | 0.44 |
| sy_tight_j50_m10_p050_09 | 9613.38 | 11079 | 15.25 | 0.59 | ⋆ | 0.59 | 0.59 | 0.65 | 0.59 | 0.59 |
| sy_tight_j50_m10_p050_10 | 8154.53 | 11576 | 41.96 | 0.09 | ⋆ | 0.09 | 0.09 | 0.13 | 0.09 | 0.09 |
| sy_tight_j50_m30_p100_01 | 2942.43 | 3978 | 35.19 | ⋆ | 0.23 | ⋆ | ⋆ | 1.84 | 0.23 | 2.16 |
| sy_tight_j50_m30_p100_02 | 2802.34 | 3917 | 39.78 | 0.03 | 0.18 | ⋆ | 0.10 | 1.97 | 0.20 | 1.66 |
| sy_tight_j50_m30_p100_03 | 3364.36 | 4346 | 29.18 | ⋆ | ⋆ | ⋆ | ⋆ | 1.96 | 0.09 | 1.47 |
| sy_tight_j50_m30_p100_04 | 3074.38 | 3840 | 24.90 | 0.08 | 0.16 | ⋆ | 0.08 | 3.41 | 0.26 | 3.46 |
| sy_tight_j50_m30_p100_05 | 2946.91 | 4147 | 40.72 | ⋆ | ⋆ | ⋆ | ⋆ | 2.10 | 0.10 | 2.22 |
| sy_tight_j50_m30_p100_06 | 3059.57 | 3842 | 25.57 | ⋆ | 0.21 | ⋆ | ⋆ | 2.08 | 0.05 | 1.98 |
| sy_tight_j50_m30_p100_07 | 3096.29 | 4153 | 34.13 | ⋆ | 0.14 | ⋆ | ⋆ | 2.62 | 0.17 | 2.75 |
| sy_tight_j50_m30_p100_08 | 3034.61 | 4046 | 33.33 | ⋆ | ⋆ | ⋆ | ⋆ | 1.38 | 0.02 | 1.68 |
| sy_tight_j50_m30_p100_09 | 2869.98 | 3895 | 35.72 | ⋆ | 0.03 | ⋆ | 0.03 | 2.16 | 0.15 | 2.26 |
| sy_tight_j50_m30_p100_10 | 3117.91 | 4273 | 37.05 | 0.05 | ⋆ | ⋆ | 0.16 | 1.68 | 0.35 | 1.59 |

**Table C.15**
Best results for Common subset with respect to the total completion time, and the percentage deviation of the average of solutions found for each algorithm. Note that we could not find valid lower bounds and compute the gaps that are, therefore, omitted from the table. Also note that CPLEX was able to find only one solution in this subset.

| Instance | Best value | Average deviation % | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | BRKGA | CPLEX | ILS | SA | TGA | TS-FULL | TS-RD |
| fu_common_j00505_m01902_p672 | 656010 | ⋆ | 20.34 | 22.86 | 16.33 | 19.77 | 21.85 | 21.28 |
| fu_common_j00586_m01497_p672 | 728067 | ⋆ | – | 30.60 | 16.99 | 25.22 | 29.17 | 24.72 |
| fu_common_j00888_m03941_p672 | 1475013 | ⋆ | – | 16.96 | 10.82 | 13.85 | 15.99 | 13.89 |
| fu_common_j01562_m02572_p672 | 3243218 | ⋆ | – | 17.51 | 13.10 | 14.99 | 16.03 | 14.53 |
| fu_common_j01954_m02112_p672 | 10327265 | ⋆ | – | 9.60 | 4.78 | 7.70 | 8.81 | 7.79 |
| fu_common_j02102_m21913_p672 | 12166632 | ⋆ | – | 3.87 | 0.48 | 1.54 | 2.93 | 2.08 |
| fu_common_j02225_m02480_p672 | 11136897 | ⋆ | – | 7.38 | 3.96 | 5.88 | 6.51 | 5.40 |
| fu_common_j02304_m02242_p672 | 7314362 | ⋆ | – | 14.65 | 10.52 | 11.92 | 13.75 | 11.41 |
| fu_common_j02381_m02148_p672 | 6912754 | ⋆ | – | 16.25 | 13.54 | 14.50 | 15.76 | 13.97 |
| fu_common_j02441_m02307_p672 | 7364892 | ⋆ | – | 12.43 | 9.15 | 10.80 | 11.40 | 10.15 |
| fu_common_j02657_m04023_p672 | 9070347 | ⋆ | – | 9.99 | 7.63 | 7.82 | 9.51 | 8.19 |
| fu_common_j02754_m04811_p672 | 5656134 | ⋆ | – | 9.31 | 6.55 | 7.91 | 8.81 | 8.23 |
| fu_common_j02912_m02817_p672 | 6682745 | ⋆ | – | 12.69 | 11.03 | 10.86 | 11.73 | 10.61 |
| fu_common_j03211_m03220_p672 | 24927222 | ⋆ | – | 6.09 | 2.92 | 4.15 | 4.93 | 3.74 |
| fu_common_j03267_m04576_p672 | 8395021 | ⋆ | – | 11.34 | 9.66 | 9.89 | 10.91 | 9.92 |
| fu_common_j03298_m01645_p672 | 29121761 | ⋆ | – | 5.81 | 3.82 | 4.84 | 5.43 | 4.46 |
| fu_common_j03310_m02915_p672 | 27182273 | ⋆ | – | 5.73 | 2.56 | 4.20 | 4.82 | 4.07 |
| fu_common_j03320_m04583_p672 | 12919845 | ⋆ | – | 7.70 | 5.50 | 6.08 | 7.01 | 6.11 |
| fu_common_j03426_m04741_p672 | 10496866 | ⋆ | – | 9.90 | 7.86 | 8.43 | 9.37 | 8.65 |
| fu_common_j03593_m02974_p672 | 10962105 | ⋆ | – | 8.93 | 7.55 | 7.39 | 8.21 | 7.10 |
| fu_common_j03696_m04574_p672 | 16849101 | ⋆ | – | 6.09 | 4.45 | 4.68 | 5.47 | 4.71 |
| fu_common_j03716_m02145_p672 | 15976560 | ⋆ | – | 9.93 | 8.27 | 8.27 | 9.14 | 7.89 |
| fu_common_j03734_m05485_p672 | 12227635 | ⋆ | – | 7.85 | 6.36 | 6.44 | 7.36 | 6.23 |
| fu_common_j03873_m05287_p672 | 6485890 | ⋆ | – | 11.07 | 9.14 | 8.54 | 10.27 | 8.97 |
| fu_common_j03897_m02588_p672 | 16826093 | ⋆ | – | 8.96 | 7.49 | 7.56 | 8.56 | 5.62 |
| fu_common_j03918_m04835_p672 | 8192353 | ⋆ | – | 11.21 | 9.39 | 9.71 | 10.95 | 9.60 |
| fu_common_j03978_m10039_p672 | 5053730 | ⋆ | – | 10.36 | 9.08 | 8.31 | 9.64 | 9.03 |
| fu_common_j04143_m06535_p672 | 10068467 | ⋆ | – | 11.24 | 9.36 | 9.83 | 10.65 | 10.10 |
| fu_common_j04241_m03139_p672 | 14479742 | ⋆ | – | 9.32 | 8.04 | 7.92 | 8.61 | 7.69 |
| fu_common_j04412_m05650_p672 | 9716415 | ⋆ | – | 9.29 | 8.39 | 7.85 | 8.71 | 8.07 |
| fu_common_j04525_m06837_p672 | 10035627 | ⋆ | – | 10.04 | 8.49 | 8.62 | 9.50 | 8.75 |
| fu_common_j04574_m03788_p672 | 26697323 | ⋆ | – | 6.41 | 4.29 | 4.46 | 5.72 | 4.49 |
| fu_common_j04604_m04813_p672 | 17426017 | ⋆ | – | 7.37 | 5.99 | 5.90 | 6.84 | 6.38 |
| fu_common_j04888_m08869_p672 | 11929438 | ⋆ | – | 7.67 | 6.28 | 6.47 | 7.22 | 6.93 |
| fu_common_j04961_m06153_p672 | 15086086 | ⋆ | – | 8.29 | 6.83 | 6.63 | 7.57 | 7.01 |
| fu_common_j05514_m05088_p672 | 39422053 | ⋆ | – | 5.34 | 4.29 | 4.08 | 4.70 | 4.23 |
| fu_common_j05710_m10019_p672 | 7969007 | ⋆ | – | 8.49 | 7.46 | 7.10 | 8.31 | 7.92 |
| fu_common_j05783_m07209_p672 | 20696327 | ⋆ | – | 8.78 | 7.21 | 7.13 | 8.40 | 7.98 |
| fu_common_j05822_m01844_p672 | 35541222 | ⋆ | – | 6.06 | 4.52 | 4.96 | 4.99 | 4.87 |
| fu_common_j06167_m05038_p672 | 44628559 | ⋆ | – | 5.46 | 3.98 | 3.84 | 4.51 | 4.28 |
| fu_common_j06274_m05911_p672 | 16579605 | ⋆ | – | 9.81 | 8.46 | 8.25 | 8.94 | 8.69 |
| fu_common_j06407_m02296_p672 | 15044964 | ⋆ | – | 11.61 | 10.66 | 10.02 | 10.94 | 10.08 |
| fu_common_j06410_m04644_p672 | 35673602 | ⋆ | – | 9.49 | 8.37 | 7.47 | 9.04 | 8.54 |
| fu_common_j06809_m20225_p672 | 12326827 | ⋆ | – | 5.25 | 4.28 | 3.61 | 4.76 | 4.68 |
| fu_common_j06919_m05493_p672 | 16895108 | ⋆ | – | 9.46 | 8.46 | 7.85 | 8.97 | 8.64 |
| fu_common_j07233_m05554_p672 | 42484255 | ⋆ | – | 6.18 | 5.58 | 5.30 | 5.92 | 5.77 |
| fu_common_j07426_m04291_p672 | 33780469 | ⋆ | – | 7.10 | 6.27 | 5.63 | 6.66 | 6.18 |
| fu_common_j08501_m06284_p672 | 44804925 | ⋆ | – | 5.08 | 4.30 | 4.53 | 4.70 | 4.31 |
| fu_common_j08647_m09919_p672 | 22129297 | ⋆ | – | 5.83 | 4.70 | 4.82 | 5.17 | 5.17 |
| fu_common_j08871_m10045_p672 | 24775901 | ⋆ | – | 8.69 | 7.68 | 6.78 | 8.30 | 7.74 |
| fu_common_j08971_m06674_p672 | 52255646 | ⋆ | – | 4.95 | 4.23 | 3.97 | 4.52 | 4.41 |
| fu_common_j09587_m01463_p672 | 48020354 | ⋆ | – | 9.95 | 8.56 | 8.29 | 9.14 | 9.01 |
| fu_common_j09706_m06043_p672 | 46264728 | ⋆ | – | 5.94 | 4.71 | 4.74 | 5.44 | 5.34 |
| fu_common_j11000_m10075_p672 | 52421049 | ⋆ | – | 4.96 | 4.28 | 4.39 | 4.78 | 4.44 |
| fu_common_j12578_m03430_p672 | 50513078 | ⋆ | – | 6.78 | 6.35 | 5.90 | 6.30 | 6.30 |
| fu_common_j13895_m10263_p672 | 71377211 | ⋆ | – | 5.62 | 5.08 | 4.34 | 5.25 | 5.05 |
| fu_common_j14066_m07701_p672 | 69320398 | ⋆ | – | 5.90 | 5.12 | 4.77 | 5.45 | 5.48 |
| fu_common_j14843_m10088_p672 | 40070093 | ⋆ | – | 5.76 | 5.24 | 4.51 | 5.43 | 5.32 |
| fu_common_j15594_m05984_p672 | 69762976 | ⋆ | – | 5.42 | 4.97 | 4.46 | 5.13 | 5.07 |
| fu_common_j16667_m13944_p672 | 70302542 | ⋆ | – | 3.34 | 3.32 | 2.67 | 2.89 | 3.07 |
| fu_common_j17114_m08130_p672 | 86694961 | ⋆ | – | 5.83 | 5.15 | 4.86 | 5.51 | 5.21 |
| fu_common_j18611_m03805_p672 | 92343781 | ⋆ | – | 5.81 | 5.48 | 4.88 | 5.66 | 5.44 |
| fu_common_j18633_m08225_p672 | 90949291 | ⋆ | – | 3.95 | 3.45 | 3.38 | 3.73 | 3.48 |
| fu_common_j18948_m05167_p672 | 114078062 | ⋆ | – | 4.02 | 3.38 | 3.33 | 3.50 | 3.43 |
| fu_common_j22029_m11659_p672 | 72843618 | ⋆ | – | 4.66 | 4.56 | 3.78 | 4.45 | 4.57 |
| fu_common_j22112_m11575_p672 | 101278908 | ⋆ | – | 4.08 | 3.84 | 3.20 | 3.92 | 3.83 |
| fu_common_j24247_m10486_p672 | 131611326 | ⋆ | – | 3.29 | 2.98 | 2.60 | 3.05 | 3.15 |
| fu_common_j25312_m08184_p672 | 123772763 | ⋆ | – | 4.43 | 4.08 | 3.38 | 4.18 | 4.21 |
| fu_common_j29454_m08194_p672 | 190501918 | ⋆ | – | 3.25 | 2.93 | 2.53 | 2.96 | 2.98 |
| fu_common_j32335_m11748_p672 | 221012193 | ⋆ | – | 2.57 | 2.44 | 1.71 | 2.24 | 2.41 |
| fu_common_j33132_m09916_p672 | 195028313 | ⋆ | – | 2.80 | 2.54 | 1.93 | 2.44 | 2.63 |

**Table C.16**
Best results for Rare subset with respect to the total completion time, and the percentage deviation of the average of solutions found for each algorithm. Note that we could not find valid lower bounds and compute the gaps for some instances. However, we report the incumbent solutions for CPLEX in such cases.

| Instance | Lower bound | Best solution | Gap% | Average deviation % | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | BRKGA | CPLEX | ILS | SA | TGA | TS-FULL | TS-RD |
| fu_rare_j00107_m01676_p672 | 8362.90 | 36691 | 77.20 | ⋆ | 2.14 | 1.20 | 5.73 | 11.71 | 4.21 | 12.41 |
| fu_rare_j00159_m01831_p672 | 7444.01 | 58715 | 87.30 | ⋆ | 3.54 | 4.97 | 9.46 | 16.61 | 5.46 | 15.83 |
| fu_rare_j00175_m01262_p672 | 11506.85 | 64569 | 82.20 | ⋆ | 1.92 | 0.86 | 6.37 | 16.50 | 5.13 | 16.50 |
| fu_rare_j00218_m03201_p672 | 11778.00 | 92856 | 87.30 | ⋆ | 9.21 | 9.91 | 4.71 | 10.78 | 11.10 | 12.14 |
| fu_rare_j00359_m02338_p672 | – | 172250 | – | ⋆ | – | 20.68 | 10.12 | 18.85 | 20.74 | 19.65 |
| fu_rare_j00476_m01905_p672 | – | 228349 | – | ⋆ | – | 25.72 | 15.85 | 23.76 | 22.80 | 23.13 |
| fu_rare_j00507_m01909_p672 | 11307.00 | 389875 | 97.10 | ⋆ | 4.44 | 22.25 | 15.72 | 20.31 | 21.27 | 21.65 |
| fu_rare_j00535_m10742_p672 | – | 563284 | – | ⋆ | – | 12.60 | 2.31 | 8.56 | 10.19 | 8.76 |
| fu_rare_j00554_m01849_p672 | – | 520349 | – | ⋆ | – | 21.57 | 14.41 | 18.85 | 19.93 | 19.18 |
| fu_rare_j00555_m03789_p672 | – | 305078 | – | ⋆ | – | 23.50 | 14.57 | 19.97 | 22.54 | 19.94 |
| fu_rare_j00635_m01960_p672 | – | 375554 | – | ⋆ | – | 21.92 | 15.16 | 21.16 | 23.29 | 20.95 |
| fu_rare_j00662_m03869_p672 | – | 304365 | – | ⋆ | – | 18.11 | 10.42 | 13.82 | 17.06 | 13.94 |
| fu_rare_j00666_m01887_p672 | – | 405400 | – | ⋆ | – | 24.45 | 15.65 | 21.04 | 25.28 | 21.66 |
| fu_rare_j00669_m02526_p672 | 7857.00 | 317557 | 97.50 | ⋆ | 14.80 | 24.28 | 17.95 | 21.21 | 25.36 | 22.89 |
| fu_rare_j00695_m02541_p672 | – | 758099 | – | ⋆ | – | 20.16 | 14.03 | 16.96 | 19.07 | 17.18 |
| fu_rare_j00712_m02322_p672 | – | 837057 | – | ⋆ | – | 18.67 | 13.14 | 17.16 | 19.28 | 16.79 |
| fu_rare_j00722_m01444_p672 | – | 877214 | – | ⋆ | – | 20.52 | 13.69 | 16.80 | 18.77 | 16.50 |
| fu_rare_j00744_m03940_p672 | – | 481356 | – | ⋆ | – | 20.08 | 14.79 | 17.74 | 20.91 | 18.99 |
| fu_rare_j00749_m03927_p672 | – | 346927 | – | ⋆ | – | 30.22 | 21.64 | 26.05 | 29.23 | 25.70 |
| fu_rare_j00756_m03882_p672 | – | 481736 | – | ⋆ | – | 23.52 | 16.83 | 20.66 | 23.56 | 21.26 |
| fu_rare_j00763_m04120_p672 | – | 487375 | – | ⋆ | – | 20.11 | 12.42 | 16.28 | 18.78 | 16.31 |
| fu_rare_j00784_m03994_p672 | – | 570621 | – | ⋆ | – | 24.33 | 16.72 | 20.17 | 22.97 | 20.83 |
| fu_rare_j00858_m02332_p672 | – | 608517 | – | ⋆ | – | 23.53 | 17.74 | 20.74 | 23.36 | 21.96 |
| fu_rare_j00861_m02582_p672 | – | 535896 | – | ⋆ | – | 22.92 | 17.46 | 19.26 | 22.46 | 19.67 |
| fu_rare_j00961_m05095_p672 | – | 501647 | – | ⋆ | – | 24.33 | 17.21 | 20.65 | 22.68 | 21.67 |
| fu_rare_j00963_m05633_p672 | 9473.00 | 442457 | 97.00 | ⋆ | 19.00 | 28.64 | 21.63 | 22.96 | 27.35 | 23.63 |
| fu_rare_j00963_m09116_p672 | – | 313495 | – | ⋆ | – | 22.40 | 16.94 | 19.17 | 22.19 | 20.55 |
| fu_rare_j00977_m01884_p672 | – | 852979 | – | ⋆ | – | 22.07 | 16.10 | 19.74 | 20.72 | 18.84 |
| fu_rare_j00978_m02822_p672 | – | 676204 | – | ⋆ | – | 18.84 | 13.86 | 17.11 | 18.45 | 17.33 |
| fu_rare_j01024_m04159_p672 | – | 643008 | – | ⋆ | – | 22.09 | 16.45 | 18.94 | 21.07 | 19.41 |
| fu_rare_j01049_m03104_p672 | – | 947897 | – | ⋆ | – | 21.03 | 16.25 | 18.34 | 19.41 | 18.06 |
| fu_rare_j01077_m04918_p672 | 9782.00 | 460765 | 97.9 | ⋆ | 15.40 | 21.11 | 15.16 | 17.46 | 19.96 | 18.23 |
| fu_rare_j01082_m05883_p672 | – | 544346 | – | ⋆ | – | 23.41 | 15.77 | 19.65 | 22.46 | 20.41 |
| fu_rare_j01117_m07460_p672 | – | 528293 | – | ⋆ | – | 23.86 | 18.16 | 20.86 | 23.69 | 20.51 |
| fu_rare_j01125_m04020_p672 | – | 523704 | – | ⋆ | – | 29.61 | 22.00 | 25.42 | 28.44 | 26.41 |
| fu_rare_j01133_m05012_p672 | – | 643001 | – | ⋆ | – | 22.54 | 17.37 | 18.77 | 21.43 | 19.72 |
| fu_rare_j01198_m04023_p672 | – | 1206559 | – | ⋆ | – | 20.64 | 16.86 | 18.68 | 19.79 | 18.80 |
| fu_rare_j01326_m02236_p672 | 4592.00 | 563894 | 99.20 | ⋆ | 4.33 | 24.59 | 16.89 | 20.56 | 23.87 | 20.91 |
| fu_rare_j01372_m09337_p672 | – | 520507 | – | ⋆ | – | 15.01 | 11.03 | 12.07 | 14.61 | 12.82 |
| fu_rare_j01399_m03783_p672 | – | 1268856 | – | ⋆ | – | 16.81 | 12.98 | 14.09 | 16.31 | 14.19 |
| fu_rare_j01408_m04054_p672 | – | 1526915 | – | ⋆ | – | 20.40 | 15.40 | 16.80 | 19.05 | 18.25 |
| fu_rare_j01504_m05047_p672 | – | 732105 | – | ⋆ | – | 22.01 | 16.47 | 19.66 | 20.84 | 19.80 |
| fu_rare_j01524_m01720_p672 | – | 1230926 | – | ⋆ | – | 23.03 | 18.05 | 20.25 | 21.45 | 20.38 |
| fu_rare_j01557_m05266_p672 | – | 858004 | – | ⋆ | – | 23.07 | 19.24 | 19.27 | 22.85 | 20.55 |
| fu_rare_j01579_m06379_p672 | – | 948083 | – | ⋆ | – | 23.53 | 19.43 | 20.96 | 23.54 | 21.76 |
| fu_rare_j01588_m04460_p672 | – | 1438897 | – | ⋆ | – | 19.00 | 15.05 | 15.62 | 17.89 | 15.77 |
| fu_rare_j01664_m03940_p672 | – | 1235487 | – | ⋆ | – | 23.73 | 19.56 | 21.36 | 22.96 | 22.41 |
| fu_rare_j01721_m05457_p672 | – | 1306159 | – | ⋆ | – | 22.05 | 18.31 | 18.74 | 20.76 | 18.93 |
| fu_rare_j01769_m14235_p672 | – | 808573 | – | ⋆ | – | 20.85 | 16.06 | 17.60 | 20.33 | 17.57 |
| fu_rare_j01957_m08911_p672 | – | 1098967 | – | ⋆ | – | 17.62 | 14.14 | 14.73 | 17.22 | 15.52 |
| fu_rare_j01976_m05570_p672 | – | 1399640 | – | ⋆ | – | 22.78 | 18.77 | 19.78 | 22.08 | 19.90 |
| fu_rare_j01982_m05463_p672 | – | 1459049 | – | ⋆ | – | 21.86 | 17.32 | 18.82 | 21.40 | 19.27 |
| fu_rare_j02113_m01415_p672 | – | 1940254 | – | ⋆ | – | 22.34 | 18.33 | 18.05 | 21.60 | 19.29 |
| fu_rare_j02113_m08312_p672 | – | 1130960 | – | ⋆ | – | 18.17 | 13.58 | 14.99 | 17.05 | 16.01 |
| fu_rare_j02542_m03331_p672 | – | 1628543 | – | ⋆ | – | 19.56 | 16.55 | 16.88 | 19.16 | 17.51 |
| fu_rare_j02704_m08309_p672 | – | 2034304 | – | ⋆ | – | 17.07 | 14.47 | 14.10 | 16.30 | 15.46 |
| fu_rare_j02869_m07184_p672 | – | 1960423 | – | ⋆ | – | 18.21 | 15.67 | 15.60 | 17.37 | 16.32 |
| fu_rare_j02873_m12267_p672 | – | 1234246 | – | ⋆ | – | 13.70 | 10.00 | 11.09 | 13.30 | 12.12 |
| fu_rare_j03008_m09551_p672 | – | 1595022 | – | ⋆ | – | 16.63 | 13.86 | 14.47 | 16.22 | 15.31 |
| fu_rare_j03082_m24956_p672 | – | 1348141 | – | ⋆ | – | 3.79 | 2.16 | 1.85 | 3.58 | 3.08 |
| fu_rare_j03416_m08906_p672 | – | 2798602 | – | ⋆ | – | 14.27 | 11.64 | 12.47 | 13.34 | 12.26 |
| fu_rare_j03440_m05758_p672 | – | 2597595 | – | ⋆ | – | 17.78 | 15.46 | 15.99 | 17.35 | 16.41 |
| fu_rare_j03751_m11593_p672 | – | 2664099 | – | ⋆ | – | 14.88 | 12.57 | 12.17 | 14.15 | 13.68 |
| fu_rare_j04062_m07353_p672 | – | 3658726 | – | ⋆ | – | 16.54 | 14.91 | 13.88 | 15.89 | 15.46 |
| fu_rare_j04365_m07548_p672 | – | 3617263 | – | ⋆ | – | 15.57 | 13.92 | 13.80 | 14.95 | 14.28 |
| fu_rare_j04580_m03800_p672 | – | 3678566 | – | ⋆ | – | 22.08 | 19.49 | 18.91 | 21.50 | 19.82 |
| fu_rare_j04632_m04780_p672 | – | 4423922 | – | ⋆ | – | 16.26 | 14.42 | 14.35 | 15.97 | 14.87 |
| fu_rare_j04702_m09247_p672 | – | 3548266 | – | ⋆ | – | 15.54 | 13.45 | 13.18 | 14.95 | 13.91 |
| fu_rare_j05066_m10200_p672 | – | 3294674 | – | ⋆ | – | 17.02 | 15.38 | 15.22 | 16.89 | 16.33 |
| fu_rare_j05253_m09421_p672 | – | 4778320 | – | ⋆ | – | 12.72 | 11.27 | 10.18 | 12.29 | 11.93 |
| fu_rare_j05902_m07910_p672 | – | 5063899 | – | ⋆ | – | 13.47 | 12.37 | 11.42 | 13.14 | 12.62 |

(*continued on next page*)

**Table C.16** (*continued*).

| Instance | Lower bound | Best solution | Gap% | Average deviation % | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | BRKGA | CPLEX | ILS | SA | TGA | TS-FULL | TS-RD |
| fu_rare_j06269_m14379_p672 | – | 4490070 | – | ⋆ | – | 10.30 | 8.97 | 8.24 | 9.99 | 9.70 |
| fu_rare_j06348_m07034_p672 | – | 5266269 | – | ⋆ | – | 18.59 | 17.47 | 16.79 | 18.22 | 17.31 |
| fu_rare_j06568_m07266_p672 | – | 6923072 | – | ⋆ | – | 11.76 | 10.51 | 9.90 | 11.25 | 10.90 |
| fu_rare_j06954_m17338_p672 | – | 6275920 | – | ⋆ | – | 14.21 | 12.86 | 12.38 | 13.88 | 13.56 |
| fu_rare_j07103_m10437_p672 | – | 7614057 | – | ⋆ | – | 12.12 | 11.45 | 10.59 | 11.95 | 11.87 |
| fu_rare_j08013_m15179_p672 | – | 7364871 | – | ⋆ | – | 10.04 | 8.94 | 8.53 | 9.80 | 9.56 |
| fu_rare_j08265_m09084_p672 | – | 8545100 | – | ⋆ | – | 11.19 | 9.61 | 9.42 | 10.74 | 10.41 |
| fu_rare_j09730_m08806_p672 | – | 12032213 | – | ⋆ | – | 11.62 | 10.66 | 9.98 | 11.47 | 11.19 |
| fu_rare_j10179_m10123_p672 | – | 9779919 | – | ⋆ | – | 10.26 | 9.24 | 8.87 | 9.83 | 9.75 |
| fu_rare_j11445_m23107_p672 | – | 9359791 | – | ⋆ | – | 3.59 | 3.07 | 2.33 | 3.35 | 3.35 |
| fu_rare_j16844_m05561_p672 | – | 17942992 | – | ⋆ | – | 9.35 | 8.68 | 7.80 | 9.11 | 9.01 |

## References


[1] C.E. Andrade, S.D. Byers, V. Gopalakrishnan, E. Halepovic, D.J. Poole, L.K. Tran, C.T. Volinsky, Connected cars in a cellular network: A measurement study, in: Proceedings of the 2017 Internet Measurement Conference, IMC '17, ACM, New York, NY, USA, 2017, pp. 235–241, http://dx.doi.org/10.1145/3131365.3131403.

[2] Harman, Updating Connected Car Software Over-The-Air - Why Wait? 2015, https://services.harman.com/resources/updating-connected-car-software-over-air-why-wait. (Accessed 10 September 2018).

[3] Telefonica, Connected Car Industry Report 2014, 2014. https://iot.telefonica.com/multimedia-resources/connected-car-industry-report-2014-english. (Accessed 10 September 2018).

[4] B. McKenzie, 2016 American Community Survey Content Test Evaluation Report: Journey to Work – Travel Mode of Commute and Time of Departure for Work, American Community Survey Reports, 2017. pp. 1–37, https://www.census.gov/content/dam/Census/library/working-papers/2017/acs/2017_McKenzie_01.pdf. (Accessed 06 December 2018).

[5] D. DeMuro, Buying a Car: How Long Can You Expect a Car to Last? 2015, http://www.autotrader.com/car-shopping/buying-a-car-how-long-can-you-expect-a-car-to-last-240725. (Accessed 06 December 2018).

[6] Consumer Technology Association, The Life Expectancy of Electronics, 2014. https://www.cta.tech/News/Blog/Articles/2014/September/The-Life-Expectancy-of-Electronics.aspx. (Accessed 06 December 2018).

[7] D. Newcomb, With Here OTA Connect, Over-the-Air Software Updates Finally Become Common For Cars, 2018. https://www.forbes.com/sites/dougnewcomb/2018/05/25/with-here-ota-connect-over-the-air-software-updates-finally-become-common-for-cars. (Accessed 10 September 2018).

[8] Automotive News, Over-The-Air Updates on Varied Paths, 2016. http://www.autonews.com/article/20160125/OEM06/301259980/over-the-air-updates-on-varied-paths. (Accessed 06 December 2018).

[9] Y. Zaki, Future Mobile Communications – LTE Optimization and Mobile Network Virtualization, Springer Vieweg, Wiesbaden, ISBN: 978-3-658-00807-9, 2013, http://dx.doi.org/10.1007/978-3-658-00808-6.

[10] C.E. Andrade, S.D. Byers, V. Gopalakrishnan, E. Halepovic, D.J. Poole, L.K. Tran, C.T. Volinsky, Managing massive firmware-over-the-air updates for connected cars in cellular networks, in: Proceedings of the 2nd ACM International Workshop on Smart, Autonomous, and Connected Vehicular Systems and Services, CarSys '17, ACM, 2017, pp. 65–72, http://dx.doi.org/10.1145/3131944.3131953, ISBN: 978-1-4503-5146-1.

[11] T. Kiz, RCPS with variable intensity activities and feeding precedence constraints, in: Perspectives in Modern Project Scheduling, Springer US, Boston, MA, ISBN: 978-0-387-33768-5, 2006, pp. 105–129, http://dx.doi.org/10.1007/978-0-387-33768-5_5, chap. 5.

[12] F. Della Croce, A. Grosso, F. Salassa, A matheuristic approach for the total completion time two-machines permutation flow shop problem, in: P. Merz, J.-K. Hao (Eds.), 11th European Conference Evolutionary Computation in Combinatorial Optimization, EvoCOP 2011, Springer Berlin Heidelberg, 2011, pp. 38–47, http://dx.doi.org/10.1007/978-3-642-20364-0_4.

[13] R. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Ann. Discret. Math. 5 (1979) 287–326, http://dx.doi.org/10.1016/S0167-5060(08)70356-X.

[14] J. Blazewicz, J.K. Lenstra, A.H.G. Rinnooy Kan, Scheduling subject to resource constraints: classification and complexity, Discrete Appl. Math. (ISSN: 0166-218X) 5 (1) (1983) 11–24, http://dx.doi.org/10.1016/0166-218X(83)90012-4.

[15] S. Hartmann, D. Briskorn, A survey of variants and extensions of the resource-constrained project scheduling problem, European J. Oper. Res. (ISSN: 0377-2217) 207 (1) (2010) 1–14, http://dx.doi.org/10.1016/j.ejor.2009.11.005.

[16] R.C. Leachman, A. Dtncerler, S. Kim, Resource-constrained scheduling of projects with variable-intensity activities, IIE Trans. 22 (1) (1990) 31–40, http://dx.doi.org/10.1080/07408179008964155.

[17] O. Olaitan, P. Young, J. Geraghty, Variable intensity RCPSP approach to a case study flow shop, in: Proceedings of the Conference on Summer Computer Simulation, SummerSim '15, Society for Computer Simulation International, San Diego, CA, USA, ISBN: 978-1-5108-1059-4, 2015, pp. 1–9, URL http://dl.acm.org/citation.cfm?id=2874916.2874963.

[18] M. Kong, X. Liu, J. Pei, P.M. Pardalos, N. Mladenovic, Parallel-batching scheduling with nonlinear processing times on a single and unrelated parallel machines, J. Global Optim. (ISSN: 1573-2916) (2018) 1–23, http://dx.doi.org/10.1007/s10898-018-0705-3.

[19] S. Liu, X. Liu, J. Pei, P.M. Pardalos, Q. Song, Parallel-batching machines scheduling problem with a truncated time-dependent learning effect via a hybrid CS-JADE algorithm, Optim. Methods Softw. (2019) 1–26, http://dx.doi.org/10.1080/10556788.2019.1577415.

[20] Y.-B. Woo, B.S. Kim, Matheuristic approaches for parallel machine scheduling problem with time-dependent deterioration and multiple rate-modifying activities, Comput. Oper. Res. (ISSN: 0305-0548) 95 (2018) 97–112, http://dx.doi.org/10.1016/j.cor.2018.02.017.

[21] C.A. Moreno-Camacho, J.R. Montoya-Torres, M.C. Vélez-Gallego, A comparison of mixed-integer linear programming models for workforce scheduling with position-dependent processing times, Eng. Optim. 50 (6) (2018) 917–932, http://dx.doi.org/10.1080/0305215X.2017.1358362.

[22] T. Kuschel, S. Bock, Solving the weighted capacitated planned maintenance problem and its variants, European J. Oper. Res. (ISSN: 0377-2217) 272 (3) (2019) 847–858, http://dx.doi.org/10.1016/j.ejor.2018.07.008.

[23] Y. Li, W. Zhao, J. Zhang, Resource-constrained scheduling of design changes based on simulation of change propagation process in the complex engineering design, Res. Eng. Des. (ISSN: 1435-6066) 30 (1) (2019) 21–40, http://dx.doi.org/10.1007/s00163-018-0302-y.

[24] A.E.F. Muritiba, C.D. Rodrigues, F.A. da Costa, A path-relinking algorithm for the multi-mode resource-constrained project scheduling problem, Comput. Oper. Res. (ISSN: 0305-0548) 92 (2018) 145–154, http://dx.doi.org/10.1016/j.cor.2018.01.001.

[25] Y. Ma, C. Chu, C. Zuo, A survey of scheduling with deterministic machine availability constraints, Comput. Ind. Eng. (ISSN: 0360-8352) 58 (2) (2010) 199–211, http://dx.doi.org/10.1016/j.cie.2009.04.014.

[26] P. Detti, G. Nicosia, A. Pacifici, G.Z.M. de Lara, Robust single machine scheduling with a flexible maintenance activity, Comput. Oper. Res. (ISSN: 0305-0548) 107 (2019) 19–31, http://dx.doi.org/10.1016/j.cor.2019.03.001.

[27] M. Benttaleb, F. Hnaien, F. Yalaoui, Minimising the makespan in the two-machine job shop problem under availability constraints, Int. J. Prod. Res. 57 (5) (2019) 1427–1457, http://dx.doi.org/10.1080/00207543.2018.1489160.

[28] Y. Chen, L.-H. Su, Y.-C. Tsai, S. Huang, F.-D. Chou, Scheduling jobs on a single machine with dirt cleaning consideration to minimize total completion time, IEEE Access (ISSN: 2169-3536) 7 (2019) 22290–22300, http://dx.doi.org/10.1109/ACCESS.2019.2898905.

[29] J.F. Gonçalves, J.R. de Almeida, A hybrid genetic algorithm for assembly line balancing, J. Heuristics (ISSN: 1381-1231) 8 (6) (2002) 629–642, http://dx.doi.org/10.1023/A:1020377910258.

[30] M. Ericsson, M.G.C. Resende, P.M. Pardalos, A genetic algorithm for the weight setting problem in OSPF routing, J. Comb. Optim. (ISSN: 1382-6905) 6 (3) (2002) 299–333, http://dx.doi.org/10.1023/A:1014852026591.

[31] L.S. Pessoa, C.E. Andrade, Heuristics for a flowshop scheduling problem with stepwise job objective function, European J. Oper. Res. (ISSN: 0377-2217) 266 (3) (2018) 950–962, http://dx.doi.org/10.1016/j.ejor.2017.10.045.

[32] C.E. Andrade, T. Silva, L.S. Pessoa, Minimizing flowtime in a flowshop scheduling problem with a biased random-key genetic algorithm, Expert Syst. Appl. (ISSN: 0957-4174) 128 (2019) 67–80, http://dx.doi.org/10.1016/j.eswa.2019.03.007.

[33] J.F. Gonçalves, M.G.C. Resende, A parallel multi-population genetic algorithm for a constrained two-dimensional orthogonal packing problem, J. Comb. Optim. (ISSN: 1382-6905) 22 (2) (2011) 180–201, http://dx.doi.org/10.1007/s10878-009-9282-1.

[34] C.E. Andrade, F.K. Miyazawa, M.G.C. Resende, Evolutionary algorithm for the *k*-interconnected multi-depot multi-traveling salesmen problem, in: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, GECCO'13, ACM, New York, NY, USA, ISBN: 978-1-4503-1963-8, 2013, pp. 463–470, http://dx.doi.org/10.1145/2463372.2463434.

[35] M.C. Lopes, C.E. Andrade, T.A. Queiroz, M.G.C. Resende, F.K. Miyazawa, Heuristics for a hub location-routing problem, Networks (ISSN: 1097-0037) 68 (1) (2016) 54–90, http://dx.doi.org/10.1002/net.21685.

[36] L.S. Pessoa, A.C. Santos, M.G.C. Resende, A biased random-key genetic algorithm for the tree of hubs location problem, Optim. Lett. (ISSN: 1862-4480) 11 (7) (2017) 1371–1384, http://dx.doi.org/10.1007/s11590-016-1082-9.

[37] C.E. Andrade, M.G.C. Resende, H.J. Karloff, F.K. Miyazawa, Evolutionary algorithms for overlapping correlation clustering, in: Proceedings of the 16th Conference on Genetic and Evolutionary Computation, GECCO'14, ACM, New York, NY, USA, ISBN: 978-1-4503-2662-9, 2014, pp. 405–412, http://dx.doi.org/10.1145/2576768.2598284.

[38] M. Caserta, T. Reiners, A pool-based pattern generation algorithm for logical analysis of data with automatic fine-tuning, European J. Oper. Res. (ISSN: 0377-2217) 248 (2) (2016) 593–606, http://dx.doi.org/10.1016/j.ejor.2015.05.078.

[39] C.E. Andrade, R.F. Toso, M.G.C. Resende, F.K. Miyazawa, Biased random-key genetic algorithms for the winner determination problem in combinatorial auctions, Evol. Comput. 23 (2015) 279–307, http://dx.doi.org/10.1162/EVCO_a_00138.

[40] F. Stefanello, V. Aggarwal, L.S. Buriol, J.F. Gonçalves, M.G.C. Resende, A biased random-key genetic algorithm for placement of virtual machines across geo-separated data centers, in: Proceedings of the 2015 on Genetic and Evolutionary Computation Conference, GECCO '15, ACM, New York, NY, USA, ISBN: 978-1-4503-3472-3, 2015, pp. 919–926, http://dx.doi.org/10.1145/2739480.2754768.

[41] C.E. Andrade, M.G.C. Resende, W. Zhang, R.K. Sinha, K.C. Reichmann, R.D. Doverspike, F.K. Miyazawa, A biased random-key genetic algorithm for wireless backhaul network design, Appl. Soft Comput. 33 (2015) 150–169, http://dx.doi.org/10.1016/j.asoc.2015.04.016.

[42] C.E. Andrade, S. Ahmed, G.L. Nemhauser, Y. Shao, A hybrid primal heuristic for finding feasible solutions to mixed integer programs, European J. Oper. Res. (ISSN: 0377-2217) 263 (1) (2017) 62–71, http://dx.doi.org/10.1016/j.ejor.2017.05.003.

[43] J.F. Gonçalves, M.G.C. Resende, Biased random-key genetic algorithms for combinatorial optimization, J. Heuristics (ISSN: 1381-1231) 17 (2011) 487–525, http://dx.doi.org/10.1007/s10732-010-9143-1.

[44] Y. Seddik, C. Gonzales, S. Kedad-Sidhoum, Single machine scheduling with delivery dates and cumulative payoffs, J. Sched. 16 (3) (2013) 313–329, http://dx.doi.org/10.1007/s10951-012-0302-0.

[45] T. Stützle, R. Ruiz, Iterated local search, in: Handbook of Heuristics, Springer International publishing, Cham, ISBN: 978-3-319-07124-4, 2018, pp. 579–605, http://dx.doi.org/10.1007/978-3-319-07124-4_8, chap. 15.

[46] M. den Besten, T. Stützle, Neighborhoods revisited: An experimental investigation into the effectiveness of variable neighborhood descent for scheduling, in: Proceedings of the Fourth Metaheuristics International Conference, MIC' 2001, Porto, Portugal, 2001, pp. 545–549.

[47] F. Ogbu, D. Smith, Simulated annealing for the permutation flowshop problem, Omega (ISSN: 0305-0483) 19 (1) (1991) 64–67, http://dx.doi.org/10.1016/0305-0483(91)90036-S.

[48] S.-W. Lin, K.-C. Ying, Applying a hybrid simulated annealing and tabu search approach to non-permutation flowshop scheduling problems, Int. J. Prod. Res. 47 (5) (2009) 1411–1424, http://dx.doi.org/10.1080/00207540701484939.

[49] M. Henneberg, J.S. Neufeld, A constructive algorithm and a simulated annealing approach for solving flowshop problems with missing operations, Int. J. Prod. Res. 54 (12) (2016) 3534–3550, http://dx.doi.org/10.1080/00207543.2015.1082670.

[50] B. Suman, P. Kumar, A survey of simulated annealing as a tool for single and multiobjective optimization, J. Oper. Res. Soc. 57 (10) (2006) 1143–1160, http://dx.doi.org/10.1057/palgrave.jors.2602068.

[51] F. Glover, Future paths for integer programming and links to artificial intelligence, Comput. Oper. Res. (ISSN: 0305-0548) 13 (5) (1986) 533–549, http://dx.doi.org/10.1016/0305-0548(86)90048-1.

[52] L. Piniganti, A Survey of Tabu Search in Combinatorial Optimization (Master's thesis), University of Nevada, 2014, URL https://digitalscholarship.unlv.edu/cgi/viewcontent.cgi?article=3133&context=thesesdissertations.

[53] A. Dabah, A. Bendjoudi, A. AitZai, An efficient tabu search neighborhood based on reconstruction strategy to solve the blocking job shop scheduling problem, J. Ind. Manag. Optim. (ISSN: 1547-5816) 13 (2017) 2015–2031, http://dx.doi.org/10.3934/jimo.2017029.

[54] T. Servranckx, M. Vanhoucke, A tabu search procedure for the resource-constrained project scheduling problem with alternative subgraphs, European J. Oper. Res. (ISSN: 0377-2217) 273 (3) (2019) 841–860, http://dx.doi.org/10.1016/j.ejor.2018.09.005.

[55] J.H. Holland, Adaptation in Natural and Artificial Systems: an Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence, University of Michigan press, ISBN: 0-472-08460-7, 1975.

[56] D. Whitley, S. Rana, R.B. Heckendorn, The island model genetic algorithm: On separability, population size and convergence, J. Comput. Inf. Technol. 7 (1998) 33–47.

[57] R.F. Toso, M.G.C. Resende, A C++ application programming interface for biased random-key genetic algorithms, Optim. Methods Softw. 30 (1) (2015) 81–93, http://dx.doi.org/10.1080/10556788.2014.890197.

[58] M. López-Ibáñez, J. Dubois-Lacoste, L.P. Cáceres, M. Birattari, T. Stützle, The irace package: Iterated racing for automatic algorithm configuration, Oper. Res. Perspect. (ISSN: 2214-7160) 3 (2016) 43–58, http://dx.doi.org/10.1016/j.orp.2016.09.002.