Exact and Heuristic Solution Approaches for Energy-Efficient Identical Parallel Machine Scheduling with Time-of-Use Costs

Mauro Gaggero\textsuperscript{a,1}, Massimo Paolucci\textsuperscript{b,2} and Roberto Ronco\textsuperscript{b,*,3}

\textsuperscript{a}Institute of Marine Engineering, National Research Council of Italy, Via De Marini 6, I-16149, Genoa, Italy.
\textsuperscript{b}Department of Informatics, Bioengineering, Robotics and Systems Engineering, University of Genoa, Viale Causa 13, I-16145 Genoa, Italy.

ARTICLE INFO

Keywords:
Scheduling, Energy efficiency, Fast heuristic approach, Mixed-integer programming, Time-of-Use prices.

ABSTRACT

Nowadays, energy-efficient scheduling has assumed a key role in ensuring sustainability of manufacturing processes. In this context, we focus on the problem of scheduling a set of jobs on identical parallel machines to simultaneously minimize the maximum completion time and the total energy consumption over a horizon partitioned into a set of discrete time slots. The energy costs are determined by a Time-of-Use pricing scheme, which is crucial in green manufacturing since it allows to shave the peaks of demand. First, we study the structure of the solution space of the problem. Our investigation uncovers symmetry-breaking properties that enable new insights into the problem combinatorics. As a consequence, we provide a compact mixed-integer linear programming formulation at the basis of an efficient exact solution algorithm. Furthermore, we propose novel heuristic approaches that improve state-of-the-art ones in terms of both quality of solution and required computational effort. Such heuristics also enable to speed up the exact algorithm when used for its initialization. Finally, we evaluate the effectiveness of all the proposed solution approaches in a thorough experimental campaign.

1. Introduction

In the last years, the compelling challenges in environmental sustainability have led to the development of a new paradigm for manufacturing that allows to carefully plan the production while restraining the resulting energetic expenditure. This paradigm, called energy-efficient scheduling or green scheduling [15], enables an energy-conscious approach to job scheduling in production. Among the demand-response strategies to regulate energy generation, provisioning, and consumption, Time-of-Use (TOU) pricing schemes have proven useful to flatten the peaks in customers demand to limit the resulting environmental pollution [33]. In more detail, scheduling with TOU energy costs, or prices, requires to schedule \( N \) jobs over a time horizon partitioned into a set \( \mathcal{T} \) of time slots characterized by a cost function \( c : \mathcal{T} \rightarrow \mathbb{R}_+ \) in order to minimize a measure of the total cost of the schedule [4]. One of the most considered energy-aware objective functions is the total energy cost (TEC), that is the sum of the energy prices \( c(t) \) associated with each time slot \( t \in \mathcal{T} \) where some job is processed.

In many applications, the need of minimizing energy costs may also conflict with other typical goals of scheduling problems, such as the minimization of the makespan [20, 21] or the minimization of the total weighted tardiness [11, 35]. In this paper, we consider a bi-objective scheduling problem with TOU costs, where \( N \) independent, non-preemptable jobs with no release time have to be scheduled on \( M \) identical, parallel, and single-server machines to simultaneously minimize the makespan and the TEC. From now on, we will refer to the latter as \textit{Bi-objective scheduling on identical parallel machines with Time-of-Use costs problem} (BPMSTP). The problem of minimizing the TEC with a fixed upper bound for the makespan is known to be strongly NP-hard, even on a single machine [4]. Fang et al. [10] and Chen and Zhang [4] both considered such a problem, showing that polynomial and pseudo-polynomial exact algorithms exist when the costs of time slot satisfy very specific properties [4, 10].

The main contribution of this paper is three-fold. First, we propose a novel mixed-integer linear programming (MILP) formulation for the BPMSTP by building upon the formulation in Anghinolfi et al. [1] and exploiting inherent structural symmetries to reduce the solution space. The compactness of the novel formulation enables to set up a

\*Corresponding author

ORCID(s): 0000-0002-5048-4141 (M. Gaggero); 0000-0003-0618-8406 (M. Paolucci); 0000-0001-6846-6374 (R. Ronco)

1Email: mauro.gaggero@cnr.it (Mauro Gaggero)
2Email: massimo.paolucci@unige.it (Massimo Paolucci)
3Email: roberto.ronco@edu.unige.it (Roberto Ronco)
fast exact solution algorithm for the problem. The second contribution is a heuristic approach for the BPMSTP that improves the one proposed in [1] in terms of required computational effort and quality of solutions. Lastly, we combine the aforementioned first and second contributions to provide an initial solution to the proposed exact algorithm in order to further reduce the computational effort required to find an optimal solution to the BPMSTP.

The rest of this paper is organized as follows. In Section 2, we report an overview of the literature on scheduling with TOU costs. In Section 3, we present the novel MILP formulation for the BPMSTP. In Section 4, we present the exact algorithm for the BPMSTP and the novel heuristic approach, while in Section 5 we discuss the numerical results obtained on an extensive simulation campaign. Conclusions are drawn in Section 6, by also prospecting possible future developments of our work.

2. Literature review

In the last years, energy-efficient scheduling has become a relevant topic in manufacturing due to the growing interest of industry towards environmentally-sustainable production (see, e.g., [14, 16]). The shift towards sustainable manufacturing is the result of the worldwide growth of customers demands, as well as more severe standards for environmental pollution, such as CO₂ emissions and extensive land use, leading to research and implement wiser practices to restrain the environmental footprint. Nevertheless, the importance of scheduling for the reduction of impacts in manufacturing emerged only in the last decade, as pointed out also by Fang et al. [9].

One of the first, significant works considering scheduling as a crucial element for sustainable manufacturing is Haapala et al. [18]. More recently, many different approaches for scheduling that focus on energy efficiency as a key goal have been proposed. Among the most recent contributions, we mention Karimi et al. [22] in the context of additive manufacturing, which aims at minimizing energy cost in response to time-varying electricity price and demand charge, Zhou et al. [36] for a single batch processing with dynamic job arrival times, where a hybrid multi-objective metaheuristic algorithm is proposed, and Barak et al. [3] for resource-constrained flexible manufacturing systems, where an approach based on multi-objective particle swarm optimization is adopted. The last work belongs to a series of papers using swarm intelligence and evolutionary algorithms to find a solution to the scheduling problem. According to [15], such techniques are widely used to solve scheduling problems that arise when dealing with several objectives and constraints (see, among others, [12, 24, 31]).

In the literature, several efforts in energy-efficient scheduling focus on the paradigm of TOU costs and aim at minimizing several performance indicators at the same time. In the remainder of this section, we review some of the most recent and relevant works that consider the following three aspects of scheduling problems with TOU costs: (i) the adoption of a suitable energy consumption model, (ii) the use of multi-objective models and metaheuristics, and (iii) the investigation of multi-objective approaches for the case of parallel, identical machines.

Concerning energy consumption models based on TOU costs, Ding et al. [8] present a time-indexed MILP model for the job scheduling problem of minimizing the TEC on parallel unrelated machines. The authors also propose a further approach based on a Dantzig-Wolfe decomposition algorithm for the problem. Cheng et al. [6] expand the work of Ding et al. by providing an improved MILP model that achieve higher computational performance on an experimental benchmark. Pei et al. [25] consider unrelated machines as well, but with the aim of optimizing both the TEC and a productivity measure. Specifically, the authors formulate the problem with a nonlinear programming model and propose an approximation algorithm based on a single-objective relaxation. Instead, Cheng et al. [5] consider again the simultaneous minimization of the makespan and the TEC, but they introduce a machine batch scheduling setting that also takes into account energy consumption of idle machines.

As regards multi-objective models and metaheuristics for TOU scheduling, Jiang and Wang [21] address a flexible job shop scheduling problem that requires the minimization of both the makespan and the TEC. In more detail, the authors present a MILP model and a multi-objective evolutionary algorithm based on decomposition (MOEA/D) to solve the problem. Instead, Sin and Chung [30] take into account preventive maintenance in a single machine scheduling problem with the objective of simultaneously minimizing the TEC and the largest value of machine unavailability. Similarly to Jiang and Wang, also Sin and Chung propose a MILP model together with a hybrid multi-objective genetic algorithm in order to solve largest instances of the problem. Finally, Qian et al. [27] developed a multi-objective evolutionary algorithm based on adaptive clustering to solve an identical parallel machines batch scheduling problem in order to minimize both the makespan and the TEC.

The last reference is an example of investigation of the case of identical and parallel machines, which is the one considered in this paper. Among the others studies available in the literature on this topic, Zeng et al. [34] focus on a
bi-objective scheduling problem on uniform parallel machines, which requires to minimize the TEC and the number of used machines. In more detail, the authors develop an iterative search framework for the construction of the Pareto front based on an insertion algorithm for the single-objective problem that consists in minimizing the TEC with a fixed number of machines. Qian et al. [28] investigate a batch scheduling problem on parallel machines to minimize the makespan and the TEC, and propose a multi-objective evolutionary algorithm. An adaptive clustering method is incorporated into such algorithm to mine the distribution structure of solutions, and a recombination strategy is designed to select individuals for “mating”. The “mating” probability is then adjusted according to historical information. Rocholl et al. [29] also investigate a bi-objective parallel batch machine scheduling problem, considering the fabrication of semiconductor wafers. More specifically, the authors propose three heuristics based on grouping genetic algorithms that allow choosing start times of the batches to account for the minimization of energy consumption, and they discuss a further heuristic that improves a given near-to-optimal Pareto front. Lastly, Zhou et al. [37] consider dynamic job arrivals with a TOU pricing scheme. Similarly to previous works, the goal is to simultaneously minimize the makespan and the TEC. To achieve this, the authors developed a multi-objective differential evolution algorithm, which is also able to solve large-scale instances of the problem.

3. Problem statement and mathematical formulations

In this section, we first formally describe the BPMSTP. Afterwards, we build upon the integer programming formulation presented in [1] to provide a novel formulation that exploits the inherent symmetries of the solution space.

Let $J = \{1, \ldots, N\}$ be the set of jobs, $H = \{1, \ldots, M\}$ the set of identical machines, and $T = \{1, \ldots, K\}$ the set of available time slots. Jobs are non-preemptable, and are characterized by an integer processing time $p_j \leq K$, $j \in J$, corresponding to an integer number of distinct time slots. Machines are endowed with an energy consumption rate $u_h \geq 0$, $h \in H$. Moreover, a non-negative cost $c_t \geq 0$, $t \in T$, is associated with each time slot. An assignment of a job $j \in J$ to a subset $T_j \subseteq T$ of $p_j$ consecutive time slots on machine $h \in H$ entails the processing of job $j$ during the time slots in $T_j$ by machine $h$. In this case, job $j$ is said to be scheduled in the time slots in $T_j$ on machine $h$. If no job is assigned to a given time slot, such slot is free. Then, we define a schedule

$$S = \{(j, h_j, T_j) : h_j \in H, T_j \subseteq T, \forall j \in J\}$$

as a set of the assignments of the jobs in $J$ such that each job $j \in J$ is scheduled on one and only one machine $h_j \in H$, and at most a single job in $J$ is assigned to each time slot in $T$ on each machine in $H$. If $T_j$ is a set of $p_j$ consecutive time slots, then the schedule $S$ is feasible. From now on, whenever $S$ is referred to simply as a schedule, it is implied that $S$ is feasible. Figure 1 sketches an example schedule of five jobs over two machines, where the energy costs of the various slots are explicitly reported.

The completion time $C_j(S)$ of a job $j \in J$ in schedule $S$ is the largest time slot in $T_j$, that is, $C_j(S) = \max_{t \in T_j} t$, $j \in J$. Furthermore, the makespan $C_{\text{max}}$ of a schedule $S$ is the largest among the completion times of the jobs in $J$, i.e.,

$$C_{\text{max}}(S) = \max\{C_j(S), j \in J\}.$$  \hspace{1cm} (2)

The energy cost associated with the processing of job $j$ on machine $h_j$ in $S$ is $u_{h_j} \sum_{t \in T_j} c_t$. As a consequence, the TEC of $S$ is given by

$$E(S) = \sum_{j \in J} \sum_{h \in H} u_{h_j} \sum_{t \in T_j} c_t.$$  \hspace{1cm} (3)

Then, the BPMSTP consists in finding a schedule $S$ that simultaneously minimizes (2) and (3). Since both the problem of minimizing the makespan on identical, parallel machines [26], and the problem of minimizing the TEC on a single-machine [4] are strongly NP-hard, the BPMSTP is strongly NP-hard as well. Hereinafter, since a schedule $S$ is a feasible solution to the BPMSTP, we will use the expressions “schedule” and “feasible solution” interchangeably. Moreover, to avoid burdening the notation, from now on we will omit the dependence of $C_j$, $C_{\text{max}}$ and $E$ on $S$. We also refer to the ordered tuple $I = (J, \{p_j, j \in J\}, H, \{u_h, h \in H\}, T, \{c_t, t \in T\})$ as an instance of the BPMSTP.

In the following paragraph, we report the MILP formulation of the BPMSTP provided in [1], referred to as “For-
Exact and Heuristic Solution Approaches for Energy-Efficient Scheduling

<table>
<thead>
<tr>
<th>$u_f c_i$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$j(1)$</td>
<td>$j(1)$</td>
<td>$j(3)$</td>
<td>$j(3)$</td>
<td>$j(2)$</td>
<td>$j(2)$</td>
<td>$j(2)$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>$j(5)$</td>
<td>$j(5)$</td>
<td>$j(5)$</td>
<td>$j(4)$</td>
<td>$j(4)$</td>
<td>$j(4)$</td>
<td>$j(4)$</td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: Example of a schedule of five jobs (named $j^{(1)}$, $j^{(2)}$, $j^{(3)}$, $j^{(4)}$, and $j^{(5)}$) on two machines (denoted by 1 and 2), with energy consumption rates $u_1 = 1$ and $u_2 = 2$. The 8 time slots are displayed below the corresponding energy cost.

To this end, we denote by $X_{j,h,t} \in \{0, 1\}, j \in \mathcal{J}, h \in \mathcal{H}, t \in \mathcal{T}$, a decision variable that is equal to 1 if $t$ is the start time slot of job $j$ on machine $h$, and 0 otherwise. Moreover, we express the makespan in (2) and the TEC in (3) with the decision variables $C_{\text{max}} \geq 0$ and $E \geq 0$, respectively.

Formulation 1.

$$\min \ C_{\text{max}},$$

$$\min \ E,$$

subject to

$$E = \sum_{h \in \mathcal{H}} \sum_{j \in \mathcal{J}} \sum_{t=1}^{K-p_j+1} X_{j,h,t} \left( \sum_{i=t}^{i+p_j-1} c_i \right),$$

$$\sum_{j \in \mathcal{J}} \sum_{i=\max\{1,J-p_j+1\}}^{t} X_{j,h,i} \leq 1, \ h \in \mathcal{H}, t \in \mathcal{T},$$

$$\sum_{h \in \mathcal{H}} \sum_{t=1}^{K-p_j+1} (t+p_j-1)X_{j,h,t} \leq C_{\text{max}}, \ j \in \mathcal{J},$$

$$C_{\text{max}} \leq K,$$

$$C_{\text{max}} \geq 0, \ E \geq 0, \ X_{j,h,t} \in \{0, 1\}, j \in \mathcal{J}, h \in \mathcal{H}, t \in \mathcal{T}. \tag{11}$$

The objectives (4) and (5) account for the minimization of the makespan and the TEC, respectively, according to the definition of the makespan introduced in (2) and of the TEC in (6). Constraints (7) impose that each job $j \in \mathcal{J}$ starts in a single slot on a single machine. Constraints (8) avoid that more than one job is processed in the same time slot on the same machine. The left-hand side of (9) defines the completion time of each job in $\mathcal{J}$, which must not exceed the makespan $C_{\text{max}}$. In turn, the makespan cannot be greater than the number of time slots $K$ owing to (10). Lastly, (11) defines the decision variables. Formulation 1 employs $NMK + 2$ decision variables and $2N + MK + 2$ constraints. The former number is due to the $NMK$ variables $X_{j,h,t}, j \in \mathcal{J}, h \in \mathcal{H}, t \in \mathcal{T}$, together with $C_{\text{max}}$ and $E$, while the latter one is due to constraints (6)–(10).

The main drawback of Formulation 1 lies in the number of decision variables that may become large as the size of the BPMSTP instances increases. To overcome this limitation, we present in the following a novel formulation for the BPMSTP, denoted as “Formulation 2”, that generally requires a sensibly smaller number of decision variables. Toward this end, let us first define

$$\mathcal{P} := \{d : \exists j \in \mathcal{J}, p_j = d\}$$
Exact and Heuristic Solution Approaches for Energy-Efficient Scheduling

as the set of distinct processing times of the jobs in \( \mathcal{J} \). We also define

\[
\mathcal{J}_d := \{ j : j \in \mathcal{J}, p_j = d \}, \quad d \in \mathcal{P}
\]  

(12)
as the subset of jobs with processing time equal to \( d \). We say that two feasible solutions \( S \) and \( S' \) to the BPMSTP are equivalent if they have the same value for \( C_{\text{max}} \) and \( E \). Furthermore, let \( b_{d,t} = \sum_{k=t}^{t+d-1} c_k, d \in \mathcal{P}, t = 1, \ldots, K - d + 1 \) be the cumulative cost associated with the time slots \( t, t+1, \ldots, t+d-1 \). As a consequence, any job \( j \) with processing time \( p_j = d \) assigned to machine \( h \) starting at time slot \( t \) is characterized by an energy cost equal to \( u_h b_{d,t} \). Let also \( Y_{d,h,t} \in \{0, 1\}, d \in \mathcal{P}, h \in \mathcal{H}, t \in \mathcal{T} \), be a binary decision variable that is equal to 1 if \( t \) is the first slot of a job with processing time equal to \( d \) on machine \( h \), and 0 otherwise.

**Formulation 2.**

\[
\begin{align*}
\min & \quad C_{\text{max}}, \\
\text{subject to} & \\
E &= \sum_{h \in \mathcal{H}} u_h \sum_{d \in \mathcal{P}} \sum_{t=1}^{K-d+1} b_{d,t} Y_{d,h,t}, \\
\sum_{h \in \mathcal{H}} \sum_{t=1}^{K-d+1} Y_{d,h,t} &= |\mathcal{J}_d|, \quad d \in \mathcal{P}, \\
\sum_{d \in \mathcal{P}} \sum_{i=max(1,d+1)}^{max(1,K-d+1} Y_{d,h,i} &\leq 1, \quad h \in \mathcal{H}, t \in \mathcal{T}, \\
(t + d - 1) Y_{d,h,t} &\leq C_{\text{max}}, \quad d \in \mathcal{P}, h \in \mathcal{H}, t = 1, \ldots, K - d + 1, \\
C_{\text{max}} &\leq K, \\
C_{\text{max}} &\geq 0, \quad E \geq 0, \quad Y_{d,h,t} \in \{0, 1\}, d \in \mathcal{P}, h \in \mathcal{H}, t \in \mathcal{T}.
\end{align*}
\]  

(13) – (20)

The objectives (13) and (14) account for the minimization of the makespan and the TEC, respectively, with the TEC here given by (15). Constraints (16) impose that, for each distinct processing time \( d \in \mathcal{J}_d \), exactly \( |\mathcal{J}_d| \) jobs with processing time \( d \) are assigned to some subsets of slots on the machines. Equation (17) guarantees that, on each machine, at most a single job is processed in a time slot. The left-hand side of (18) defines the completion time of jobs, which must be less than or equal to the makespan \( C_{\text{max}} \). Similarly to Formulation 1, \( C_{\text{max}} \) must not exceed the scheduling horizon \( K \), owing to (19). Lastly, (20) defines the domain of the decision variables.

The following property holds.

**Property 1.** For each feasible solution \( S \) to the BPMSTP, there are at least \( |\mathcal{J}_d|! - 1 \) other different, equivalent feasible solutions.

**Proof.** Let the schedule \( S \) be given as in (1). Let also \( \mathcal{Z} = \{ (h_j, T_j), j \in \mathcal{J} \} \) be the set of all distinct unordered pairs of machines and consecutive time slots such that there is a job \( j \in \mathcal{J} \) scheduled in the time slots in \( T_j \) on machine \( h_j \) in the schedule \( S \). We observe that \( \mathcal{Z} \) can be rewritten as \( \bigcup_{d \in \mathcal{P}} \mathcal{Z}_d \), where \( \mathcal{Z}_d = \{ (h_j, T_j), j \in \mathcal{J}_d \} \). Since all the jobs in \( \mathcal{J}_d \) require the same number \( d \) of time slots, all the possible assignments of the jobs in \( \mathcal{J}_d \) to the elements of \( \mathcal{Z}_d \), for each \( d \in \mathcal{P} \), generate schedules that are equivalent to \( S \). As the number of distinct assignments of the jobs in \( \mathcal{J}_d \) to \( \mathcal{Z}_d \) corresponds to the number \( |\mathcal{J}_d|! \) of permutations of the jobs in \( \mathcal{J}_d \), then the distinct number of assignments of the jobs in \( \mathcal{J} \) to \( \mathcal{Z} \) is the product of \( |\mathcal{J}_d|! \) for each \( d \in \mathcal{P} \). The observation that the schedule \( S \) is one of such assignments concludes the proof.

Each feasible solution to Formulation 2 defines a class of equivalent schedules. Indeed, it guarantees that, for each \( Y_{d,h,t} = 1 \), a job with processing time \( d \) is non-preemptively scheduled in the slots \( t, t+1, \ldots, t+d-1 \) on machine \( h \), but it does not specify which particular job in \( \mathcal{J} \) is assigned to such slots on \( h \).
Exact and Heuristic Solution Approaches for Energy-Efficient Scheduling

Algorithm 3.1 Generate-Schedule

**Input:** The assignment variables $Y_{d,h,t}$, $d \in P$, $h \in H$, $t \in T$

**Output:** A feasible schedule $S$

1: Let $S \leftarrow \emptyset$
2: for $d \in P$ do
3: Let $J'_d \leftarrow J_d$
4: end for
5: for $(d, h, t) \in \{(d, h, t) : Y_{d,h,t} = 1, d \in P, h \in H, t \in T\}$ do
6: Let $j \in J'_d$
7: $S \leftarrow S \cup (j, h, \{\hat{t}, \hat{t} + 1, \ldots, \hat{t} + d - 1\})$
8: $J'_d \leftarrow J'_d \setminus \{j\}$
9: end for
10: return $S$

A possible schedule out of the class of equivalent schedules defined by a solution to Formulation 2 can be generated by means of Algorithm 3.1. In more detail, such an algorithm first initializes the schedule $S$ to the empty set at line 1 and the sets needed for subsequent computations (lines 2–4). Then, for each $d$, $h$, and $t$ such that $Y_{d,h,t} = 1$, a job in $J'_d$ is assigned to $d$ consecutive slots on machine $h$ starting from slot $t$ (lines 5–9). Finally, the computed schedule $S$ is returned (line 10). At the end of the algorithm, $J'_d = \emptyset$ for each $d \in P$, all the jobs in $J$ are assigned, and there are no slots on the same machine assigned to more than one job.

Formulation 2 employs $|P| MK + 2$ decision variables and $|P| + MK + |P| \sum_{d \in P} (K - d + 1) + 2$ constraints. The former number is due to the $|P| MK$ variables $Y_{d,h,t}$, $j \in J$, $h \in H$, $t \in T$, together with $C_{\text{max}}$ and $E$, while the latter one is due to constraints (15)–(19). Let us now compare the number of variables needed by Formulation 1 and Formulation 2. The worst case for Formulation 2 occurs when $|P| = N$, i.e., when all the processing times in $J$ are distinct. In this case, Formulation 2 has the same number $NMK + 2$ of decision variables characterizing Formulation 1. On the contrary, the most convenient situation for Formulation 2 occurs when the processing times of all the jobs in $J$ are equal, i.e., when $|P| = 1$. In such case, Formulation 1 is still characterized by $NMK + 2$ decision variables, while Formulation 2 has only $MK + 2$ variables. Thus, Formulation 2 uses less decision variables than Formulation 1, except for the case $|P| = N$ when the two formulations are equivalent in terms of number of decision variables.

Let us now better characterize the worst case for the number of decision variables of Formulation 2. To this end, we observe that a necessary condition for an instance of the BPMSTP to admit at least a feasible solution is that the sum of all the time slots required by the jobs in $J$ does not exceed the overall number $MK$ of slots available for the scheduling, i.e.,

$$N \leq \sum_{j \in J} p_j \leq MK,$$

where the equality $\sum_{j \in J} p_j = N$ holds when $p_j = 1$ for each $j \in J$. We formulate the following stronger necessary condition for feasibility by building upon (21).

**Proposition 1** (Necessary condition for the existence of a solution). For a BPMSTP instance that admits at least a feasible solution, the following inequality holds:

$$|P| \leq \left\lfloor \frac{1 + \sqrt{1 + 8MK}}{2} \right\rfloor.$$  

**Proof.** First,

$$\sum_{j \in J} p_j = \sum_{d \in P} |J_d| \geq \sum_{i=1}^{|P|} i = \frac{|P|(|P| + 1)}{2}$$

and

$$\sum_{j \in J} p_j = \sum_{d \in P} |J_d| \geq \sum_{i=1}^{|P|} i = \frac{|P|(|P| + 1)}{2}$$

M. Gaggero et al.: Preprint submitted to Elsevier
since \( |J_d| \geq 1 \) and the elements in \( P \) are pairwise distinct positive integers. By combining (21) with (23), we obtain
\[
\frac{|P|(|P| + 1)}{2} \leq MK,
\]
which entails \( |P|^2 + |P| - 2MK \leq 0 \), and therefore
\[
0 \leq |P| \leq \frac{1 + \sqrt{1 + 8MK}}{2}.
\]

We observe that since (21) and Proposition 1 only depend on the parameters of the BPMSTP, they are valid for both Formulation 1 and Formulation 2. In more detail, Proposition 1 is useful to identify a larger class of unfeasible solutions with respect to (21), and therefore it enables to avoid solving several instances for Formulation 2 by simply checking the validity of (22) beforehand.

In order to illustrate how Proposition 1 allows to better describe the worst case of Formulation 2 as regards the number of decision variables, let us consider an instance with \( K = 200 \) and \( M = 10 \) as a simple example. The greatest value of \( N \) for the existence of at least a feasible solution correspond to the case \( p_j = 1 \) for all \( j \in J \), and it is equal to \( MK = 2 \cdot 10^3 \), owing to (21). In this case, the number of decision variables of Formulation 1 is \( 4 \cdot 10^6 + 2 \), whereas it is equal to \( 2 \cdot 10^3 + 2 \) for Formulation 2 since \( |P| = 1 \). Observe that, for such an instance, condition (22) also holds. Instead, if \( |P| = N \), the number of decision variables for Formulation 1 and Formulation 2 is the same. In particular, according to Proposition 1, a necessary condition for feasibility is \( N \leq \lfloor (1 + \sqrt{16001}) / 2 \rfloor = 63 \). Hence, in order for the considered instance to be possibly feasible, the number of the variables has to be no greater than \( 1.26 \cdot 10^3 + 2 \). The necessary condition (21) would instead provide the higher upper bound \( M^2K^2 + 2 = 4 \cdot 10^6 + 2 \).

To complete the comparison of Formulation 1 and Formulation 2, we also have to take into account the number and nature of the sets of constraints. However, as it will be discussed in Section 4.1, these constraints can be neglected in the framework of the developed exact solution algorithm. In particular, Formulation 2 requires a larger number of constraints than Formulation 1 (compare equations (18) and (9)). In Section 5 we will also report the significantly lower computational effort required to solve Formulation 2 with respect to Formulation 1 in all the considered experimental tests.

We conclude this section by observing that, since all the Pareto-optimal solutions of the BPMSTP are equivalent from a theoretical standpoint, practitioners may be interested in evaluating all of them and then selecting the most suitable one according to their specific needs. Thus, instead of focusing on finding a single solution \( S \), in the following sections we will develop exact and heuristic approaches to compute a set of non-dominated solutions.

4. Solution approaches

In this section, we describe the exact solution algorithm and the novel heuristic for the BPMSTP. In more detail, we describe the exact algorithm in Section 4.1. Then, we recall the heuristic presented in [1] in Section 4.2. Finally, we introduce a novel heuristic in Section 4.3. Such a solution approach builds upon the algorithmic ideas presented in Section 4.2 to further improve the quality of the solutions while increasing the computational effectiveness.

4.1. The exact algorithm

The proposed exact algorithm for the BPMSTP relies on the combination of MILP and the \( \varepsilon \)-constraint multi-objective paradigm [19]. Specifically, the algorithm iteratively exploits either Formulation 1 or Formulation 2 to compute the set of Pareto-optimal solutions for a given BPMSTP instance \( I \). Without loss of generality, in this section we describe the algorithm by only referring to Formulation 2.

The basic idea of the \( \varepsilon \)-constraint paradigm is to minimize (or maximize) one of the objectives while the other ones are constrained to be lower (or greater) than fixed values. For the considered instance \( I \) of the BPMSTP, the exact algorithm first sets an upper bound on the makespan, and then minimizes the TEC. The algorithm iterates over the previous two steps and progressively reduces the upper bound until an unfeasible solution is found. In this way, the algorithm is able to find all the points of the two-dimensional Pareto front of \( I \).
First, we observe that, for a BPMSTP instance \( I \), each Pareto-optimal solution \( S^* \) of \( I \) corresponds to a non-dominated point \( (C^{\text{max}}(S^*), E(S^*)) \) in the optimal Pareto front. In particular, there are at most \( K = \overline{K}(I) + 1 \) points in the optimal Pareto front, where

\[
\overline{K}(I) = \max \left\{ \left\lfloor \sum_{j \in J} \frac{p_j}{M} \right\rfloor, \max_{j \in J} \{p_j\} \right\}.
\]  

(24)

Indeed, since the processing times \( p_j, j \in J \), are integer numbers, \( C^{\text{max}}(S^*) \) is an integer that ranges between the lower bound \( \overline{K}(I) \) and the upper bound \( K \). However, we observe that \( \overline{K}(I) \) given by (24) is not a tight lower bound for all the instances of the BPMSTP.

**Example 1.** Let us consider a BPMSTP instance with a set of \( N = 4 \) jobs, denoted by \( j^{(1)}, j^{(2)}, j^{(3)}, \) and \( j^{(4)} \), with processing times 2, 9, 9, and 10, respectively, to be scheduled on \( M = 3 \) machines with \( u_1 = u_2 = u_3 = 1 \), and a number \( K = 11 \) of time slots. The lower bound for \( C^{\text{max}} \) given by (24) is equal to 10, but there is no feasible solution with such a makespan. In fact, jobs \( j^{(2)}, j^{(3)}, \) and \( j^{(4)} \) have to be scheduled on three different machines, without leaving two consecutive free time slots for \( j^{(1)} \).

Let us now define, for a given \( \hat{K} \) such that \( 1 \leq \hat{K} \leq K \), a downsized instance of the BPMSTP as

\[
D(\hat{K}) = (J, \{p_j, j \in J\}, \mathcal{H}, \{u_h, h \in \mathcal{H}\}, \{1, \ldots, \hat{K}\}, \{t_i, t \in \{1, \ldots, \hat{K}\}\}).
\]

(25)

A downsized instance (25) considers a subset of slots \( \{1, 2, \ldots, \hat{K}\} \subseteq \mathcal{T} \) instead of the whole set of time slots \( \mathcal{T} \). Then, we define the reduced formulation of the BPMSTP as the optimization of (14) subject to constraints (15)-(17) and (20). In other words, the reduced formulation only requires the minimization of the TEC without considering constraints (18) and (19) that are related to the makespan.

Algorithm 4.1 reports the pseudo-code of the proposed exact solution algorithm for the BPMSTP based on the aforementioned ideas. It takes a BPMSTP instance \( I \) as input, and returns the set of the Pareto-optimal solutions for \( I \) as output. The algorithm first initializes the solution set \( \mathcal{F} \) and the parameter \( \hat{K} \) at line 1. The latter is used in the downsized instances within the subsequent loop. Then, Algorithm 4.1 repeats lines 2–10 until either \( \hat{K} \) is lower than the lower bound \( \overline{K}(I) \) or an unfeasible solution is obtained before reaching \( \overline{K}(I) \). In more detail, the reduced formulation associated with \( D(\hat{K}) \) is solved at line 3. Then, if no feasible solution exists, the loop is stopped (line 5). Otherwise, Algorithm 3.1 is called to obtain a representation of the optimal solution to \( D(\hat{K}) \) as the schedule \( S^* \) (line 7). Afterwards, \( S^* \) is added to \( \mathcal{F} \) (line 8). After line 8, any solution \( S' \) to \( D(\hat{K}) \) with makespan \( C^{\text{max}}(S') \) such that \( C^{\text{max}}(S^*) \leq C^{\text{max}}(S') \leq \hat{K} \) is either equivalent to or dominated by \( S^* \). Hence, at line 9, Algorithm 4.1 updates the number of slots \( \hat{K} \) for the next iteration as \( C^{\text{max}}(S^*) - 1 \). Finally, the set of Pareto-optimal solutions \( \mathcal{F} \) is returned (line 11).

The computational efficiency to solve the reduced formulation of \( D(\hat{K}) \) with MILP at line 3 in Algorithm 4.1 can
be enhanced by providing an initial feasible solution to the MILP solver using a given heuristic. Toward this end, we propose the use of the heuristic schemes described later on in Section 4.3 to perform initialization. The computational advantages of such choice will be investigated in Section 5.

4.2. Split-greedy Heuristic and Exchange Search

In this subsection, we summarize the Split-greedy Heuristic (SGH) and Exchange Search (ES) introduced in [1] to solve the BPMSTP by describing the concepts at the foundation of the two algorithms. To this end, for a given BPMSTP instance \( I \), let us first denote a location as a pair \( l = (h, \mathcal{A}) \), where \( h \in \mathcal{H} \) and \( \mathcal{A} \) is a subset of consecutive slots in \( \mathcal{T} \). In addition, \( l \) is a free location for job \( j \) if \( |\mathcal{A}| = p_j \) and the slots in \( \mathcal{A} \) are free. Instead, \( l \) is a split-location for job \( j \) if \( \mathcal{A} \) is a set of \( p_j \) slots such that there is at least a pair of slots in \( \mathcal{A} \) that are not consecutive, and the following condition holds: for each pair of slots \( t, t' \in \mathcal{A}, t \neq t' \), either \( t \) and \( t' \) are consecutive, or \( t \) and \( t' \) are not consecutive and, for each slot \( t'' \in \mathcal{T} \) such that \( t < t'' < t' \), there is some job in \( \mathcal{J} \) assigned to \( t'' \). If all the slots in \( \mathcal{A} \) are free, then \( l \) is a free split-location. Finally, a split-schedule is a preemptive schedule where at least one job is assigned to a split-location.

The core idea of SGH is to greedily assign the jobs in \( \mathcal{J} \) to free locations or free split-locations with the smallest cost. If the resulting schedule is a split-schedule, it is converted into an equivalent feasible one. We recall that two schedules are equivalent if they have the same makespan and TEC. The pseudo-code of SGH is reported in Algorithm 4.2. Formally, the algorithm takes a BPMSTP instance \( D(\hat{K}) \) as input, with \( \hat{K} \) such that \( 1 \leq \hat{K} \leq K \), and returns a feasible schedule as output, if it exists. First, SGH initializes the schedule \( S \) as a set of empty schedules \( S_h \), one for each machine \( h \in \mathcal{H} \) (line 1). Then, it iterates over each \( d \in \mathcal{P} \) in non-increasing order (line 2), according to the well-known longest processing time first (LPT) rule [26]. For a fixed \( d \), SGH builds a list \( L_{d,h} \) of the free locations and free split-locations on machine \( h \) for any job with processing time \( d \) (line 3). Afterwards, the algorithm iterates over each \( j \in \mathcal{J}_d \) (line 4). If there are no free locations for \( j \) in \( L_{d,h} \), \( h \in \mathcal{H} \), then a feasible schedule for \( D(\hat{K}) \) does not exists, and SGH returns an empty set (line 5). Otherwise, at each iteration, a smallest cost location \( \hat{l} = (\hat{h}, \hat{\mathcal{A}}) \) is randomly selected from the locations in the lists \( L_{d,h} \), \( h \in \mathcal{H} \) (line 6). Then, \( j \) is assigned to \( \hat{l} \) (line 7), and \( L_{d,h} \) is updated by removing locations that are not free after the assignment of \( j \), and by adding the new split-locations that may have arisen from the assignment of \( j \) (line 8). Finally, if \( S \) is a split-schedule, SGH converts it into an equivalent feasible schedule (line 11). Specifically, the sequence of jobs on each machine is preserved in the converted schedule, and each job starts as soon as possible, but not earlier than its start time in the original split-schedule. Figure 2 reports an example of such conversion.

ES is a local search algorithm that takes a feasible schedule \( S \) for a BPMSTP instance \( D(\hat{K}) \) as input, and attempts to improve the TEC without worsening the makespan. The improving moves performed by ES are based on the notion of exchangeable period sequence (EPS). An EPS is a set \( \mathcal{E} \) of consecutive time slots on a machine \( h \in \mathcal{H} \) such that, if a job \( j \) is assigned to a time slot in \( \mathcal{E} \) on \( h \), then all the slots of \( j \) are in \( \mathcal{E} \) on \( h \). In particular, an EPS-J is an EPS that contains all and only the assigned slots of a job. Instead, an EPS-I is an EPS containing at least an idle slot.
Algorithm 4.3 Exchange Search (ES)

\textbf{Input:} A feasible schedule \(S\) for a downsized BPMSTP instance \(D(\hat{K})\) as in (25)

\textbf{Output:} A feasible schedule \(S'\) for \(D(\hat{K})\), with \(C_{\max}(S') \leq C_{\max}(S)\) and \(E(S') \leq E(S)\)

1: repeat
2: \hspace{1em} Let \(r \leftarrow \text{false}\)
3: \hspace{1em} for \(d \in P\) in non-increasing order do
4: \hspace{2em} for each EPS-J \(E^J\) on \(h^J\) in \(S\) such that \(|E^J| = d\) do
5: \hspace{3em} for each EPS-I \(E^I\) on \(h^I\) in \(S\) such that \(|E^I| = d\) do
6: \hspace{4em} Let \(S'\) be the schedule resulting from the EPS move involving \(E^J\) on \(h^J\) and \(E^I\) on \(h^I\) in \(S\)
7: \hspace{4em} if \(E(S') < E(S)\) then
8: \hspace{5em} Let \(S \leftarrow S'\)
9: \hspace{5em} \(r \leftarrow \text{true}\)
10: \hspace{5em} break
11: \hspace{4em} end if
12: \hspace{3em} end for
13: \hspace{1em} end for
14: \hspace{1em} until \(r\) is false
15: \hspace{1em} \(S' \leftarrow S\)
16: return \(S'\)

For a given schedule \(S\) and an EPS \(E \subseteq T\) on a machine \(h \in H\), let us denote the set of the job assignments in \(E\) on \(h\) in the schedule \(S\) as \(S_E\). Formally, \(S_E\) is the set of the assignments (\(j, h, T_j\)) \(\in S\) such that \(h_j = h\) and \(T_j \subseteq E\). Then, let \(E \subseteq T\) and \(E' \subseteq T\) be two EPSs on machines \(h \in H\) and \(h' \in H\), respectively, such that \(|E| = |E'|\). An EPS swap is an algorithm that reassigns the jobs in \(S_E\) to a subset of slots of \(E'\) on \(h'\), and the jobs in \(S_{E'}\) to a subset of slots of \(E\) on \(h\), without changing the relative assignments of the jobs. Specifically, if job \(j\) is assigned to the \(i\)-th slot of \(E\) on \(h\) before the EPS swap, then \(j\) is assigned to the \(i\)-th slot of \(E'\) on \(h'\) after the swap, and vice versa. The assignments of the jobs to the slots of an EPS \(E\) on a machine \(h\) can be changed by means of an EPS rearrangement, which is a procedure that reschedules the jobs in \(S_E\) in \(E\) on \(h\) with the goal of reducing the TEC. An EPS move combines an EPS swap with an EPS rearrangement. In particular, an EPS move involving two EPSs \(E\) and \(E'\) such that \(|E| = |E'|\) on machine \(h\) and \(h'\), respectively, first applies an EPS move of \(E\) on \(h\) and \(E'\) on \(h'\), and then an EPS rearrangement of both, separately.

The pseudo-code of ES is reported in Algorithm 4.3. For a given input schedule \(S\), the core idea of ES is to perform all the EPS moves for \(S\) that improve the TEC of \(S\), without worsening its makespan. Specifically, for each \(d \in P\) taken in non-increasing order, and for each EPS-I \(E^I\) with cardinality equal to \(d\), ES considers every EPS-I \(E^I\) with the same cardinality of \(E^J\), until it performs an improving EPS move that involves \(E^J\) and \(E^I\). Afterwards, ES proceeds with the next EPS-J (lines 6–10). At the end of the iterations (line 15), ES stops if it did not find an improving EPS move for each \(d \in J\). Otherwise, it starts over with the iterations to search for another improving EPS move.

Example 2. Figure 3 provides an example of EPS move involving the EPS-J \(E^J = \{1, 2, 3, 4, 5\}\) on machine 1, associated with job \(j^{(1)}\), and the EPS-I \(E^I = \{4, 5, 6, 7, 8\}\) on machine 2, including jobs \(j^{(4)}\) and \(j^{(5)}\) and an idle slot (see Figure 3(a)). The cost associated with \(E^J\) and \(E^I\) in Figure 3(a) is 52. Figure 3(b) shows the result of the EPS swap of \(E^I\) and \(E^J\). After the EPS swap, the cost associated with \(E^J\) and \(E^I\) increases to 56. The EPS rearrangement of \(E^I\)

\[ T \]
\begin{tabular}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
1 & (3) & (3) & (1) & (1) & (3) & (3) & (3) \\
2 & (2) & (4) & (4) & (4) & (3) & (2) & (2) \\
\end{tabular}

\[ T \]
\begin{tabular}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
1 & (3) & (3) & (3) & (1) & (1) & (1) & (3) \\
2 & (2) & (2) & (4) & (4) & (4) & (4) & (4) \\
\end{tabular}

\textbf{Figure 2:} Example of conversion of a split-schedule (a) into a feasible schedule (b). Indeed, jobs \(j^{(3)}\) and \(j^{(2)}\) in (a) are assigned to split-locations on machines 1 and 2, respectively. Instead, the jobs in (b) are feasibly scheduled.
involving jobs \( j^{(4)} \) and \( j^{(5)} \) in Figure 3(b) yields the schedule in Figure 3(c). In such schedule, the cost associated with 
\( j^{(1)}, j^{(4)} \) and \( j^{(5)} \) is finally equal to 49, i.e., it is reduced as compared to the original cost before the move (equal to 52).

Lastly, we describe the Split-greedy Scheduler (SGS) heuristic, which is used to compute a set of non-dominated solutions for an instance \( I \) of the BPMSTP by means of SGH. Similar to the exact algorithm introduced in Section 4.1, SGS exploits the \( e \)-constraint paradigm for multi-objective optimization. Specifically, it first initializes a set \( \mathcal{F} \) of heuristic solutions to an empty set. Then, it iterates over \( \hat{K} \) from the initial value \( K \) to the lower bound \( \hat{K}(I) \). At each iteration, SGS solves the BPMSTP instance \( D(\hat{K}) \) with SGH instead of using a MILP solver as in line 3 of Algorithm 4.1. If there is no feasible solution for \( D(\hat{K}) \), SGS returns an empty set. Otherwise, the set \( \mathcal{F} \) is updated by adding the solution to \( D(\hat{K}) \) and then, differently from line 9 of Algorithm 4.1, \( \hat{K} \) is decreased by 1. At the end of iterations, SGS returns the set of non-dominated solutions in \( \mathcal{F} \).

SGS may be combined with ES to improve the computed solutions. Split-greedy Scheduler with Exchange Search (SGS-ES) is the result of such combination. In more detail, SGS-ES differs from SGS since it improves the solution computed by SGH by means of ES before adding such solution to \( \mathcal{F} \).

4.3. Enhanced Heuristic Scheduler

In this section, we introduce the novel heuristic for the BPMSTP, denoted by Enhanced Heuristic Scheduler (EHS), which improves both SGH and ES while retaining the structure of SGS-ES. Such a novel heuristic is able to further improve the quality of the solutions with a lower computational burden, as showcased in Section 5.

Let us first observe that, as pointed out in Section 4.2, both the exact algorithm and SGS-ES solve a sequence of downsized instances for distinct numbers of time slots. As a result, for a given BPMSTP instance \( I \) and a positive integer \( \hat{K} < K \), the optimal solutions of the two instances \( D(\hat{K}) \) and \( D(\hat{K} + 1) \) are generally unrelated. However, the heuristic solutions to \( D(\hat{K}) \) and \( D(\hat{K} + 1) \) generated by SGS-ES share a similar structure. Indeed, many jobs in a solution to \( D(\hat{K} + 1) \) are intuitively expected to have the same assignment in a solution to \( D(\hat{K}) \), as the two instances only differ for the last time slot \( \hat{K} + 1 \), which does not exist in \( D(\hat{K}) \). Formally, let \( S' \) be a solution to \( D(\hat{K} + 1) \). Then, the assignments in \( S' \) involving jobs whose last slot is no greater than \( \hat{K} \) can be exploited to compute a solution \( S \) to \( D(\hat{K}) \). Assignments involving slot \( \hat{K} + 1 \) are instead unfeasible for \( S \), as its makespan has to be less than or equal to \( \hat{K} \). Hence, the jobs involved in such assignments have to be rescheduled in \( S \).

These observations suggest the new heuristic, hereinafter referred to as Split-greedy Heuristic with assignment history (A-SGH), that solves \( D(\hat{K}) \) by exploiting a subset of the job assignments in \( S' \), and employs SGH as a sub-routine to schedule the jobs whose assignment in \( S' \) is unfeasible for \( D(\hat{K}) \). Specifically, the idea at the core of A-SGH is to start from an initially empty schedule \( S \), and perform the following steps for each \( d \in \mathcal{P} \) considered in non-increasing order, where \( s_j(S') \) is the first assigned slot of \( j \) in the schedule \( S' \):

(i) update \( S \) with the set of job assignments \((j, h_j, T_j) \in S', j \in \mathcal{J}_d\), such that \( s_j(S') + p_j - 1 \leq \hat{K} \) and \( S \cup (j, h_j, T_j) \) is feasible;

(ii) schedule the jobs in \( \{j \in \mathcal{J}_d : s_j(S') + p_j - 1 > \hat{K} \} \) in \( S \) by means of SGH,

Step (i) updates \( S \) with all the assignments in \( S' \) for the jobs in \( \mathcal{J}_d \) that are feasible in \( S \). Then, step (ii) relies on SGH to schedule in \( S \) the jobs that are disregarded at step (i).

The general idea at the basis of A-SGH described above might also be implemented in a simpler way, summarized by the two steps (i') and (ii') described in the following. Starting from an initially empty schedule \( S \), step (i') adds to
Exact and Heuristic Solution Approaches for Energy-Efficient Scheduling

$S$ each $(j, h_j, T_j) \in S', j \in J$, such that $s_j(S') + p_j - 1 \leq \hat{K}$. Then, step (ii') considers the jobs in the assignments in $S' \setminus S$, and assigns them to the partial schedule $S$ by means of SGH. The advantage of steps (i') and (ii') with respect to the implementation that iteratively exploits steps (i) and (ii) lies in their higher computational efficiency. Indeed, at iteration $i$, $2 \leq i \leq |P|$, step (i) is unable to perform the assignments in $S'$ that, even if compliant with the maximum makespan constraint of $D(K)$, are infeasible due to the assignments performed by step (ii) during iterations $1, 2, \ldots, i - 1$. As a result, step (ii) schedules a generally higher number of jobs than step (ii'), thus burdening SGH with a more computationally demanding task. However, steps (i') and (ii') may not find a feasible solution in cases where steps (i) and (ii) are able to generate one, as highlighted by the following simple example.

Example 3. Let us consider a BPMSTP instance with three jobs $j^{(1)}$, $j^{(2)}$, and $j^{(3)}$ having processing time $1, 2$ and $1$, respectively, two machines characterized by $u_1 = u_2 = 1$, and three time slots with $c_1 = c_2 = c_3 = 1$. Let us suppose that the solution $S'$ to $D(3)$ is the one depicted in Figure 4(a). Then, steps (i') and (ii') fail in computing a solution to $D(2)$ since the assignments of jobs $j^{(1)}$ and $j^{(3)}$ in schedule $S'$ are retained in the new schedule as well, leaving no free locations or free split-locations for job $j^{(2)}$. Figure 4(b) reports such partial, unfeasible schedule. Instead, Figure 4(c) shows one of the possible feasible schedules computed by steps (i) and (ii).

The pseudo-code of A-SGH is reported at Algorithm 4.4. In more detail, A-SGH takes the BPMSTP instance $D(K)$ and the feasible solution $S'$ to the BPMSTP instance $D(K + 1)$ as inputs, and returns a feasible schedule $S$ for $D(K)$, if such a schedule exists. A-SGH first initializes $S$ (line 1) as performed by SGH (see line 1 of Algorithm 4.2), and it declares $J_d$ (line 2) according to (12). Then, A-SGH starts iterating over each $d \in P$ according to the LPT rule (line 3). As the first step in the loop, the set $R$ is initialized as an empty set (line 4). Such a set is used in the subsequent lines to keep track of the jobs involved in the assignments of $S'$ that are unfeasible for $S$. Then, A-SGH iterates over each $j \in J_d$ to schedule all the jobs in $J_d$ (lines 5–22). At each iteration, it verifies if the assignment of job $j$ in $S'$ is feasible for $S$. In this case, the assignment of $j$ in $S'$ is repeated in $S$, otherwise $j$ is added to $R$. To this end, A-SGH employs two boolean variables $\lambda$ and $\mu$ to indicate whether job $j$ would be preceded or followed, respectively, by some job in $S$ on the same machine $h'_j$ if the assignment of $j$ in $S'$ was added to $S$ as well. In more detail, $\lambda$ and $\mu$ are first initialized to false (line 6). Then, the sets $J^P$ and $J^S$ are declared as the sets of possible predecessors and successors of job $j$ on $h'_j$, respectively, if the assignment of $j$ in $S'$ was added to $S$ (lines 7–8). If $J^P$ is non-empty, A-SGH assigns the predecessor of $j$ with the greatest start time in $J^P$ to $l$ (line 10), and sets $\lambda$ to true (line 11). Similarly, if $J^S$ is non-empty, A-SGH assigns the successor of $j$ with the lowest start time in $J^S$ to $u$ (line 14), and sets $\mu$ to true (line 15). As a result of lines 9–16, $\lambda$ and $\mu$ are true if and only if there is at least a possible predecessor and a possible successor of $j$ on $h'_j$, respectively. If $j$ ends after $K$ in $S'$, or there would exist an overlapping predecessor or successor of $j$ on $h'_j$ if the assignment of $j$ in $S$ was the same as in $S'$ (line 17), then job $j$ is added to the set $R$ (line 18). Otherwise, the assignment of $j$ in $S'$ is feasible in $S$ as well, and therefore $S$ is updated accordingly (line 20). Finally, A-SGH schedules the jobs belonging to the set $R$ by means of SGH (line 23). If SGH returns an empty schedule for the jobs in $R$, then there is no feasible schedule for $D(K)$ (line 24), and A-SGH stops. Otherwise, $S$ is properly updated at line 25. Finally, A-SGH returns the computed schedule at line 27.

Next, we present the improved version of ES (Algorithm 4.3), called Exchange Search with Rescheduling (R-ES), that circumvents the following computational drawback of ES. In more detail, among all possible EPS moves, ES considers also the moves involving EPS-Is containing only free slots (see line 5 of Algorithm 4.3). For brevity, we
Algorithm 4.4 Split-greedy Heuristic with Assignment history (A-SGH)

**Input:** A downsized BPMSTP instance $D(\mathcal{K})$
A feasible schedule $S' = \{(j, h', T'_j) : h' \in \mathcal{H}, T'_j \subseteq \mathcal{T}, \forall j \in \mathcal{J}\}$ for the BPMSTP instance $D(\mathcal{K} + 1)$

**Output:** A feasible schedule $S$ for $D(\mathcal{K})$, if it exists; otherwise, an empty schedule

1. Let $S \leftarrow \{ S_h, h \in \mathcal{H} \}$ be a collection of the empty sets $S_h = \emptyset, \ h \in \mathcal{H}$
2. Let $J_d \leftarrow \{ j \in \mathcal{J} : p_j = d \}, \ d \in \mathcal{P}$
3. for $d \in \mathcal{P}$ in non-increasing order do
   4. Let $R \leftarrow \emptyset$
   5. for $j \in J_d$ do
      6. Let $\lambda, \mu \leftarrow$ false
      7. Let $J^p \leftarrow \{ q : q \in \mathcal{J}, s_q(S) \leq s_j(S'), h_q = h'_j \}$
      8. Let $J^s \leftarrow \{ r : r \in \mathcal{J}, s_r(S) > s_j(S'), h_r = h'_j \}$
      9. if $J^p \neq \emptyset$ then
         10. Let $l \leftarrow \arg\max_{q \in J^p} s_q(S)$
         11. $\lambda \leftarrow$ true
      12. end if
      13. if $J^s \neq \emptyset$ then
         14. Let $u \leftarrow \arg\min_{q \in J^p} s_q(S)$
         15. $\mu \leftarrow$ true
      16. end if
      17. if $s_j(S') + p_j - 1 > \mathcal{K}$ or ($\lambda$ and $s_l(S) + p_l - 1 \geq s_j(S')$) or ($\mu$ and $s_u(S) \leq s_j(S') + p_j - 1$) then
         18. $R \leftarrow R \cup \{ j \}$
      19. else
         20. $S_{h'_j} \leftarrow S_{h'_j} \cup (j, h'_j, T'_j)$
      21. end if
   22. end for
   23. $\hat{S} \leftarrow$ SGH($\{ (R, \{ p_j, j \in R \}, \mathcal{H}, \{ u_h, h \in \mathcal{H} \}, \{ 1, \ldots, \mathcal{K} \}, \{ c_t, t \in \{ 1, \ldots, \mathcal{K} \} \} ) \}$)
   24. if $\hat{S} = \emptyset$ then return $\emptyset$
   25. $S \leftarrow S \cup \hat{S}$
   26. end for
27. return $S$

refer to such EPS-Is as empty. Moreover, ES iterates over the EPS-Is by disregarding their energy cost. In this way, it considers the empty EPS-Is in no specific order. These observation may prospect a high computational burden, especially in the last iterations performed by ES, when the number of possible improving EPS moves is smaller as compared to the first iterations. The core idea of R-ES is to apply a local search strategy that improves a feasible input schedule $S$ by first applying ES while disregarding the empty EPS-Is, and then by performing the rescheduling of each job to consider all such empty EPS-Is. Similarly to SGH, the latter step of R-ES considers the jobs according to the LPT rule. In particular, R-ES removes a job at a time from the current schedule $S$, and greedily reinserts the same job in $S$ by assigning it to the free location or free split-location with the smallest cost in $S$.

The pseudo-code of R-ES is reported in Algorithm 4.5. In particular, R-ES takes a feasible schedule $S$ for the downsized BPMSTP instance $D(\mathcal{K})$ as input, and returns a feasible schedule $S'$ for $D(\mathcal{K})$ with $C_{\max}(S') \leq C_{\max}(S)$ and $E(S') \leq E(S)$. In fact, the purpose of R-ES is to improve the TEC of $S$, while possibly improving also the makespan as a byproduct of the performed EPS moves. However, the makespan cannot be worsened by construction, as in ES. Specifically, R-ES iteratively improves the TEC of $S$ until at least an improving move is performed (see the loop at lines 1–15). At the beginning of the loop, R-ES stores the current schedule in $S'$ (line 3). This allows to check the termination condition at the end of the loop (line 15). Then, it tries to improve $S$ by applying ES without considering the empty EPS-Is in $S$ (line 3). Next, similarly to SGH (Algorithm 4.2), R-ES iterates over the processing times of the jobs in $\mathcal{J}$ in non-increasing order (lines 4–13), by first building the list of free locations and free split-locations.
Algorithm 4.5 Exchange Search with Rescheduling (R-ES)

Input: A feasible schedule $S$ for the downsized BPMSTP instance $D(\hat{K})$ as in (25)
Output: A feasible schedule $S'$ for $D(\hat{K})$, with $C^{\text{max}}(S') \leq C^{\text{max}}(S)$ and $E(S') \leq E(S)$

1: repeat
2: Let $S' \leftarrow S$
3: Update $S$ with ES by disregarding empty EPS-Is (at line 5 of Algorithm 4.3)
4: for $d \in \mathcal{P}$ in non-increasing order do
5: Let $L_{d,h}, h \in \mathcal{H}$, be the lists of the smallest cost free locations and free split-locations on $h$ for $j : p_j = d$
6: for $j \in J_d$ do
7: $S \leftarrow S \setminus (j, h_j, T_j)$
8: if $L_{d,h} = \emptyset, \forall h \in \mathcal{H}$ then return $\emptyset$
9: Select a smallest cost location $\hat{l} = (\hat{h}, \hat{A})$ from $\bigcup_{h \in \mathcal{H}} \{l \in L_{d,h}\}$
10: Assign job $j$ to $\hat{l}$ by adding $(j, \hat{h}, \hat{A})$ to $S_{\hat{h}}$
11: Update list $L_{d,h}$
12: end for
13: end for
14: if $S$ is a split-schedule then convert it into an equivalent feasible schedule
15: until $E(S) < E(S')$
16: return $S'$

locations (line 5) as in line 3 of Algorithm 4.2. Subsequently, it iterates over each $j \in J_d$ (lines 6–12). Inside this loop, R-ES first removes the assignment $(j, h_j, T_j)$ from $S$ (line 7), and then reassigns $j$ to $S$ through lines 8–11, which are identical to lines 5–8 of Algorithm 4.2. If the resulting schedule is a split-schedule, then it is converted into a feasible one (line 14). R-ES stops the loop if it could not improve the TEC of $S$ (line 15), i.e., if $E(S) = E(S')$. Otherwise, it starts another iteration to search for further improvements. Finally, R-ES returns the resulting schedule (line 16).

Finally, we are able to describe Enhanced Heuristic Scheduler (EHS), which combines A-SGH and R-ES in a similarly way as SGS-ES does with SGH and ES. This algorithm computes a set of non-dominated solutions for an instance $I$ of the BPMSTP with a very low computational burden. Clearly, EHS benefits from the aforementioned computational improvements of A-SGH and R-ES since it applies such two algorithms sequentially. However, such a combination entails a further advantage with respect to SGS-ES. Specifically, within SGS-ES, it is useless to consider EPS swaps in ES involving an EPS-I and an empty EPS-I before the first improving EPS move since ES is preceded by SGH, i.e., a constructive greedy heuristic based on the LPT rule. Indeed, as empty EPS-Is correspond to free locations, if there was such an improving EPS swap, then the empty EPS-I involved in the swap would have been greedily chosen by SGH as a free location for the job in the involved EPS-J.

The pseudo-code of EHS is reported in Algorithm 4.6. EHS is based on the $c$-constraint paradigm, like the exact algorithm described in Section 4.1. In more detail, it first initializes the set $\mathcal{F}$ of computed solutions to an empty set, and variable $\hat{K}$ to $K$ (line 2). Then, EHS iterates over $\hat{K}$ from $K$ to the lower bound $K(I)$ (lines 3–16). At the first iteration, i.e., when $\hat{K} = K$, EHS cannot exploit any previous assignment, $S'$ is empty, and the solution $S$ for $D(K)$ is generated with SGH (line 6). Instead, for $\hat{K} = K - 1, K - 2, \ldots, K(I)$, it is possible to leverage the assignments in the feasible schedule $S'$ to compute $S$. Hence, in this case, EHS computes the solution $S$ for $D(\hat{K})$ by means of A-SGH (line 8). The loop is stopped at line 11 if $S$ is unfeasible. Otherwise, $S$ is improved via R-ES, and the resulting schedule is assigned to $S'$ (line 13). Then, the set $\mathcal{F}$ is updated with $S'$ (line 14), and $\hat{K}$ is decreased by 1 (line 15). Finally, EHS returns the set of non-dominated solutions in $\mathcal{F}$ (line 17).

5. Numerical results

In this section, we report the results of the experimental tests aimed at evaluating the performance of the solution approaches described in Section 4. Specifically, the tests are motivated by the following goals:

(a) investigate the differences between Formulation 1 and Formulation 2 when used in the exact algorithm from an experimental standpoint.
Algorithm 4.6 Enhanced Heuristic Scheduler (EHS)

**Input:** A BPMSTP instance $I$

**Output:** A set of non-dominated heuristic solutions to $I$

1. Let $K(I)$ be the lower bound defined in (24)
2. Let $\mathcal{F} \leftarrow \emptyset$, $\bar{K} \leftarrow K$, $S' \leftarrow \emptyset$
3. while $\bar{K} \geq K(I)$ do
4. Let $\mathcal{D}(\bar{K})$ be an instance as in (25)
5. if $S' = \emptyset$ then
6. $S \leftarrow \text{SGH}(\mathcal{D}(\bar{K}))$
7. else
8. $S \leftarrow \text{A-SGH}(\mathcal{D}(\bar{K}), S')$
9. end if
10. if $S$ is unfeasible then // Checks if $S = \emptyset$
11. break
12. end if
13. $S' \leftarrow \text{R-ES}(S)$
14. Set $\mathcal{F} \leftarrow \mathcal{F} \cup \{S'\}$
15. $\bar{K} \leftarrow \bar{K} - 1$
16. end while
17. return the set of non-dominated solutions in $\mathcal{F}$

(b) assess the effectiveness of EHS with respect to the state-of-the art heuristic SGS-ES;

(c) measure the speed-up achieved by the exact algorithm when provided with an initial solution computed by EHS;

(d) evaluate the performance of EHS with respect to the exact algorithm with Formulation 2. In particular, the comparison of their computational times allows to investigate the impact of the trade-off between solutions quality and computational efficiency.

The remainder of this section is organized as follows. Section 5.1 provides a description of the test instances and the algorithms implementation. Section 5.2 presents the metrics used to evaluate the performance of the algorithms. Section 5.3 focuses on goal (a), Section 5.4 deals with goal (b), while Section 5.5 addresses both goals (c) and (d).

5.1. Test instances and implementation details

The experimental tests were carried out on a set of 90 BPMSTP instances, numbered from 1 to 90. The first 60 instances were originally proposed in [32], while the last set of 30 instances were introduced in [1]. Instances 1–30, 31–60, and 61–90 are called small-scale, medium-scale, and large-scale instances, respectively. The instances differ in the values of the number of jobs $N$, the number of machines $M$, and the number of time slots $K$. In particular, the values of $N$, $M$, and $K$ for instances 1–30 are no greater than the values of $N$, $M$, and $K$, respectively, for instances 31–60. The same applies for instances 31–60 and 61–90. Appendix A provides a detailed description of all the considered 90 instances.

Concerning implementation, we used the Java 16 programming language for the exact algorithm (Algorithm 4.1), EHS, and SGS-ES. The exact algorithm was also coded with Java CPLEX 20.1.0 API, by setting the maximum optimality gap of the MILP solver to $10^{-5}$ and a time limit of 4 hours. We performed all the experimental tests on a Windows 10 system equipped with an Intel Core i9-9900K Octa-core 3.6 GHz processor and 16 GB of RAM.

For the sake of computational performance, the implementation of the exact algorithm does not generate a CPLEX representation of the mathematical formulation for $\mathcal{D}(\bar{K})$ from scratch at each iteration, as it may be suggested by the pseudo-code at line 3 of Algorithm 4.1. Instead, such a representation is generated only at the first iteration, i.e., for $\bar{K} = K$. The CPLEX representations in the subsequent iterations are obtained from the first one by adding proper constraints, which set the decision variables related to the slots that are not in $\mathcal{D}(\bar{K})$ to zero as follows:

$$X_{j,h,t} = 0, \quad j \in J, h \in H, \bar{K} - p_j + 1 < t \leq K$$
Exact and Heuristic Solution Approaches for Energy-Efficient Scheduling

for Formulation 1, and

\[ Y_{d,h,t} = 0, \quad d \in P, h \in H, \hat{K} - d + 1 < t \leq K \]

for Formulation 2. Such constraints avoid to assign jobs with completion times exceeding \( \hat{K} \). As a consequence, the makespan cannot exceed \( \hat{K} \), as desired. According to experimental evidences, the computational overhead caused by such additional constraints is lower than the one due to the generation of a new CPLEX representations at each iteration.

5.2. Performance metrics

Comparing different Pareto fronts is not straightforward, differently from solutions to single-objective optimization problems. For this reason, we employ two distinct classes of metrics to provide an in-depth analysis of the performances achieved by the proposed solution approaches. The metrics in the first class assess the quality of a Pareto front, while the ones in the second class measure the uniformity of the distribution of the non-dominated points in a front [2].

To this end, let \( F \subseteq \mathbb{R}^n \) be an \( n \)-dimensional Pareto front, and \( F^* \) be its reference optimal Pareto front. The first quality metric used in this paper is called Hypervolume \([17, 38]\), and it measures the hypervolume covered by \( F \) with respect to a reference point in the space of the objectives. As such, it can be used to compare two or more fronts by assuming a common reference point. Formally, let \( r \) be a reference point in \( \mathbb{R}^n \). Then, the Hypervolume of \( F \) is the measure of the region weakly dominated by \( F \) and bounded above by \( r \), i.e.,

\[
H(F) = \Lambda\left(\{q \in \mathbb{R}^n : \exists p \in F : p \leq q \leq r\}\right),
\]

where \( \Lambda(\cdot) \) is the Lebesgue measure. Larger Hypervolume values denote a better approximation of the optimal Pareto front.

The other quality metric used for the comparisons is called \( D1 \) \([7, 23]\), and given by

\[
D1(F, F^*) = \frac{1}{|F^*|} \sum_{y \in F^*} \min\{d_{x,y} : x \in F\}
\]

where \( d_{x,y} \) is the Euclidean distance between the two points \( x \) and \( y \) in \( F \) and \( F^* \), respectively. Smaller values of \( D1 \) denote fronts with a higher quality.

The distribution metric used in this paper is the Minimal Spacing, which is based on the Spacing metric \([2]\). Formally, let \( f_k(x), k = 1, \ldots, n, \) be the the value of the \( k \)-th objective function in \( F \) computed for some \( x \in F \). Then, the Spacing metric is given by

\[
SP(F) = \sqrt{\frac{1}{N_d(F)} \sum_{i=1}^{N_d(F)} (\delta_i - \overline{\delta})^2}, \quad \delta_i = \min_{j \neq i} \sum_{k=1}^{n} |f_k(x_i) - f_k(x_j)|,
\]

where

\[
\overline{\delta} = \frac{1}{N_d(F) - 1} \sum_{i=1}^{N_d(F) - 1} \delta_i.
\]

The smaller the values of Spacing, the better the uniformity of the distribution of the solutions in \( F \). The Minimal Spacing metric can be computed by means of an algorithmic procedure which employs the evaluation of Spacing as a subroutine. The interested reader is referred to \([2]\) for a thorough description.

5.3. Comparing the two mathematical formulations

This section compares the two mathematical formulations introduced in Section 3 when used in the exact algorithm to evaluate their computational differences. In order to perform such a comparison, the MILP solver used at line 3 of Algorithm 4.1 was first implemented with Formulation 1, and then with Formulation 2.

Table 1 reports the results of the comparison for instances 1–60. The exact algorithm was not able to solve large-scale instances 61–90 with Formulation 1 due to the memory constraints of the experimental setup. In particular,
Table 1: Comparison of the computational times obtained by the exact algorithm when using Formulation 1 (F1) and Formulation 2 (F2) for instances 1–30 and 31–60.

Table 1 shows the computational times obtained by using Formulation 1 and Formulation 2, along with their percentage deviation $\Delta_{F2}^{F1}$. Formally, given the computational times $\delta_{1,i}$ and $\delta_{2,i}$, $i = 1, 2, \ldots, 60$, obtained with Formulation 1 and Formulation 2, respectively, the value of $\Delta_{F2}^{F1}$ is given by $1 - \frac{\delta_{2,i}}{\delta_{1,i}}$. In the table, all the values of the computational times were rounded to 4 decimal places, and the best value for each instance is highlighted in bold. At the bottom of the table, the row “Avg.” reports the averages of computational times and percentage deviations on instances 1–30 and 31–60. The row “N. best” shows the number of instances where one of the implementations of the exact algorithm outperforms or equals the other. Finally, the last row displays the p-values obtained with the non-parametric Friedman test [13]. Such test was aimed at assessing the statistical significance of the difference between the computational times obtained by Formulation 1 and Formulation 2. In more detail, we assume that, if $p$ is smaller than 5%, then we can reject the null hypothesis that the two formulations generate non-significantly different results.

In the tables presented in the remainder of the section, we adopt the same conventions used in Table 1 as regards boldface highlighting, decimal rounding, as well as rows “Avg.”, “N. best”, and $p$. The percentage deviation will be only displayed for computational times. Its mathematical symbol will always report the implementation attaining the lowest computational time as a subscript, and the one achieving the higher time as a superscript.

The implementation of the exact algorithm based on Formulation 2 outperforms the one exploiting Formulation 1 in all the 60 instances, with the exception of instance 4. In particular, we observe that the average computational time of the former implementation on medium-scale instances 31–60 is two orders of magnitude lower than the average time achieved by the latter, with an average percentage deviation equal to 96.66%. Instead, the advantage of Formulation 2 in the small-scale instances, i.e., instances 1–30, is less evident, although still remarkable. In fact, the average percentage deviation is equal to 49.15%. The small values of $p$ denote the statistical significance of the achieved results.
5.4. Comparing the heuristics: EHS and SGS-ES

This section compares EHS, i.e., the proposed novel heuristic scheme, with the state-of-the-art heuristic SGS-ES. Specifically, Table 2 reports the values of the performance metrics introduced in Section 5.2 and the computational times for instances 1–30 and 31–60. Similarly, Table 3 shows the results for instances 61–90. Both tables adopt the same conventions used for Table 1. In particular, $\Delta_{\text{EHS}}^{\text{SGS-ES}}$ indicates the percentage deviation of the computational time of EHS with respect to SGS-ES. The stochasticity of the random choices performed by EHS and SGS-ES was taken into account by averaging the results obtained over 10 runs.

It turns out that EHS and SGS-ES achieve comparable results in terms of Hypervolume, D1, and Minimal Spacing for instances 1–30 and 31–60. In more detail, the performances of EHS for instances 8 and 9 are better than the ones of SGS-ES in terms of Hypervolume, but worse as concerns D1. The opposite behavior can be observed for instances 32, 36, 37, 43, and 57. These observations, along with the large $p$-values and the similar values of “N. best”, corroborate the substantial equivalence of the two heuristics in terms of performance metrics for the small-scale and medium-scale instances 1–60. As regards computational times, EHS outperforms SGS-ES on 57 instances over 60. In this case, the small values of $p$ support the relevance of such outcomes. In more detail, SGS-ES is able to find a solution in less time than EHS only for the small-scale instances 19, 21, and 25. However, in this case the percentage deviation quantifying the difference between EHS and SGS-ES is always less than 13%. Instead, EHS finds a solution in a much reduced computational time with respect to SGS-ES for all medium-scale instances, as also proved by the larger values of the percentage deviation $\Delta_{\text{EHS}}^{\text{SGS-ES}}$.

The results achieved for the large-scale instances 61–90 display the superiority of EHS with respect to SGS-ES for all the values of Hypervolume, D1, and computational times. As concerns Minimal Spacing, EHS outperforms SGS-ES in 27 instances over 30. These results showcase the capability of EHS in generating high-quality Pareto fronts, while preserving uniformity in the distribution of non-dominated points. The statistical significance of these conclusions are supported by the small values of $p$ over all performance metrics and computational times.
Table 2
Comparison of the results obtained by EHS and SGS-ES for instances 1–30 and 31–60.
5.5. The exact algorithm and EHS

This section first presents an evaluation of the computational advantage introduced by the use of EHS to provide an initial solution for the MILP solver used by the implementation of the exact algorithm (Algorithm 4.1) based on Formulation 2. Furthermore, this section compares the exact algorithm with EHS to measure the trade-off between quality of solutions and computational requirements.

Table 4 shows the values of the computational times obtained for instances 1–90 by the exact algorithm based on Formulation 2 with and without providing the solution computed by EHS to the MILP solver for initialization. The table adopts the same conventions used for Tables 1–3. In particular, \( \Delta F^2_{\text{init}} \) indicates the percentage deviation of the computational time of the exact algorithm without initialization with respect to the implementation exploiting the initial solutions provided by EHS. We observe that the times reported for the latter implementation consider the computational overhead due to the execution of EHS for initialization.

It turns out that, differently from the implementation of the exact algorithm with Formulation 1, the memory requirements of Formulation 2 enable to solve the large-scale instances 61–90 as well (we recall that Formulation 1 was able to find a solution only for the first 60 instances, as pointed out in Section 5.3). However, the maximum time limit of 4 hours for the MILP solver was not sufficient to compute the optimal Pareto fronts for instances 62, 79, 83, 85, 86, 88, and 89. On the contrary, the initialization of the exact algorithm by means of EHS allows to solve also instances 62, 79, 83, 85 and 86 within the time limit. Moreover, such an initialization enables large time savings in 87 instances over 90, with the exception of instances 88 and 89, where the time limit is again reached, and instance 15, where the use of EHS overburdens the overall time without providing a computational advantage.

With a little abuse of notation, in the remainder of this section we will simply refer as “exact algorithm with initialization” to the implementation of Algorithm 4.1 based on Formulation 2 with initialization provided by EHS. Table 5 reports the results of the comparison between the exact algorithm with initialization and EHS in terms of performance metrics introduced in Section 5.2 and computational times for instances 1–30 and 31–60. Similarly, Table 6 shows the
times are always greater for the former approach with respect to the latter, with the minimum difference equal to about than the one given by EHS, even if the values of Hypervolume and D1 differ by less than Minimal Spacing. For the other instances, the quality of results provided by the exact algorithm is always larger algorithm with initialization is always greater than except for instances 88 and 89. In fact, the exact algorithm with initialization could not compute the optimal Pareto a trade-off between quality of solutions and computational effort.

Table 4
Comparison of the computational times obtained by the exact algorithm when using Formulation 2 (F2) and Formulation 2 with initialization provided by EHS (F2-init) for instances 1–30, 31–60, and 61–90.

results of the comparison for large-scale instances 61–90. Both tables adopt the usual conventions. The percentage deviation of the computational time of EHS with respect to the exact algorithm with initialization is denoted by $\Delta F_{EHS}^{2-init}$.

Let us first focus on Table 5. It is interesting to observe that EHS computes the optimal Pareto front for instances 1, 2, 3, 4, 5, 6, 7, 17, 19, 26 and 35. In fact, EHS and the exact algorithm achieved the same values of Hypervolume, D1, and Minimal Spacing. For the other instances, the quality of results provided by the exact algorithm is always larger than the one given by EHS, even if the values of Hypervolume and D1 differ by less than $10^{-1}$. The computational times are always greater for the former approach with respect to the latter, with the minimum difference equal to about 72.77%, and an average of 97.70% and 90.84% for instances 1–30 and 31–60, respectively.

Similar arguments can be reported as regards the results for the large-scale instances 61–90 shown in Table 6. In this case, the quality of solutions of the exact algorithm with initialization is always greater than the one of EHS, except for instances 88 and 89. In fact, the exact algorithm with initialization could not compute the optimal Pareto front for such two instances within the 4 hours time limit. For the other instances, the values of Hypervolume and D1 are equal up to the second decimal place. The advantage in terms of computational requirements of EHS over the exact algorithm with initialization is always greater than 98%. Overall, we can conclude that EHS is excellent in providing a trade-off between quality of solutions and computational effort.

Finally, we observe that all the aforementioned considerations are always supported by the small $p$-values, which confirm the statistical difference between the results provided by the two approaches.
<table>
<thead>
<tr>
<th>Inst.</th>
<th>Hypervolume F2-init</th>
<th>EHS</th>
<th>D1 F2-init</th>
<th>EHS</th>
<th>Minimal Spacing F2-init</th>
<th>EHS</th>
<th>CPU time (s) F2-init</th>
<th>EHS</th>
<th>ΔF2-init (%) EHS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.6976</td>
<td>0.6976</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.1111</td>
<td>0.1111</td>
<td>0.2097</td>
<td>0.0017</td>
<td>99.21</td>
</tr>
<tr>
<td>2</td>
<td>0.7683</td>
<td>0.7683</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.1042</td>
<td>0.1042</td>
<td>0.1427</td>
<td>0.0042</td>
<td>97.08</td>
</tr>
<tr>
<td>3</td>
<td>0.7425</td>
<td>0.7425</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.1558</td>
<td>0.1558</td>
<td>0.0910</td>
<td>0.0025</td>
<td>97.31</td>
</tr>
<tr>
<td>4</td>
<td>0.7438</td>
<td>0.7438</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0977</td>
<td>0.0977</td>
<td>0.1230</td>
<td>0.0054</td>
<td>95.62</td>
</tr>
<tr>
<td>5</td>
<td>0.4531</td>
<td>0.4531</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.1830</td>
<td>0.1830</td>
<td>0.0828</td>
<td>0.0034</td>
<td>95.87</td>
</tr>
<tr>
<td>6</td>
<td>0.7532</td>
<td>0.7532</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.1486</td>
<td>0.1486</td>
<td>0.1050</td>
<td>0.0067</td>
<td>93.62</td>
</tr>
<tr>
<td>7</td>
<td>0.7460</td>
<td>0.7460</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0465</td>
<td>0.0465</td>
<td>0.3102</td>
<td>0.0050</td>
<td>98.39</td>
</tr>
<tr>
<td>8</td>
<td>0.7756</td>
<td>0.7745</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0837</td>
<td>0.0738</td>
<td>0.4047</td>
<td>0.0074</td>
<td>98.18</td>
</tr>
<tr>
<td>9</td>
<td>0.7640</td>
<td>0.7454</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0993</td>
<td>0.1018</td>
<td>0.2162</td>
<td>0.0039</td>
<td>98.22</td>
</tr>
<tr>
<td>10</td>
<td>0.7555</td>
<td>0.7553</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0944</td>
<td>0.0935</td>
<td>0.2182</td>
<td>0.0091</td>
<td>95.82</td>
</tr>
<tr>
<td>11</td>
<td>0.8019</td>
<td>0.8014</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.1101</td>
<td>0.1100</td>
<td>0.1625</td>
<td>0.0051</td>
<td>96.86</td>
</tr>
<tr>
<td>12</td>
<td>0.8247</td>
<td>0.8234</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0902</td>
<td>0.0903</td>
<td>0.3177</td>
<td>0.0097</td>
<td>96.95</td>
</tr>
<tr>
<td>13</td>
<td>0.7259</td>
<td>0.7258</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0509</td>
<td>0.0509</td>
<td>0.3035</td>
<td>0.0029</td>
<td>99.03</td>
</tr>
<tr>
<td>14</td>
<td>0.8153</td>
<td>0.8138</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0430</td>
<td>0.0426</td>
<td>0.4953</td>
<td>0.0083</td>
<td>98.32</td>
</tr>
<tr>
<td>15</td>
<td>0.7687</td>
<td>0.7556</td>
<td>0.0000</td>
<td>0.0017</td>
<td>0.0635</td>
<td>0.0607</td>
<td>0.4220</td>
<td>0.0047</td>
<td>98.89</td>
</tr>
<tr>
<td>16</td>
<td>0.7732</td>
<td>0.7723</td>
<td>0.0000</td>
<td>0.0013</td>
<td>0.0593</td>
<td>0.0593</td>
<td>0.3089</td>
<td>0.0099</td>
<td>96.78</td>
</tr>
<tr>
<td>17</td>
<td>0.8077</td>
<td>0.8077</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0724</td>
<td>0.0724</td>
<td>0.3576</td>
<td>0.0056</td>
<td>98.45</td>
</tr>
<tr>
<td>18</td>
<td>0.8506</td>
<td>0.8483</td>
<td>0.0000</td>
<td>0.0030</td>
<td>0.0680</td>
<td>0.0673</td>
<td>0.5900</td>
<td>0.0129</td>
<td>97.82</td>
</tr>
<tr>
<td>19</td>
<td>0.6900</td>
<td>0.6900</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0382</td>
<td>0.0382</td>
<td>0.2878</td>
<td>0.0040</td>
<td>98.41</td>
</tr>
<tr>
<td>20</td>
<td>0.7548</td>
<td>0.7532</td>
<td>0.0000</td>
<td>0.0016</td>
<td>0.0327</td>
<td>0.0323</td>
<td>0.4902</td>
<td>0.0095</td>
<td>98.07</td>
</tr>
<tr>
<td>21</td>
<td>0.7520</td>
<td>0.7507</td>
<td>0.0000</td>
<td>0.0012</td>
<td>0.0395</td>
<td>0.0394</td>
<td>0.4817</td>
<td>0.0060</td>
<td>98.75</td>
</tr>
<tr>
<td>22</td>
<td>0.7092</td>
<td>0.7096</td>
<td>0.0000</td>
<td>0.0003</td>
<td>0.0317</td>
<td>0.0315</td>
<td>0.7314</td>
<td>0.0136</td>
<td>97.86</td>
</tr>
<tr>
<td>23</td>
<td>0.7077</td>
<td>0.7072</td>
<td>0.0000</td>
<td>0.0006</td>
<td>0.0706</td>
<td>0.0700</td>
<td>0.2651</td>
<td>0.0070</td>
<td>97.35</td>
</tr>
<tr>
<td>24</td>
<td>0.8143</td>
<td>0.8094</td>
<td>0.0000</td>
<td>0.0061</td>
<td>0.0365</td>
<td>0.0365</td>
<td>1.1456</td>
<td>0.0140</td>
<td>98.77</td>
</tr>
<tr>
<td>25</td>
<td>0.5794</td>
<td>0.5788</td>
<td>0.0000</td>
<td>0.0005</td>
<td>0.0520</td>
<td>0.0525</td>
<td>0.1883</td>
<td>0.0048</td>
<td>97.47</td>
</tr>
<tr>
<td>26</td>
<td>0.7294</td>
<td>0.7294</td>
<td>0.0000</td>
<td>0.0000</td>
<td>0.0338</td>
<td>0.0338</td>
<td>0.4742</td>
<td>0.0077</td>
<td>98.37</td>
</tr>
<tr>
<td>27</td>
<td>0.7458</td>
<td>0.7455</td>
<td>0.0000</td>
<td>0.0004</td>
<td>0.0517</td>
<td>0.0517</td>
<td>0.3005</td>
<td>0.0066</td>
<td>97.80</td>
</tr>
<tr>
<td>28</td>
<td>0.7970</td>
<td>0.7967</td>
<td>0.0000</td>
<td>0.0008</td>
<td>0.0362</td>
<td>0.0350</td>
<td>0.7499</td>
<td>0.0117</td>
<td>98.44</td>
</tr>
<tr>
<td>29</td>
<td>0.7743</td>
<td>0.7707</td>
<td>0.0000</td>
<td>0.0003</td>
<td>0.0383</td>
<td>0.0393</td>
<td>0.7224</td>
<td>0.0065</td>
<td>99.11</td>
</tr>
<tr>
<td>30</td>
<td>0.8414</td>
<td>0.8404</td>
<td>0.0000</td>
<td>0.0017</td>
<td>0.0403</td>
<td>0.0401</td>
<td>0.8344</td>
<td>0.0180</td>
<td>97.85</td>
</tr>
</tbody>
</table>

**Table 5**
Comparison of the results obtained by the exact algorithm when using Formulation 2 with initialization provided by EHS and EHS itself for instances 1–30 and 31–60.
6. Conclusions

Energy-efficient manufacturing has become a compelling matter in the latest years owing to the pressing environmental issues and the consequent desire to shift towards a sustainable industry model. In this context, we have considered the Bi-objective scheduling on parallel identical machines with Time-of-Use costs problem (BPMSTP). We have provided a novel mathematical formulation based on an insight into the combinatorics of the BPMSTP. Such a formulation is the foundation of an exact algorithm able to compute the optimal Pareto front for the problem with a reduced computational burden as compared with previous formulations. Moreover, we have proposed a new heuristic approach for the BPMSTP, called Enhanced Heuristic Scheduler (EHS). We have shown that EHS outperformed its heuristic competitor in an experimental campaign, and it has also proved useful to increase the computational efficiency of the exact approach.

Future research directions will address the generalization of the combinatorial properties of the BPMSTP to similar problems. For instance, classical objectives in scheduling, such as the total weighted tardiness or the maximum lateness, could be considered along with the makespan or the TEC in order to increase the capability of the problem in modeling real manufacturing systems. From the computational viewpoint, we will investigate the possibility of speeding up the exact algorithm by tackling the sequential single-objective problems by means of parallel computing, so as to fully exploit the available CPU and memory resources. Finally, we will focus on extending the local search used by EHS to consider a larger sets of improving moves, which may allow to further enhance the quality of solutions at the expense of slightly increasing computational times.

<table>
<thead>
<tr>
<th>Inst.</th>
<th>Hypervolume</th>
<th>D1</th>
<th>Minimal Spacing</th>
<th>CPU time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>F2-init EHS</td>
<td>F2-init EHS</td>
<td>F2-init EHS</td>
<td>F2-init EHS</td>
</tr>
<tr>
<td>61</td>
<td>0.8223 0.8174</td>
<td>0.0000 0.0043</td>
<td>0.0237 0.0229</td>
<td>632.9495 6.2721</td>
</tr>
<tr>
<td>62</td>
<td>0.8313 0.8268</td>
<td>0.0000 0.0042</td>
<td>0.0074 0.0074</td>
<td>1163.6471 12.8846</td>
</tr>
<tr>
<td>63</td>
<td>0.7704 0.7659</td>
<td>0.0000 0.0036</td>
<td>0.0050 0.0051</td>
<td>1112.1173 8.5857</td>
</tr>
<tr>
<td>64</td>
<td>0.7886 0.7838</td>
<td>0.0000 0.0038</td>
<td>0.0055 0.0055</td>
<td>1623.1707 14.2903</td>
</tr>
<tr>
<td>65</td>
<td>0.7299 0.7215</td>
<td>0.0000 0.0039</td>
<td>0.0049 0.0049</td>
<td>1082.5546 9.7774</td>
</tr>
<tr>
<td>66</td>
<td>0.8064 0.8006</td>
<td>0.0000 0.0067</td>
<td>0.0057 0.0057</td>
<td>1793.2979 17.4887</td>
</tr>
<tr>
<td>67</td>
<td>0.7590 0.7555</td>
<td>0.0000 0.0043</td>
<td>0.0051 0.0052</td>
<td>1121.3717 8.6052</td>
</tr>
<tr>
<td>68</td>
<td>0.7446 0.7384</td>
<td>0.0000 0.0032</td>
<td>0.0053 0.0053</td>
<td>4879.9146 22.7786</td>
</tr>
<tr>
<td>69</td>
<td>0.7418 0.7365</td>
<td>0.0000 0.0048</td>
<td>0.0038 0.0038</td>
<td>2063.4526 10.2830</td>
</tr>
<tr>
<td>70</td>
<td>0.7563 0.7472</td>
<td>0.0000 0.0046</td>
<td>0.0054 0.0054</td>
<td>2359.5639 27.4660</td>
</tr>
<tr>
<td>71</td>
<td>0.8055 0.7996</td>
<td>0.0000 0.0070</td>
<td>0.0041 0.0042</td>
<td>810.7961 6.5119</td>
</tr>
<tr>
<td>72</td>
<td>0.8763 0.8719</td>
<td>0.0000 0.0048</td>
<td>0.0073 0.0073</td>
<td>722.2614 11.4006</td>
</tr>
<tr>
<td>73</td>
<td>0.7984 0.7927</td>
<td>0.0000 0.0038</td>
<td>0.0081 0.0083</td>
<td>986.6733 8.5557</td>
</tr>
<tr>
<td>74</td>
<td>0.8522 0.8479</td>
<td>0.0000 0.0046</td>
<td>0.0066 0.0067</td>
<td>1077.1803 13.9233</td>
</tr>
<tr>
<td>75</td>
<td>0.7892 0.7849</td>
<td>0.0000 0.0036</td>
<td>0.0075 0.0076</td>
<td>1061.2612 9.0737</td>
</tr>
<tr>
<td>76</td>
<td>0.8550 0.8498</td>
<td>0.0000 0.0037</td>
<td>0.0063 0.0063</td>
<td>1126.2403 16.0839</td>
</tr>
<tr>
<td>77</td>
<td>0.7514 0.7449</td>
<td>0.0000 0.0044</td>
<td>0.0069 0.0074</td>
<td>1613.4761 11.0064</td>
</tr>
<tr>
<td>78</td>
<td>0.7935 0.7878</td>
<td>0.0000 0.0052</td>
<td>0.0057 0.0056</td>
<td>2606.8366 24.8040</td>
</tr>
<tr>
<td>79</td>
<td>0.7519 0.7465</td>
<td>0.0000 0.0044</td>
<td>0.0049 0.0048</td>
<td>2166.0482 16.2491</td>
</tr>
<tr>
<td>80</td>
<td>0.8171 0.8129</td>
<td>0.0000 0.0043</td>
<td>0.0053 0.0052</td>
<td>2200.3908 22.5035</td>
</tr>
<tr>
<td>81</td>
<td>0.8235 0.8172</td>
<td>0.0000 0.0036</td>
<td>0.0053 0.0054</td>
<td>667.6995 7.7633</td>
</tr>
<tr>
<td>82</td>
<td>0.8684 0.8626</td>
<td>0.0000 0.0052</td>
<td>0.0095 0.0095</td>
<td>1407.0705 14.3503</td>
</tr>
<tr>
<td>83</td>
<td>0.8221 0.8168</td>
<td>0.0000 0.0047</td>
<td>0.0072 0.0072</td>
<td>1079.7292 10.1428</td>
</tr>
<tr>
<td>84</td>
<td>0.8314 0.8268</td>
<td>0.0000 0.0043</td>
<td>0.0073 0.0074</td>
<td>1755.0392 17.8029</td>
</tr>
<tr>
<td>85</td>
<td>0.8250 0.8207</td>
<td>0.0000 0.0035</td>
<td>0.0057 0.0057</td>
<td>1248.2436 10.9481</td>
</tr>
<tr>
<td>86</td>
<td>0.8335 0.8280</td>
<td>0.0000 0.0038</td>
<td>0.0067 0.0070</td>
<td>1769.6430 20.4612</td>
</tr>
<tr>
<td>87</td>
<td>0.8107 0.8051</td>
<td>0.0000 0.0045</td>
<td>0.0056 0.0056</td>
<td>1142.7366 12.5326</td>
</tr>
<tr>
<td>88</td>
<td>0.7039 0.8359</td>
<td>0.0000 0.0047</td>
<td>0.0070 0.0070</td>
<td>1440.0000 26.2579</td>
</tr>
<tr>
<td>89</td>
<td>0.5860 0.7613</td>
<td>0.0000 0.0024</td>
<td>0.0051 0.0059</td>
<td>1440.0000 17.7636</td>
</tr>
<tr>
<td>90</td>
<td>0.8131 0.8075</td>
<td>0.0000 0.0036</td>
<td>0.0066 0.0068</td>
<td>3079.3064 33.2349</td>
</tr>
</tbody>
</table>

Table 6
Comparison of the results obtained by the exact algorithm when using Formulation 2 with initialization provided by EHS and EHS itself for instances 61–90.

Future research directions will address the generalization of the combinatorial properties of the BPMSTP to similar problems. For instance, classical objectives in scheduling, such as the total weighted tardiness or the maximum lateness, could be considered along with the makespan or the TEC in order to increase the capability of the problem in modeling real manufacturing systems. From the computational viewpoint, we will investigate the possibility of speeding up the exact algorithm by tackling the sequential single-objective problems by means of parallel computing, so as to fully exploit the available CPU and memory resources. Finally, we will focus on extending the local search used by EHS to consider a larger sets of improving moves, which may allow to further enhance the quality of solutions at the expense of slightly increasing computational times.

M. Gaggero et al.: Preprint submitted to Elsevier
Exact and Heuristic Solution Approaches for Energy-Efficient Scheduling

Table 7

Numerical values of the number of jobs, machines, time slots, and distinct processing times of the test instances used in this paper.

<table>
<thead>
<tr>
<th>Inst.</th>
<th>Problem data</th>
<th>Inst.</th>
<th>Problem data</th>
<th>Inst.</th>
<th>Problem data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>6 3 50 4</td>
<td>31</td>
<td>30 8 100 3</td>
<td>61</td>
<td>25 25 350 12</td>
</tr>
<tr>
<td>2</td>
<td>6 3 80 4</td>
<td>32</td>
<td>60 8 100 3</td>
<td>62</td>
<td>25 25 500 12</td>
</tr>
<tr>
<td>3</td>
<td>6 5 50 4</td>
<td>33</td>
<td>100 8 100 3</td>
<td>63</td>
<td>30 25 350 12</td>
</tr>
<tr>
<td>4</td>
<td>6 5 80 3</td>
<td>34</td>
<td>150 8 100 3</td>
<td>64</td>
<td>30 25 500 12</td>
</tr>
<tr>
<td>5</td>
<td>6 7 50 5</td>
<td>35</td>
<td>200 8 100 4</td>
<td>65</td>
<td>35 25 350 12</td>
</tr>
<tr>
<td>6</td>
<td>6 7 80 3</td>
<td>36</td>
<td>30 16 100 3</td>
<td>66</td>
<td>35 25 500 12</td>
</tr>
<tr>
<td>7</td>
<td>10 3 50 5</td>
<td>37</td>
<td>60 16 100 3</td>
<td>67</td>
<td>40 25 350 12</td>
</tr>
<tr>
<td>8</td>
<td>10 3 80 5</td>
<td>38</td>
<td>100 16 100 3</td>
<td>68</td>
<td>40 25 500 12</td>
</tr>
<tr>
<td>9</td>
<td>10 5 50 4</td>
<td>39</td>
<td>150 16 100 3</td>
<td>69</td>
<td>50 25 350 12</td>
</tr>
<tr>
<td>10</td>
<td>10 5 80 5</td>
<td>40</td>
<td>200 16 100 3</td>
<td>70</td>
<td>50 25 500 12</td>
</tr>
<tr>
<td>11</td>
<td>10 7 50 5</td>
<td>41</td>
<td>30 20 100 3</td>
<td>71</td>
<td>75 25 350 12</td>
</tr>
<tr>
<td>12</td>
<td>10 7 80 4</td>
<td>42</td>
<td>60 20 100 3</td>
<td>72</td>
<td>75 25 500 12</td>
</tr>
<tr>
<td>13</td>
<td>15 3 50 5</td>
<td>43</td>
<td>100 20 100 3</td>
<td>73</td>
<td>300 30 350 12</td>
</tr>
<tr>
<td>14</td>
<td>15 3 80 5</td>
<td>44</td>
<td>150 20 100 3</td>
<td>74</td>
<td>300 30 500 12</td>
</tr>
<tr>
<td>15</td>
<td>15 5 50 4</td>
<td>45</td>
<td>200 25 100 4</td>
<td>75</td>
<td>350 30 350 12</td>
</tr>
<tr>
<td>16</td>
<td>15 5 80 5</td>
<td>46</td>
<td>30 8 300 4</td>
<td>76</td>
<td>350 30 500 12</td>
</tr>
<tr>
<td>17</td>
<td>15 7 50 5</td>
<td>47</td>
<td>60 8 300 4</td>
<td>77</td>
<td>400 30 350 12</td>
</tr>
<tr>
<td>18</td>
<td>15 7 80 5</td>
<td>48</td>
<td>100 8 300 4</td>
<td>78</td>
<td>400 30 500 12</td>
</tr>
<tr>
<td>19</td>
<td>20 3 50 5</td>
<td>49</td>
<td>150 8 300 4</td>
<td>79</td>
<td>400 30 500 12</td>
</tr>
<tr>
<td>20</td>
<td>20 3 80 5</td>
<td>50</td>
<td>200 8 300 4</td>
<td>80</td>
<td>500 30 500 12</td>
</tr>
<tr>
<td>21</td>
<td>20 5 50 5</td>
<td>51</td>
<td>30 16 300 4</td>
<td>81</td>
<td>250 40 350 12</td>
</tr>
<tr>
<td>22</td>
<td>20 5 80 5</td>
<td>52</td>
<td>60 16 300 4</td>
<td>82</td>
<td>250 40 500 12</td>
</tr>
<tr>
<td>23</td>
<td>20 7 50 5</td>
<td>53</td>
<td>100 16 300 4</td>
<td>83</td>
<td>300 40 350 12</td>
</tr>
<tr>
<td>24</td>
<td>20 7 80 5</td>
<td>54</td>
<td>150 16 300 4</td>
<td>84</td>
<td>300 40 500 12</td>
</tr>
<tr>
<td>25</td>
<td>25 3 50 5</td>
<td>55</td>
<td>200 16 300 4</td>
<td>85</td>
<td>350 40 350 12</td>
</tr>
<tr>
<td>26</td>
<td>25 3 80 5</td>
<td>56</td>
<td>30 25 300 4</td>
<td>86</td>
<td>350 40 500 12</td>
</tr>
<tr>
<td>27</td>
<td>25 5 50 5</td>
<td>57</td>
<td>60 25 300 4</td>
<td>87</td>
<td>400 40 350 12</td>
</tr>
<tr>
<td>28</td>
<td>25 5 80 5</td>
<td>58</td>
<td>100 25 300 4</td>
<td>88</td>
<td>400 40 500 12</td>
</tr>
<tr>
<td>29</td>
<td>25 7 50 5</td>
<td>59</td>
<td>150 25 300 4</td>
<td>89</td>
<td>500 40 350 12</td>
</tr>
<tr>
<td>30</td>
<td>25 7 80 5</td>
<td>60</td>
<td>200 25 300 4</td>
<td>90</td>
<td>500 40 500 12</td>
</tr>
</tbody>
</table>

Acknowledgment

The authors are in debt with Prof. Raffaele Pesenti for his useful comments on the first version of the manuscript.

A. Test instances

In this Appendix, we report the details of the set of test instances used in Section 5. For each instance, the number of jobs $N$, the number of machines $M$, and the number of time slots $K$ is given by an element $(N, M, K)$ of the Cartesian product between

- $\{6, 10, 15, 20, 25\}, \{3, 5, 7\}, \{50, 80\}$ for instances 1–30;
- $\{30, 60, 100, 150, 200\}, \{8, 16, 25\}, \{100, 300\}$ for instances 31–60, except for instances 41–44, where the number of machines is equal to 20 instead of 25;
- $\{250, 300, 350, 400, 500\}, \{25, 30, 40\}, \{350, 500\}$ for instances 61–90.

The processing times $p_{j} \in \mathbb{Z}_{+}$, for each $j$ in the set of jobs $\mathcal{J}$, were randomly drawn from the uniform distributions $U[1, 4], U[1, 4], U[1, 12]$ for instances 1–30, 31–60, and 61–90, respectively. Similarly, the consumption rates $u_{h} \in \mathbb{Z}_{+}$, for each $h$ in the set of machines $\mathcal{H}$, were drawn from $U[1, 3], U[1, 3], U[1, 6]$ for instances 1–30, 31–60, and 61–90, respectively. Finally, the time slot costs $c_{t} \in \mathbb{Z}_{+}$, for each $t$ in the set of time slots $\mathcal{T}$, belong to the sets $\{1, 2, 3, 4\}, \{1, 2, 3, 4\}, \{1, 2, \ldots , 8\}$ for instances 1–30, 31–60, and 61–90, respectively.

Table 7 shows the parameters $N, M, K$, and the number $|\mathcal{P}|$ of distinct processing time in $\mathcal{J}$ for all the instances 1–90. For the sake of compactness, the maximum processing time is not shown in the table since it is equal to $|\mathcal{P}|$ for all the instances, except for instances 1, 3, 4, 6, 9, 12 and 15, where it is equal to 5. The whole set of instances is available at https://github.com/ORresearcher/Exact-and-Heuristic-Solution-Approaches-for-Energy-Efficient-Identical-Parallel-Machine-Scheduling.
References


