European Journal of Operational Research 289 (2021) 416–434

Contents lists available at ScienceDirect

European Journal of Operational Research

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

Discrete Optimization

A bi-objective heuristic approach for green identical parallel machine scheduling

Davide Anghinolfi [a], Massimo Paolucci [b,*], Roberto Ronco [b]

[a] IROI Srl, Salita S. Viale, Genova 16128, Italy
[b] Department of Informatics, Bioengineering, Robotics and Systems Science, University of Genova, Via Opera Pia 13, Genova 16145, Italy

ARTICLE INFO

Article history:
Received 24 May 2019
Accepted 10 July 2020
Available online 15 July 2020

Keywords:
Scheduling
Identical parallel machines
Bi-objective optimization heuristics
Makespan
Total energy consumption

ABSTRACT

Sustainability in manufacturing has become a fundamental topic in the scientific literature due to the preeminent role of manufacturing industry in total world energy consumption and carbon emission. This paper tackles the multi-objective combinatorial optimization problem of scheduling jobs on multiple parallel machines, while minimizing both the makespan and the total energy consumption. The electricity prices vary according to a time-of-use policy, as in many cases of practical interest. In order to face this problem, an ad-hoc heuristic method is developed. The first part of the method, called Split-Greedy heuristic, consists in an improved and refined version of the constructive heuristic (CH) proposed in Wang, Wang, Yu, Ma and Liu (2018). The second part, called Exchange Search, is a novel local search procedure aimed at improving the quality of the Pareto optimal solutions. The experimental results prove the effectiveness of the proposed method with respect to three competitors: CH, NSGA-III, and MOEA/D.

1. Introduction

In recent years, sustainability in manufacturing industry has become a fundamental topic in the scientific literature. Consideration of aspects such as the reduction of resources consumption and carbon footprint in manufacturing production is driven by concerns for energy costs and climate changes. Such sustainability issues have increased the interest in actions aiming at improving energy efficiency in manufacturing production (Despeisse, Ball, Evans, & Levers, 2012). Strategies to pursue energy efficiency can consist in revising production processes and in investing on new equipment (Mori, Fujishima, Inamasu, & Oda, 2011). However, a more attractive possibility from the economic standpoint is to include objectives related to energy consumption in new operational management approaches (Fang, Uhan, Zhao, & Sutherland, 2013). In fact, even though the energy cost has usually been considered as an externality with respect to manufacturing production, in the last few years increasing attention has been devoted to the adoption of planning and scheduling approaches aiming at improving sustainability by reducing the energy consumption (Gahm, Denz, Dirr, & Tuma, 2016). As a consequence, the notion of energy-aware, energy-efficient or green scheduling appears often in the recent literature (e.g., Safarzadeh & Niaki, 2019; Zhang & Chiong, 2016). This paper, in particular, deals with a class of predictive scheduling problems (Pinedo, 2012). Scheduling consists in defining a detailed plan for the use of the available machines in order to perform a set of production activities (also called jobs). Scheduling is called predictive when all the relevant data are assumed known and deterministic, and the plan is defined before its actual realization in the shop floor. In general, scheduling involves three distinct decisions: assignment, i.e., dispatching the jobs to the machines providing the required kind of processing; sequencing, i.e., establishing the order of processing for the jobs assigned to each machine; timing, i.e., specifying the time instants at which the processing of each job on the assigned machine must start. Several possibilities associated with these three types of decisions can be exploited to define schedules aiming at reducing the energy consumption. If the jobs can be processed by alternative machines with different energy requirements, which may depend both on the machine and on the job, energy cost can be taken into consideration in the assignment of jobs to machines (Paolucci, Anghinolfi, & Tonelli, 2017; Safarzadeh & Niaki, 2019). Whenever the sequencing and timing introduce idle times on the machines, the possibility of turning off and on the machines can be considered (Mouzon, Yildirim, & Twomey, 2007), and the option of controlling the transition between different machine states can be taken into account as well (Che, Wu, Peng, & Yan, 2017a; May, Stahl, Taisch, & Prabhu, 2015). Energy cost reduction can also be pursued by exploiting time-of-use (TOU) electricity prices, i.e., a tariff scheme according to which

* Corresponding author.
E-mail addresses: davide.anghinolfi@iroi.it (D. Anghinolfi), massimo.paolucci@unige.it (M. Paolucci), roberto.ronco@edu.unige.it (R. Ronco).

https://doi.org/10.1016/j.ejor.2020.07.020
0377-2217/

the price of energy can vary in different time slots during a day. TOU prices favor energy cost savings by scheduling the production activities so that working loads are shifted from higher cost peak periods to cheaper off-peak ones to take advantage of lower energy prices (Che, Zhang, & Wu, 2017b; Shrouf, Ordieres-Meré, García-Sanchez, & Ortega-Mier, 2014; Wang, Wang, Yu, Ma, & Liu, 2018; Zhang, Zhao, Fang, & Sutherland, 2014). In addition, the reduction of the peak of energy demand (also called peak shaving) is considered in the timing phase (Bruzzone, Anghinolfi, Paolucci, & Tonelli, 2012). Finally, since in certain manufacturing systems the machines can work at different speeds (thus consuming energy at different rates), machine speed scaling (Fang et al., 2013; Mansouri, Aktas, & Besikci, 2016) or the possibility of controlling the processing times of the jobs (Giglio, Paolucci, & Roshani, 2017; Li, Shi, Yang, & Cheng, 2011) is also studied. Implementing an energy-efficient strategy in manufacturing scheduling requires to combine green metrics with traditional performance indicators (May et al., 2015; Taticchi, Tonelli, & Pasqualino, 2013). Thus, green scheduling problems are often formulated as bi-objective ones to model the trade-off between energy consumption or cost, and a main classic scheduling objective, such as makespan or jobs' tardiness. In particular, in this paper we propose a new effective scheduling heuristic to face the bi-objective green scheduling problem emerged from a real manufacturing context (automobile stamping die), recently proposed by Wang et al. (2018). Such a problem consists of scheduling a set of jobs on a set of identical parallel machines, considering TOU energy prices and minimizing both the makespan (the completion time of the whole set of jobs) and the total energy consumption (TEC) expressed as the total energy cost. In Wang et al. (2018) a constructive heuristic (CH) method, which is also extended with a local search procedure, is proposed and tested on two sets of problem instances (small-scale and medium- and large-scale). In Wang et al. (2018) CH is compared with NSGA-II, which was first proposed in Deb, Pratap, Agarwal, and Meyarivan (2002) and then improved in Deb and Jain (2014).

The main contributions of the this paper are briefly summarized as follows: (a) the considered problem is analyzed highlighting several theoretical properties, which can be exploited to produce solutions of better quality with respect to CH; (b) an issue affecting CH causing the production of unfeasible solutions for some instances is underlined and discussed; (c) an improved local search procedure based on the developed theoretical analysis is proposed; (d) the results obtained from an extended experimental campaign are presented and discussed. The rest of the paper is structured as follows. In Section 2, the relevant literature is reviewed. In Section 3, the considered problem is defined. In Section 4, the proposed heuristic approach, together with its theoretical foundation, is introduced. Section 5 illustrates the extended experimental analysis, and finally Section 6 draws the conclusions.

2. Literature review

We first mention some of the most significant among the most recent works on the extensive field of multi-objective scheduling; afterwards, we focus on the literature on green scheduling.

Shao, Pi, and Shao (2019) present a Pareto-based estimation distribution algorithm to simultaneously optimize the makespan and the total weight tardiness for a no-wait flowshop scheduling in a distributed factories environment. Instead, Jia, Li, Li, and Chen (2020) propose a Pareto-based ant colony optimization algorithm to minimise the makespan and the total cost for jobs rejection in a batch scheduling problem on a set of parallel batch machines with non-identical capacities. Alhadi, Kacem, Laroche, and Osman (2020) consider the multi-objective scheduling problem of minimizing the maximum lateness and the makespan on two identical machines, proposing an exact algorithm based on a dynamic programming, together with two fully polynomial time approximation schemes, to generate the complete Pareto Frontier in a pseudo-polynomial time. Meng, Rao, and Luo (2020) present a new mixed-integer programming modeling a bi-objective scheduling problem in a metal-cutting plant, including both identical and unrelated parallel machines, to minimize the total makespan and total tardiness. Finally, Queiroz and Mundim (2020) propose a multi-objective variable neighborhood descent heuristic to solve a bi-objective parallel machine scheduling problem with sequence-dependent setup times, where the makespan and the flow time are simultaneously minimized.

In recent years, the number of works dealing with the reduction of energy consumption or cost, alongside the optimization of classic objectives for manufacturing scheduling, has shown a significant increase. In this regard, two valuable surveys are due to Giret, Trentesaux, and Prabhu (2015) and Gahm et al. (2016). Several approaches for scheduling problems which consider energy efficiency as an objective in different machine environments have been proposed. For instance, for single machine problems, we can mention the works based on the optimization of the machine operating modes by Mouzon et al. (2007) and Shrouf et al. (2014). In flow shops, Zhang et al. (2014) consider TOU energy prices, while Mansouri et al. (2016) exploit the different energy consumption levels associated with the variable speeds of machines. Among the most recent papers, both Wang et al. (2018) and Safarzadeh and Niaki (2019) consider parallel machine environments and provide a quite extended literature review for this class of problems. The approaches proposed for energy efficient parallel machine scheduling are based on multi-objective optimization techniques and algorithms, and exploit some specific characteristics of the manufacturing production context under concern. Energy consumption is taken into account together with the minimization of the makespan (Che et al., 2017b; Cota et al., 2019; Ji, Wang, & Lee, 2013; Li et al., 2011; Li, Zhang, Leung, & Yang, 2016; Safarzadeh & Niaki, 2019; Wang et al., 2018; Wang, Wang, & Lin, 2017; Yeh, Chuang, & Lee, 2015), the total weighed job tardiness (Fang & Lin, 2013), the total completion time (Ding, Song, Zhang, Chiong, & Wu, 2016) and the number of used machines (Zeng, Che, & Wu, 2018). Among these works, parallel machine scheduling problems under the TOU tariffs are considered in Che et al. (2017b), Zeng et al. (2018), Wang et al. (2018). Several approaches model the involved bi-objective problem using a technique similar to the ϵ-constraint method, i.e., they solve a scalar optimization problem minimizing a single objective having constrained the other objective not to exceed a fixed upper bound. This is the strategy used in Li et al. (2011), Ji et al. (2013), Yeh et al. (2015), Li et al. (2016), where an upper bound for the consumed energy is fixed. On the other hand, solving a multi-objective optimization problems consists in determining the set of efficient or Pareto optimal solutions (Branke, Deb, Miettinen, & Slowiński, 2008). A solution x is Pareto optimal if it is not dominated by a different solution, i.e., it does not exist a solution x' whose objective values are all not worse than the ones of x and at least one objective value is better than the one of x. As solving multi-objective combinatorial problem is very often NP-hard, the approaches proposed in the literature aim at identifying a good approximation of the set of efficient solutions (also called Pareto front). In Wang et al. (2018), as well as in the present paper, the ϵ-constraint method is adopted. In both approaches, the upper bound for the constrained objective (the makespan) is systematically varied in order to produce an estimation of the Pareto front. Other green scheduling works exploit multi-objective metaheuristics, like genetic algorithms (May et al. (2015); Liu, Dong, Lohse, and Petrovic (2016)), hybrid backtracking search algorithms (Lu, Gao, Li, Pan, & Wang, 2017), particle swarm optimization (Tang, Dai, Salido, & Giret, 2016), or hybrid grey wolf discrete optimization algorithms (Lu, Gao, Li, Zheng, & Gong, 2018).

3. Problem statement

The considered problem consists in scheduling N independent jobs on M identical parallel machines. All jobs are available for processing (no release time is considered), and each machine can process only a job at a time. Let $J = \{1, \ldots, N\}$ be the set of jobs, and $H = \{1, \ldots, M\}$ be the set of machines. For each job $j \in J$, the processing time p_j is identical for all the machines and it is assumed equal to an integer number of time slots. The jobs must be scheduled in a time horizon consisting of a set $T = \{1, \ldots, K\}$ of time slots. Jobs cannot be preempted, i.e., each job j must be processed in a set S_j of p_j consecutive time slots on a single machine $h \in H$. Due to the TOU energy costs, the time slots in T are grouped into time intervals, each including a sequence of time slots characterized by the same energy price. Namely, a given energy price c_t is assigned to each time slot $t \in T$. Each machine $h \in H$ is associated with a fixed energy consumption rate e_h, and the energy consumption cost associated with the processing of job j on h is $e_h \cdot \sum_{t \in S_j} c_t$. The completion time C_j of a job j is the last time slot assigned to such job. Furthermore, the makespan C^{max} of a schedule is the maximum among the completion times of the jobs in J, i.e., $C^{max} = max\{C_j : j \in J\}$. The value of TEC associated with a schedule is defined as $\sum_{h \in H} e_h \sum_{j \in J_h} \sum_{k \in S_j} c_k$, where J_h is the set of jobs scheduled on machine h. Finally, the problem is to assign and sequence the jobs on the machines, determining the starting time of each job while minimizing C^{max} and TEC at the same time. This problem can be formulated as a mixed integer programming problem as follows.

Variables.

- $X_{jht} \in \{0, 1\}$, $j \in J$, $h \in H$, $t \in T$, equal to 1 if job j is processed on machine h starting in slot t, 0 otherwise;
- $C_j \geq 0$, $j \in J$, completion time of job j;
- $C^{max} \geq 0$ makespan;
- $TEC \geq 0$ total energy consumption.

Formulation.

$$min\ C^{max} \tag{1}$$

$$min\ TEC \tag{2}$$

subject to

$$TEC = \sum_{h \in H} e_h \sum_{j \in I} \sum_{t=1}^{K-p_j+1} X_{jht} \left(\sum_{i=t}^{t+p_j-1} c_i \right) \tag{3}$$

$$\sum_{h \in H} \sum_{t=1}^{K-p_j+1} X_{jht} = 1 \quad j \in J \tag{4}$$

$$\sum_{j \in J} \sum_{i=max\{0, t-p_j+1\}}^{t} X_{jhi} \leq 1 \quad t \in T, h \in H \tag{5}$$

$$C_j = \sum_{h \in H} \sum_{t=1}^{K-p_j+1} (t + p_j - 1) X_{jht} \quad j \in J \tag{6}$$

$$C^{max} \geq C_j \quad j \in J \tag{7}$$

$$C^{max} \leq K \tag{8}$$

$$TEC \geq 0, C^{max} \geq 0, C_j \geq 0 \quad j \in J \tag{9}$$

$$X_{jht} \in \{0, 1\} \quad j \in J, h \in H, t \in T \tag{10}$$

Makespan and TEC are the objective functions minimized in (1) and (2), respectively. Eq. (3) defines TEC. Constraints (4) impose that each job is assigned to a single machine, ensuring that the processing of any job starts in a single slot on a single machine. Constraints (5) guarantee that at most a single job is processed in each time slot on each machine. Eqs. (6) provide the completion time of the jobs, implicitly imposing the non-preemption. Constraints (7) define the makespan, whereas (8) imposes that the makespan must not exceed the available time horizon; note that, due to (8), the problem may not be solvable if K is not sufficiently large. Finally, (9) and (10) give the variable definitions. It is worth noting that the above model differs from the one in Wang et al. (2018), since the latter does not guarantee the non-overlapping of jobs for some instances of the problem.

4. The scheduling approach

In this section, a new scheduling algorithm, called Split-Greedy Scheduler (SGS), is introduced. Such algorithm is based on a new heuristic that we called Split-Greedy Heuristic (SGH), designed in order to enhance and refine CH (Wang et al., 2018). A novel ad-hoc local search, called Exchange Search (ES), is also described. The scheduling algorithm which employs SGH for finding solutions, and ES to improve them, is called SGS-ES. Both SGS and SGS-ES are detailed in Sections 4.1 and 4.2. Moreover, the improvements that our approach entails with respect to CH (Wang et al., 2018) are discussed in depth.

4.1. The Split-Greedy Heuristic

First, it is necessary to introduce some definitions regarding time slots and jobs.

Definition 4.1. Two slots t and $t + k$ on the same machine are said free-consecutive if they are both free and all the slots in $[t + 1, t + k - 1]$ are occupied.

We call *location* a pair $l = (h, F)$, with $h \in H$ and $F \subseteq T$. We also denote as EC_l the energy cost $e_h \cdot \Sigma_{t \in F} c_t$ of l.

Definition 4.2. A free location for a job j is a location (h, F) such that F contains only free adjacent and/or free-consecutive slots on machine h, and $|F| = p_j$.

In Fig. 4.1 we report an example of 6 slots on a machine h where a single job j with $p_j = 2$ is scheduled. Slots 2 and 5 are free-consecutive slots on h. Considering a job j' with $p_{j'} = 2$, then the location (h, F), with $F = \{2, 5\}$, is a free location for j' on h. Note that assigning this location to j' does not produce a feasible schedule since j' is preempted.

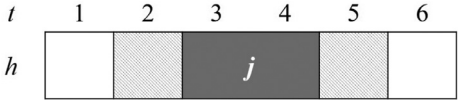

Fig. 4.1. An example of free-consecutive slots.

Definition 4.3. The assigned location of a job j is a location (h, A_j), where A_j is the set of adjacent slots assigned to j and $|A_j| = p_j$.

If a job j has an assigned location (h, A_j), its start time s_j is $min_{t \in A_j}\{t\}$. Note that, in the example of Fig. 4.1, $(h, \{3, 4\})$ is the location assigned to job j.

Definition 4.4. A feasible schedule $S = \{(h, A_j), j \in J : h \in H, A_j \subseteq T\}$ is a set of assigned locations such that for all (h, A_j) and $(h, A_{j'})$, $A_j \cap A_{j'} = \emptyset$ if $j \neq j'$.

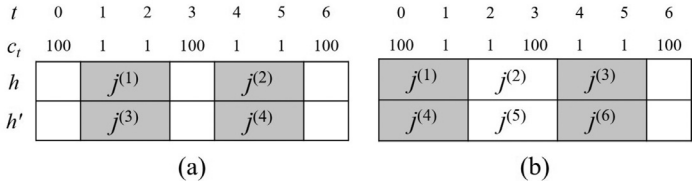

Fig. 4.2. (a) the partial schedule generated by CH for Example 4.1; (b) a feasible schedule for the same example.

Definition 4.4 states that in a feasible schedule, each job is assigned to a set of adjacent slots on a machine, and that the non-overlapping condition for the jobs assigned to the same machine is satisfied, i.e., any two assigned locations on the same machine must have no common time slots. If (h, A_j) is an assigned location, with $j \in J$, $A_j \subseteq T$, and such location satisfies the non-overlapping condition on h, then j is said *feasibly scheduled*.

In order to generate the best approximation of the Pareto front, both CH and SGS iterate a scheduling construction procedure. For the sake of clarity and completeness, CH is briefly summarized in the following. Note that we also include the local search procedure (step 4) introduced in Wang et al. (2018), since this whole algorithm is referred as CH in the experimental tests reported in Wang et al. (2018).

1. The jobs are sorted according to the longest processing time (LPT) rule in non-increasing order of processing times;
2. The lower bound for the makespan is defined as $K^{min} = \sum_{j \in J} p_j / M$, the current maximum accepted makespan $\hat{K}$ is fixed to the time horizon K of the problem, and the set of found solutions N_s is initialized to $\emptyset$;
3. For each job j, among the set L of free locations for j, which includes only free adjacent slots, l^{min} such that $EC_{l^{min}} = min\{EC_l, l \in L\}$ is determined and assigned to j (breaking ties by choosing the earliest start time among free locations with equal energy consumption cost);
4. A local search that tries to improve TEC without worsening the makespan by shifting blocks of contiguous jobs on each machine is applied;
5. The makespan C^{max} and TEC of the obtained solution are computed, the solution is added to N_s, the current accepted makespan is updated as $\hat{K} = C^{max} - 1$, and the algorithm is iterated to step 3 provided that $\hat{K} \geq K^{min}$;
6. The set of the non-dominated solutions of N_s is returned.

CH(201

Since no information is given about the strategy used to break ties in step 1, we assume that they are randomly broken. The first improvement introduced by SGS is the use of a more appropriate lower bound for the maximum accepted makespan, i.e., $K^{min} = max\{\lfloor \sum_{j \in J} p_j / M \rfloor, max_{j \in J}\{p_j\}\}$. For each fixed $\hat{K}$, both CH and SGS iteratively construct a solution by assigning each job to one of the available locations, which, in the case of CH, corresponds only to free adjacent slots. However, from the description presented in Wang et al. (2018), CH may not be able to build feasible schedules, even if a feasible schedule exists, when no location including only free adjacent slots is available for a job due to the previous assignments. This behaviour is highlighted in Example 4.1.

Example 4.1. Consider machines h and h', with unitary energy consumption $e_h = 1$ and $e_{h'} = 2$, and a set $J = \{j^{(i)}, i = 1, .., 6\}$ of jobs, with $p_j = 2$ for each $j \in J$. The time horizon K is 7, and the energy prices of each time interval are 100, 1, 1, 100, 1, 1, 100, as shown in Fig. 4.2. Since all the jobs have the same processing time, CH considers the jobs in J in a random order. For simplicity, in Fig. 4.2 we assume that such jobs are sorted according to the non-decreasing order of their indices. Then, CH assigns the first four jobs $j^{(1)}$, $j^{(2)}$, $j^{(3)}$ and $j^{(4)}$ to locations $(h, \{1, 2\})$, $(h, \{4, 5\})$, $(h', \{1, 2\})$ and $(h', \{4, 5\})$ respectively, since such locations have the same energy cost. As an example, Fig. 4.2(a) highlights one among the possible alternative assignments of these jobs. However, Fig. 4.2(a) also shows that there is no valid location left for $j^{(5)}$ and $j^{(6)}$. Hence, since CH does not handle such kind of situations, it returns a partial (i.e., not feasible) schedule. However, a feasible schedule exists for this example, since the available slots on the two machines are sufficient to process all the 6 jobs with a different assignment, as shown in Fig. 4.2(b).

In order to overcome this issue, SGS can temporarily assign the jobs to free locations that, according to Definition 4.2, may include free-consecutive slots. Then, it can produce feasible schedules by exploiting an approach based on the theoretic framework described in the following.

abordagem para solu

Definition 4.5. A split-location is a location including at least two free-consecutive slots. A split-location is said free for a job j if it is also a free location for j. A split-location (h, A) is denoted as assigned if there exists a job $j \in J$ such that all the slots in A are assigned to j.

A job assigned to a split-location l is said *split-scheduled* in l. Note that the set of all the free locations for a job j contains the set of all the free split-locations for such job. Fig. 4.3 shows an example of a split-location assigned to job j, where slots 1 and 2 are adjacent, while 2 and 6 are free-consecutive slots. In addition, Fig. 4.3 features a split-schedule, which is defined in the following.

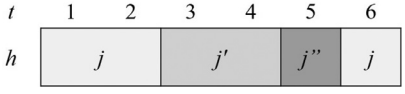

Fig. 4.3. An example of assigned split-location of a job j.

Definition 4.6. A split-schedule $S = \{(h, I_j), j \in J : h \in H, I_j \subseteq T\}$ is a set of assigned locations and split-locations such that, for all (h, I_i) and (h, I_j) assigned to $i, j \in J$, then $I_i \cap I_j = \emptyset$ if $i \neq j$.

Compared to Definition 4.4, Definition 4.6 states that in a split-schedule at least one job is assigned to a split-location. Hence, split-schedules are not feasible and they are a subset of preemptive schedules. Fig. 4.4 highlights the difference between these two types of schedules. Fig. 4.4(a) shows a split-schedule with a split-scheduled job $j^{(1)}$, whereas Fig. 4.4(b) shows a schedule with a preempted job $j^{(2)}$; such job is not split-scheduled since slots 2 and 6 are not free-consecutive because slot 5 is free.

Definition 4.7. A feasible schedule (or split-schedule) S is said equivalent to a feasible schedule (or split-schedule) S' if $C^{max}(S) = C^{max}(S')$ and $TEC(S) = TEC(S')$.

Definition 4.8. A split-schedule block B' of a split-schedule S is a set of assigned locations and split-locations (h, G_j) on a machine h for a subset of jobs $j \in J' \subseteq J$, such that $\bigcup_{j \in J'} G_j$ corresponds to a subset $\{t^{(i)}, t^{(i+1)}, \ldots, t^{(i+b)}\}$ of assigned adjacent slots on h, and slots $t^{(i-1)}$ and $t^{(i+b+1)}$, if exist, are free.

"bloco de split-locations and assigned location

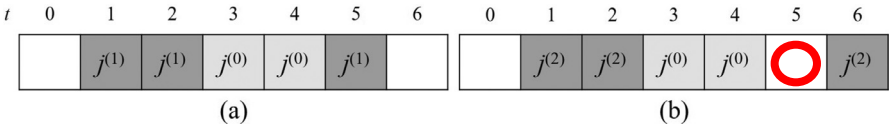

Fig. 4.4. An example of split-schedule (a) and preemptive schedule (b).

bloco de (validas) assigned locations

Definition 4.9. A feasible schedule block B' of a feasible schedule S is a set of assigned locations (h, G_j) on a machine h for a subset of jobs $j \in J' \subseteq J$, associated with a set of assigned adjacent slots $\{t^{(i)}, t^{(i+1)}, \ldots, t^{(i+b)}\}$ on h such that slots $t^{(i-1)}$ and $t^{(i+b+1)}$, if exist, are free.

Namely, a split-schedule block corresponds to a set of consecutive time slots, all assigned to jobs that are split-scheduled in the block. An example of split-schedule block and feasible schedule block is given in Fig. 4.5. Note that a split-schedule block is a split-schedule itself (i.e., if a job is assigned to a time slot in the split-schedule block, then the same job is completely processed in the block). The following theorem is essential to the formulation of SGH.

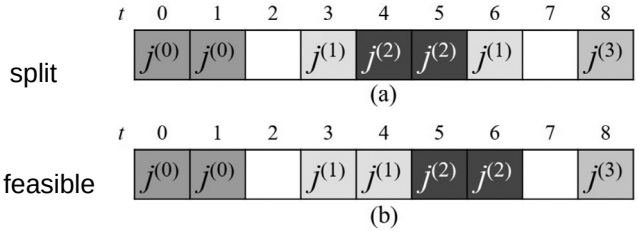

Fig. 4.5. An example of split-schedule block and feasible schedule block: slots 3-6 in (a) are a split-schedule block including jobs $j^{(1)}$ and $j^{(2)}$, whereas in (b) are a feasible schedule block for the same two jobs.

Theorem 4.1. *Any split-schedule S can be converted into an equivalent feasible schedule S'.*

Proof. Since the schedules on each machine are independent, it is sufficient to consider the schedule S_h on each machine $h \in H$ separately. In general, a split-schedule S_h on a machine can be partitioned into b split-schedule blocks B_i, $i = 1, \ldots, b$. We note that no permutation of time slots in B_i may affect the makespan or TEC of S, since such time slots are all assigned to jobs. Hence, in order to prove the theorem, it is sufficient to observe that a block B_i can be converted into an equivalent feasible schedule block by selecting a permutation of the time slots in B_i such that each job is assigned to adjacent slots in the block. □

Therefore, Theorem 4.1 ensures the possibility of converting any split-schedule into an equivalent feasible schedule. A procedure performing this task, essential to both SGH and ES, is shown in Algorithm 4.1. Among the alternative equivalent feasible schedules that can be obtained from a split-schedule S, Algorithm 4.1 generates the one that, on each machine, schedules the jobs with the same sequence of jobs in S in a feasible way, so that each job starts as soon as possible, but not earlier than its start slot in S. As detailed in the remainder of Section 4.1, Algorithm 4.1 efficiently generates an equivalent feasible schedule, since it considers each job, and possibly reschedules it at most once. More importantly, the computational complexity of Algorithm 4.1 does not affect the one of SGH (Algorithm 4.2), which is the only presented algorithm directly employing Algorithm 4.1 as a subroutine, as shown later in this section.

For each machine, Algorithm 4.1 performs two main steps: (1) it considers the time intervals in non-decreasing order to determine the starting time s_j of each job j (loop in lines 4 – 9), and (2) as soon as the starting time of a job j is identified, it assigns j to p_j consecutive time slots, starting from the first available slot greater than or equal to s_j (lines 5 – 7). In order to improve the computational efficiency of SGS-ES, all the algorithms presented in Section 4 employ Left-Leaning Red-Black (LLRB) trees (Sedgewick & Wayne, 2011) to represent schedules. LLRBs are based on Red-Black trees (Cormen, Leiserson, Rivest, & Stein, 2009), a type of self-balancing binary search trees. In this way, a schedule is stored as a set of LLRBs, each one associated with a single machine. Each LLRB stores the start times of the jobs as the keys of the tree, augmenting each node with the location of the related job as additional information. In our implementation, schedules S and S' in Algorithm 4.1 are represented accordingly.

Algorithm 4.1 Convert Schedule

Input A set $J = \{1, ..., N\}$ of jobs, with processing time p_j, $j \in J$.
A set $H = \{1, ..., M\}$ of machines.
A set $T = \{1, ..., K^{max}\}$ of time slots.
A split-schedule $S = \{S_h, h \in H\}$, a collection of sets of locations.
Output A feasible schedule S' equivalent to S.

1: $S' \leftarrow \{S'_h, h \in H\}$, a collection of empty sets $S'_h = \emptyset, h \in H$
2: **for each** $h \in H$ **do**
3: $s \leftarrow 0$ and $p \leftarrow 0$
4: **while** it exists a job j such that $\hat{s}_j = \min\{s_i : i \in J, (h, I_i) \in S_h, s_i > s\}$ **do**
5: $s'_j \leftarrow max(s + p, \hat{s}_j)$ //*The actual start time of j in the converted schedule*
6: $I'_j \leftarrow \{s'_j, s'_j + 1, ..., s'_j + p_j - 1\}$; $l' \leftarrow (h, I'_j)$
7: Reassign j to l' by adding l' to S'_h
8: $s \leftarrow \hat{s}_j$ and $p \leftarrow p_j$
9: **end while**
10: **end for**
11: **return** S'

As regards the complexity of Algorithm 4.1, for each $h \in H$, lines 4 – 9 are executed $O(\Omega)$ times, with $\Omega = min(N, K^{max})$, since the maximum number of jobs scheduled on a machine is N, and it is upper bounded by the makespan K^{max}. Lines 3 and 5 both take $O(1)$. The extraction of the entry with the lowest key among the ones with keys greater than a given one from a LLRB (line 4) consisting of n elements is $O(log_2 n)$. The insertion operation on a LLRB (line 7) has the same computational complexity. Therefore, lines 4 and 7 have a $O(log_2 \Omega)$ complexity.

In pseudocode, line 6 is stated consistently with the notation of locations introduced earlier in this section. However, since the assigned slots are always sequential, line 6 can easily be implemented to store only the starting time s_j and the ending time $s_j + p_j - 1$ of each job j. Therefore, the complexity of line 6 is considered to be $O(1)$. Since lines 2 – 10 are executed M times, the worst-case complexity of Algorithm 4.1 is $O(M \cdot \Omega \cdot log_2 \Omega)$.

The pseudocode of SGH is introduced in Algorithm 4.2. For a fixed value K^{max}, SGH, similarly to CH, constructively builds a schedule for the given set of jobs constrained to a maximum makespan equal to K^{max}. However, compared to CH, SGH (1) returns no feasible solution if K^{max} is too tight to schedule all the

Algorithm 4.2 Split-Greedy Heuristic (SGH)

Input A set $J = \{1, ..., N\}$ of jobs, with processing time p_j, $j \in J$.
A set $H = \{1, ..., M\}$ of machines, with unitary energy consumption rate e_h, $h \in H$.
A set $T = \{1, ..., K^{max}\}$ of time slots, with energy price c_k, $k \in T$, where K^{max} is the maximum allowed makespan.

Output A feasible schedule of the N jobs in the K^{max} time slots on the M machines, if it exists; otherwise, an empty schedule.

1: $S \leftarrow \{S_h, h \in H\}$, a collection of empty sets $S_h = \emptyset, h \in H$
2: Build a list $\hat{P}$ out of the elements in P_J, and sort $\hat{P}$ in non-increasing order
3: Build a map $F: P_J \rightarrow \mathcal{P}(J)$, with $\mathcal{P}(J)$ power set of J, and $\forall p \in P_J$, $F(p) \leftarrow \{j \in J: p_j = p\}$
4: **for each** $p \in \hat{P}$ **do**
5: **for each** $h \in H$ **do**
6: Build a list L_{ph} of the alternative free locations with least energy cost on h for any j, $p_j = p$
7: **end for**
8: **for each** $j \in F(p)$ **do**
9: **if** $L_{ph} = \emptyset \ \forall h \in H$ **then return** $\emptyset$ //*Return no feasible solution (no free location for j)*
10: Remove a least-cost, randomly chosen location $(\hat{h}, \hat{l})$ from $\bigcup_{h \in H} L_{ph}$, and denote it as $\hat{l}$
11: Assign location $\hat{l}$ to job j, and then insert $\hat{l}$ in $S_{\hat{h}}$
12: Remove any location or split-location $(\hat{h}, I)$ such that $I \cap \hat{l} \neq \emptyset$ from $L_{p\hat{h}}$
13: Add any split-location $(\hat{h}, I')$ with $|I'| = p$ (produced by the assignment of $\hat{l}$ to j) to $L_{p\hat{h}}$
14: **end for**
15: **end for**
16: **if** S is a split-schedule **then**
17: Convert schedule S with Algorithm 4.1
18: **end if**
19: **return** S

jobs (line 9); (2) it temporarily allows the generation of split-schedule blocks while scheduling the jobs; (3) it finally converts any split-schedule obtained by such construction into an equivalent feasible one using Algorithm 4.1. As in Algorithm 4.1, S is initialized as a set of empty LLRBs in $O(M)$ time. We denote the set $\{p_j: j \in J\}$ of distinct processing times of the jobs in J as P_J. In line 3, SGH partitions J into subsets of jobs $F(p)$ with the same processing time p. The computational complexity for lines $2-3$ is $O(|P_J| \cdot log_2|P_J| + N)$, since line 2 requires sorting $\hat{P}$ and line 3 builds F with the jobs in J. In lines $4-15$, SGH iterates the assignment of the jobs in the subsets $F(p)$, $p \in P_J$, according to the LPT rule. For each machine $h \in H$, SGH builds the list L_{ph} of all the available, free (possibly split-) locations on h with the least energy consumption cost for the jobs with processing time p (lines $5-7$). This operation is carried out over the time slots by advancing a pointer until K^{max} is reached. A queue with maximum capacity p is employed to store the free slots reached by the pointer. As soon as a free slot t is found, it is enqueued until the maximum capacity is reached. When this happens, a new free (possibly split-) location is identified. The start and the end time of such location are respectively equal to the oldest and the most recent elements in the queue. Then, whenever a new free slot t is found, the oldest element in the queue is dequeued and t is enqueued, progressively identifying new locations. Since the maximum number of jobs scheduled on each machine is Ω, querying the job scheduled in each slot, which consists in retrieving an element in a LLRB, requires $O(log_2 \Omega)$. However, we can improve this complexity by implementing SGH with an additional auxiliary table that stores information on the scheduled jobs. At any time of the execution of SGH, each entry (m, t) of this table stores the end time of the job with start time t scheduled on machine m, if such jobs exists; otherwise, a null value is stored.

This table can be easily initialized at the beginning of the procedure in $O(M \cdot K^{max})$. If the entry (m, t) is null, the slot t is not the start time of a job; otherwise, assume that j is the job scheduled on m with start time t. Then, the ending slot of j (which, if j is split-scheduled, is greater than $s_j + p_j - 1$) is given by the entry (m, t). This information can be queried in $O(1)$, and it can be used to skip already assigned jobs. Therefore, lines $5-7$ have a $O(M \cdot K^{max})$ computational complexity, and are executed for a total number of $O(|P_J|)$ times. Note that the computation of the energy consumption cost of each location does not impact the overall complexity, since it is carried out by adding (subtracting) the energy cost of enqueued (dequeued) slots in $O(1)$ while advancing over them. Line 9 evaluates the possibility of scheduling j by verifying in $O(M)$ time that at least a free location is available for j. In the opposite case, an empty schedule is returned. In line 10, we implemented an uniformly random selection among the possible least-cost locations for j in $O(M)$ time. We observed that this random choice produced better values of TEC compared to a deterministic choice that instead favours the earliest starting location. The machines containing the least cost locations are first identified in $O(M)$. Let us denote the set of such machines as H'. Then, we can partition the range $[0, \sum_{h \in H'} |L_{ph}|]$ according to the number of locations, i.e. $|L_{ph}|$, on each $h \in H'$. In this way, a random number $r \sim U[0, \sum_{h \in H'} |L_{ph}|]$, followed by an $O(M)$ manipulation of it, determines the exact position of a location in $\bigcup_{h \in H'} L_{ph}$. In this way, we can avoid explicitly performing the union operation reported in line 10, which has a $O(M \cdot K^{max})$ worst case complexity. Line 11 updates $S_{\hat{h}}$ by inserting the new location $\hat{l}$, and since the number of locations in $S_{\hat{h}}$ are $O(\Omega)$, the complexity of line 11 is $O(log_2 \Omega)$. After $\hat{l}$ is assigned to j in line 11, some slots become assigned, possibly affecting previously free locations, and even generating new ones. The list $L_{p\hat{h}}$ is updated accordingly (lines $12-13$) in $O(K^{max})$ time (see the analysis of lines $5-7$). Finally, if the resulting schedule S is a split-schedule, it is converted into a feasible one in line 17 with Algorithm 4.1 in $O(M \cdot \Omega \cdot log_2 \Omega)$. Therefore, the complexity of Algorithm 4.2 is $O(M + |P_J| \cdot log_2|P_J| + N) + O(|P_J| \cdot M \cdot K^{max}) + O(N \cdot M) + O(M \cdot \Omega \cdot log_2 \Omega) + O(N \cdot K^{max}) + O(M \cdot \Omega \cdot log_2 \Omega)$. Each of the terms of the sum respectively corresponds to the complexity of lines $1-3$, $5-7$, $9-10$, 11, $12-13$, and $16-18$, within the execution of SGH. The complexity of Algorithm 4.2 can be more compactly expressed as $O(|P_J| \cdot log_2|P_J| + N \cdot K^{max} + M \cdot (\Omega \cdot log_2 \Omega + |P_J| \cdot K^{max} + N))$.

4.2. Exchange Search

The performance of SGH is further improved with an ad-hoc designed local search procedure called Exchange Search. ES, starting from a feasible schedule S, changes the assignment of subsets of scheduled jobs to the machines in order to improve TEC, without worsening C^{max} and in some cases also possibly improving it. In order to describe ES, we state the following definitions.

Definition 4.10. An exchangeable period sequence (EPS) is a subset E of adjacent time slots on a given machine h in a feasible schedule S such that, if a time slot in E is assigned to a job j scheduled on h, then all the slots assigned to j are in E.

Fig. 4.6 provides an example to clarify the definition of EPS. The subset of time slots E_1 in Fig. 4.6(a) represents an EPS with cardinality equal to 5, since it contains jobs entirely scheduled in E_1 and an idle time slot. Differently, in Fig. 4.6(b), E_2 is not an EPS, since

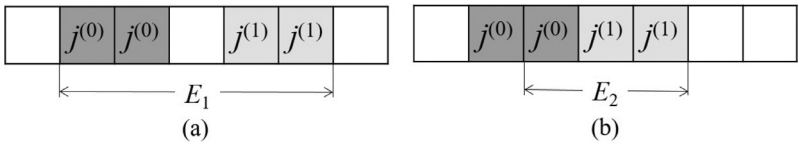

Fig. 4.6. Examples of sets of time slots that are (a) and not are (b) EPS.

job $j^{(0)}$ is not completely scheduled within E_2. Note that not all the time slots in an EPS are necessarily assigned to a job. In addition, according to Definition 4.10, both free and assigned locations are EPS's.

Definition 4.11. In a feasible schedule S, given two EPS's E and E', with $|E| = |E'|$, belonging to machines h and h', and such that $E \cap E' = \emptyset$ if $h = h'$, an EPS swap of E and E' is a procedure that schedules in E' (E) all the jobs scheduled in E (E'), without changing the relative assignment of such jobs to the slots in E (E').

Fig. 4.7 (a)–(b) gives an example of EPS swap involving job $j^{(0)}$ assigned to EPS E_1 on machine h and the jobs $j^{(1)}$ and $j^{(2)}$ assigned to EPS E_2 on machine h'. EPS swaps do not affect the feasibility of a schedule since the jobs involved in a swap are completely assigned to the slots in the swapped EPS. In the following, we introduce the definition of *rearrangement* of an EPS.

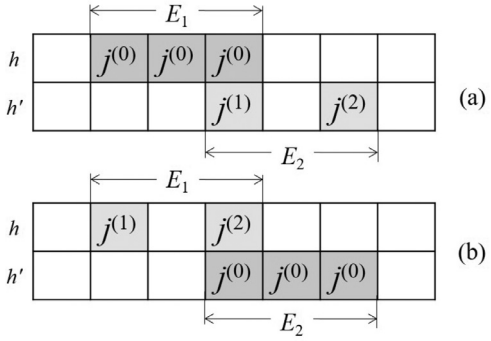

Fig. 4.7. An example of EPS swap.

Definition 4.12. Consider an EPS E in a feasible schedule S, and let $J_E(S)$ be the set of the jobs scheduled in E. A *rearrangement* of E with respect to S is a procedure which reassigns the slots in E to all the jobs in $J_E(S)$ considering only free adjacent slots so that all the jobs are feasibly scheduled.

In other words, in general, a rearrangement of E produces feasible schedule blocks that may differ from the ones previously present in E (e.g., see Fig. 4.8).

With Definition 4.12, we can now state that, given a feasible schedule S, any rearrangement of an EPS of S yields a feasible schedule S', which is trivial to prove.

In the following, in order to better describe ES, we denote an EPS which contains only slots assigned to a single job as an EPS-J, and an EPS with cardinality not exceeding p_{max}, which includes at least one idle slot, as an EPS-I, where $p_{max} = max_{p \in P_J}\{p\}$. Starting from a feasible schedule, ES explores a neighborhood generated by a subset of EPS swaps, updating the current best solution according to a *first improvement* policy which considers only TEC, without worsening makespan. In particular, the set of EPS-Js is considered in non-increasing order of their cardinality, and the evaluated swaps only involve an EPS-J and an EPS-I. The performed swaps are followed by a rearrangement of the jobs finally assigned to the slots in the involved EPS-J. To this end, SGH is easily generalized to reschedule the subset of jobs in an EPS-J (or an EPS-I), i.e., to operate on subsets $J' \subseteq J$, $H' \subseteq H$, and $T' \subseteq T$, with $|H'| = 1$. Since a single job is scheduled in an EPS-J, at most p_{max} jobs can be scheduled in an EPS-I, and the number of different processing times of the jobs scheduled in an EPS-J or an EPS-I is upper bounded by p_{max}, then $|P_{J'}| \leq p_{max}$ and $|J'| \leq p_{max}$. Therefore, the computational complexity of this modified procedure, denoted as *generalized* SGH, is $O(|P_{J'}| \cdot log_2|P_{J'}| + |J'| \cdot p_{max} + |J'| \cdot log_2 \Omega + |P_{J'}| \cdot p_{max} + |J'|) = O(p^2_{max} + p_{max} \cdot log_2 \Omega)$. Note that such expression directly derives from the considerations on the computational complexity of SGH done at the end of Section 4.1. In fact, generalized SGH builds the list of free locations with $O(p_{max})$ complexity for each of the $O(p_{max})$ processing times, it updates such list with the same complexity for each of the $O(p_{max})$ jobs, and it reschedules a subset of such jobs updating the schedule S by inserting each location in $O(log_2 \Omega)$ time.

Given a feasible schedule S, we define $E^J_p(S)$ and $E^I_p(S)$ respectively as the sets of EPS-Js and EPS-Is of cardinality $p \in P_J$ in S. We also introduce two sets of maps that allow to associate a time slot k on a machine h with an EPS E on the same machine, and such that the first slot of E is k. Formally, we denote as $\mathcal{M}^J_p(S) : H \times T \rightarrow E^J_p(S) \cup \emptyset$, $p \in P_J$, the maps that, for each $E \in E^J_p(S)$ on a machine h, associates $(h, min_{t' \in E}\{t'\})$ with E. The maps $\mathcal{M}^J_p(S)$ also associate any (h, t) with the empty EPS when there is no EPS-J E on machine h, with cardinality p, and having its first slot $min_{t' \in E}\{t'\}$ equal to t. Similarly, we denote as $\mathcal{M}^I_p(S) : H \times T \rightarrow E^I_p(S) \cup \emptyset$ the analogous maps for the EPS-Is. Finally, we define $E^J(S)$ and $E^I(S)$ respectively as the two collections of sets $\{E^J_p(S) : p \in P_J\}$ and $\{E^I_p(S) : p \in P_J\}$; similarly, we define $\mathcal{M}^J(S)$ and $\mathcal{M}^I(S)$ as $\{\mathcal{M}^J_p(S) : p \in P_J\}$ and $\{\mathcal{M}^I_p(S) : p \in P_J\}$, respectively. From now on, the dependence on S is omitted whenever it is obvious from the context.

The pseudocode of ES is shown in Algorithm 4.3. ES employs the subroutines *GenerateEPS*, *EvaluateSwapRearrange*, and *UpdateEPS*, which are not reported in pseudocode and are instead only described for the sake of brevity. In this way, we omit implementation details that are neither relevant to the analysis of the computational complexity, nor essential to the completeness of the presentation. It is important to note that, in the subroutines, the computation of energy consumption cost related to sequential slots has been speeded up by computing and storing the cumulative energy prices $\mu_{th}, t = 1, \ldots, K^{max}, h \in H$, with $\mu_{th} = e_h \cdot \sum_{i=1}^{t} c_i$ in $O(M \cdot K^{max})$ before the execution of ES. We also set $\mu_{0h} = 0$ for any $h \in H$. In this way, the energy consumption cost of any lo-

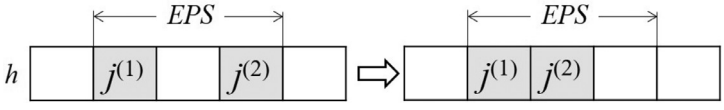

Fig. 4.8. An example of a valid EPS rearrangement.

Algorithm 4.3 Exchange-Search (ES)

Input A set $J = \{1, \dots, N\}$ of jobs, with processing time p_j, $j \in J$.
A set $H = \{1, \dots, M\}$ of machines, with unitary energy consumption rate e_h, $h \in H$.
A set $T = \{1, \dots, K^{max}\}$ of time slots, with energy price c_k, $k \in T$, where K^{max} is the maximum allowed makespan.
A split-schedule S as $\{L_h, h \in H\}$, a collection of sets of locations.

Output A new feasible schedule.

1: $E^J, E^I \leftarrow GenerateEPS(S, H, T, P_J)$
2: **for** each $p \in P_J$ **do**
3: Build the maps $\mathcal{M}^J_p : H \times T \to E^J_p$ and $\mathcal{M}^I_p : H \times T \to E^I_p$
4: **end for**
5: $\mathcal{M}^J \leftarrow \{\mathcal{M}^J_p, p \in P_J\}$; $\mathcal{M}^I \leftarrow \{\mathcal{M}^I_p, p \in P_J\}$
6: Build a list $\hat{P}$ out of the elements in P_J, and sort $\hat{P}$ in non-increasing order
7: **repeat**
8: $Improvement \leftarrow false$
9: **for** each $p \in \hat{P}$ **do**
10: **for** each $((h_g, s_g), g) \in \mathcal{M}^I_p$ **do**
11: **for** each $((h_l, s_l), l) \in \mathcal{M}^I_p$ **do**
12: $S, \delta \leftarrow EvaluateSwapRearrange(S, g, l)$
13: **if** $\delta < 0$ **then**
14: $E^J, E^I, \mathcal{M}^J, \mathcal{M}^I \leftarrow UpdateEPS(S, E^J, E^I, \mathcal{M}^J, \mathcal{M}^I, h_g, I_g, P_J)$
15: $E^J, E^I, \mathcal{M}^J, \mathcal{M}^I \leftarrow UpdateEPS(S, E^J, E^I, \mathcal{M}^J, \mathcal{M}^I, h_l, I_l, P_J)$
16: $Improvement \leftarrow true$
17: **break**
18: **end if**
19: **end for**
20: **end for**
21: **end for**
22: **until** *Improvement*
23: **return** S

cation l starting at slot t, and enclosing p time slots, defined as $\mathcal{C}_{t,t+p-1,h} := \mu_{t+p-1,h} - \mu_{t-1,h}$, can be computed in $O(1)$. Moreover, for the sake of compactness in the following explanations, we denote an EPS swap followed by a rearrangement of the involved EPS's as an EPS move.

We first present an overview of the subroutines and their use in Algorithm 4.3, providing details of the implementations and the analysis of the computational complexity later in this section. *GenerateEPS* identifies all the EPS-Js and the EPS-Is in a feasible schedule S, exploiting LLRBs and cumulative energy costs. This procedure is directly invoked at the beginning of Algorithm 4.3 to allow the computation of the maps $\mathcal{M}^J$ and $\mathcal{M}^I$. In this way, Algorithm 4.3 is able to evaluate EPS moves with *EvaluateSwapRearrange* in the main loop of the algorithm. In particular, Algorithm 4.3 repeatedly iterates over each EPS-J until no move involving an EPS-J and any EPS-I can improve TEC. *EvaluateSwapRearrange* efficiently evaluates if an EPS move entails an improvement in TEC disregarding some of the moves that cannot lead to improvement by exploiting a pruning condition. If the considered EPS move improves TEC, it is performed, and the maps $\mathcal{M}^J$ and $\mathcal{M}^I$ are updated accordingly by *UpdateEPS*. Invoked twice for each performed EPS move, *UpdateEPS* modifies the representations of the maps $\mathcal{M}^J$ and $\mathcal{M}^I$, which may become inconsistent with S after an EPS move. Note that *UpdateEPS* performs the needed changes invoking *GenerateEPS* to identify EPS's. We now delve into a formal and more detailed explanation of each subroutine.

GenerateEPS accepts a feasible schedule S, the subsets $\hat{H} \subseteq H$ and $\hat{T} \subseteq T$, and P_J as parameters, producing the two sets $\hat{E}^J$ and $\hat{E}^I$ according to S so that they respectively contain all the EPS-Js and the EPS-Is including slots in $\hat{T}$ on any machines $h \in \hat{H}$. The identification of an EPS in a schedule is conceptually similar to the task of finding free locations done in line 6 of Algorithm 4.2. In particular, for any $p \in P_J$ and $h \in \hat{H}$, two pointers, one for the head s of each EPS and the other for its tail e, are advanced through the slots in $\hat{T}$ until $max_{t \in \hat{T}}\, t$ is reached. By definition of EPS, $e - s + 1 = p$, and in order for the slots in $[s, e]$ to qualify as an EPS-I (or EPS-J), s must be free, or be the start time of a job, and e must be free, or be the end time of a job. If these conditions hold, the slots in $[s, e]$ are contained in an EPS. In particular, such EPS is an EPS-J if and only if all the slots in $[s, e]$ are assigned to a single job; otherwise, if the slots are not all assigned, such EPS is an EPS-I. Checking these conditions requires multiple queries to the LLRBs used to represent schedules, which yield a worst-case $O(log_2\, \Omega)$ complexity for each of such queries. In fact, each query is carried out at most twice for each $t \in \hat{T}$, one when the head pointer reaches t, and one when the tail pointer does. As a side, but significant, implementation note, given an EPS E on a machine h containing slots $\{t_i, t_{i+1}, ..., t_{i+p-1}\}$, the actual implementation of $\mathcal{M}^J_p$ and $\mathcal{M}^I_p$ associates the key $K^{max} \cdot h + t_i$ to E. This allows to uniquely address each EPS by its first slot. In order for *EvaluateSwapRearrange* not to evaluate moves that cannot lead to an improvement of TEC, along with each EPS E, *GenerateEPS* stores two additional information: the number of the assigned slots in E and the cumulative energy consumption cost of the list of slots contained in E, sorted by energy cost in non-decreasing order. Therefore, the computational complexity of *GenerateEPS* is $O(|P_J| \cdot |\hat{H}| \cdot |\hat{T}| \cdot max(log_2\, \Omega,\ p_{max} \cdot log_2(p_{max}))$. Note that the term $max(log_2\, \Omega, p_{max} \cdot log_2(p_{max}))$ is due to the fact that the cost of the innermost loop is dominated by the maximum between the cost of the queries on the LLRB and the cost of sorting the slots of the identified EPS's.

EvaluateSwapRearrange accepts a schedule S, an EPS-I $E^{(0)}$, and an EPS-J $E^{(1)}$ of the same cardinality p, and evaluates the impact on TEC due to a move involving them. The procedure exploits the additional information stored with each EPS by *GenerateEPS* for pruning an EPS move if it does not entail an improvement in TEC. Formally, consider an EPS E on machine h_E consisting of the time slots I_E, and denote with I^s_E the list of all the time slots in E, sorted according to their energy cost in non-decreasing order. If $I_{E^{(1)}} = \{t_i, t_{i+1}, ..., t_{i+p-1}\}$, we denote $I^s_{E^{(1)}}$ as $(t^s_i, t^s_{i+1}, ..., t^s_{i+p-1})$. We also denote as $EC_{E^{(0)}}(S)$ the energy cost associated with $E^{(0)}$ in the schedule S. We omit the dependence on S when it is clear from the context, and with a little abuse of notation we define $EC_{E^{(0)}E^{(1)}} := EC_{E^{(0)}} + EC_{E^{(1)}}$. Moreover, we denote the energy cost associated with E if all of its slots were assigned as $EC^{max}_E = e_{h_E} \sum_{k \in I_E} c_k$. Clearly, the number of assigned slots in E is $\alpha(J_E) := \sum_{j \in J_E} p_j$. Therefore, a move involving $E^{(0)}$ and $E^{(1)}$ would result in an energy cost which is lower bounded by $\hat{EC}^{lb}_{E^{(0)}E^{(1)}} = EC^{max}_{E^{(0)}} + C_{t^s_i, t^s_i + \alpha(J_{E^{(0)}}) - 1, h}$. The first term is due to the fact that, if the move is performed, the only job in the EPS-J $E^{(1)}$ is scheduled in the entire EPS-I $E^{(0)}$. The second term is instead related to the fact that if $\alpha(J_{E^{(0)}})$ slots are assigned in $E^{(0)}$, then the energy consumption cost associated with $E^{(1)}$ after the move is at least the sum of its $\alpha(J_{E^{(0)}})$ least cost slots. If $\hat{EC}^{lb}_{E^{(0)}E^{(1)}}$ is not less than $EC_{E^{(0)}E^{(1)}}$, then this move cannot entail an improvement in TEC. Therefore, the move is not performed, and $EC_{E^{(0)}E^{(1)}} - \hat{EC}^{lb}_{E^{(0)}E^{(1)}}$ is returned. Otherwise, the move is performed by scheduling the jobs in $J_{E^{(0)}}$ with the generalized SGH, yielding a schedule $\hat{S}$, and an energy cost $\hat{EC}_{E^{(0)}E^{(1)}}$ for the swapped EPS's. Afterwards, the move is rolled back only if it does not improve the current solution. Finally, the function returns $EC_{E^{(0)}E^{(1)}} - \hat{EC}_{E^{(0)}E^{(1)}}$. Clearly, a positive return value signals that a move has actually been performed

(without being rolled back). The computational complexity of *EvaluateSwapRearrange* is dominated by the one of generalized SGH, which is invoked once for each of the two EPS's involved. Therefore, the complexity of *EvaluateSwapRearrange* is $O(p_{max}^2 + p_{max} \cdot log_2 \Omega))$.

UpdateEPS accepts a schedule S, the two sets E^J and E^I, the two maps $\mathcal{M}^J$ and $\mathcal{M}^I$, a machine h_E, a slot subset I_E, and P_J, and returns the updated sets of EPS's and maps. This function iterates over each $p \in P_J$, and for each $t \in [max\{0, min_{t' \in I_E}\{t' - p + 1\}\}, min\{K^{max} - p + 1, max_{t' \in I_E}\{t'\}\}]$, removes the entry, if present, associated with (h_E, t) from both E_p^J and E_p^I. The total complexity of all the removal operations is $O(|P_J| \cdot p_{max})$. Afterwards, *UpdateEPS* invokes *GenerateEPS* with parameters S, $\{h_E\}$, I_E, and P_J. Let us denote its return values as $\hat{E}^I$ and $\hat{E}^J$. In order to add the newfound EPS's to the respective sets E^I and E^J, the operations $E_p^I \cup \hat{E}_p^I$ and $E_p^J \cup \hat{E}_p^J$ are carried out for each $p \in P_J$ in $O(|P_J| \cdot p_{max})$ time. The maps $\mathcal{M}^J$ and $\mathcal{M}^I$ are then updated accordingly, again in $O(|P_J| \cdot p_{max})$ time. Therefore, the complexity of *UpdateEPS* is $O(|P_J| \cdot p_{max} + |P_J| \cdot p_{max} \cdot max(log_2 \Omega,\ p_{max} \cdot log_2(p_{max})) = O(|P_J| \cdot p_{max} \cdot max(log_2 \Omega,\ p_{max} \cdot log_2(p_{max})))$. Note that this complexity is dominated by *GenerateEPS*.

As regards the computational complexity of Algorithm 4.3, line 1 employs the already discussed procedure *GenerateEPS* in order to identify the EPS's associated with the whole schedule. Then, for each $p \in P_J$, lines $2-4$ associate each key with the proper entry of the maps $\mathcal{M}_p^J$ and $\mathcal{M}_p^I$. Since there is a total of $M \cdot K^{max}$ keys, then lines $2-4$ take $O(|P_J| \cdot M \cdot K^{max})$ time. Afterwards, line 5 builds M^J and M^I according to their definition in $O(|P_J|)$ time, and line 6 builds the list $\hat{P}$ as done in Algorithm 4.2 in $O(|P_J| \cdot log_2 |P_J|)$ time. Lines $7-22$ are executed repeatedly, as long as they entail at least an improving move. Each possible move involving any EPS-J and any EPS-I is considered. Each of such moves is evaluated with *EvaluateSwapRearrange*, and if it is improving, it is performed. The lists of EPS's and the maps are updated with two calls to *UpdateEPS*. Note that an EPS-J of cardinality p can be swapped (and then rearranged) with up to $O(M \cdot (K^{max} - p + 1))$ EPS-Is. Therefore, since there are N different EPS-Js, the maximum number of moves to be evaluated is $O(N \cdot M \cdot K^{max})$. However, the exact number of moves, which is dependent on the set of the EPS-Is in the schedule, varies during the execution of ES. In particular, both J and the behaviour of (generalized) SGH affect the set of EPS-Is. We can observe that generally, the higher N and/or $\sum_{p \in P_J} p$, the lower is the number of idle slots, hence the smaller is the set of EPS-Is. Moreover, not all the possible ES moves are evaluated: the first improvement policy stops the evaluation of moves for an EPS-J E as soon as an improving move involving E is performed. In order to simplify the analysis without losing generality, we suppose that lines $7-22$ can be executed at most R times. As explained in the following, this constant has a very small upper bound in practice. Since the innermost loop of lines $7-22$ always invokes *EvaluateSwapRearrange* once, and *UpdateEPS* at most twice, the computational complexity of ES is $O(|P_J| \cdot M \cdot K^{max} \cdot max(log_2(K^{max}),\ p_{max} \cdot log_2(p_{max})) + O(|P_J| \cdot M \cdot K^{max}) + O(|P_J| \cdot log_2 |P_J|) + O(R \cdot N \cdot M \cdot K^{max} \cdot (p_{max}^2 + p_{max} \cdot log_2 \Omega + |P_J| \cdot p_{max} \cdot max(log_2(\Omega), p_{max} \cdot log_2(p_{max})))$. Each of the terms of this sum respectively corresponds to the complexity of *GenerateEPS* in line 1, of the construction of the maps in lines $2-4$, of lines $5-6$, and of the main loop in lines $7-22$. Note that the complexity of ES can be more compactly expressed as $O(R \cdot N \cdot M \cdot K^{max} \cdot p_{max}^2 \cdot log_2(p_{max}) \cdot log_2 \Omega)$. The analysis of the worst-case complexity of ES may suggest unpromising computational times. However, the actual performance observed even for the largest instance ($M = 40, N = 500, K = 500, |P_J| \leq 12, p_{max} \leq 12$) employed in the experimental campaign reported in Section 5 is valuable. In this connection, we also observed that R is at least an order of magnitude lower than T. The detailed validation of the effectiveness of ES is provided by our experimental campaign reported in Section 5.

Finally, LLRBs introduce an important optimization in *EvaluateSwapRearrange*. If each schedule is implemented as an ordered list, the computational complexity for scheduling each of the $O(p_{max})$ jobs in an EPS is $O(\Omega)$ instead of $O(log_2 \Omega)$. In fact, in this case, each of the job requires $O(log_2 \Omega)$ for binary search to identify the insertion point, and then $O(\Omega)$ time for shifting the elements after the insertion point, degrading the performance with respect to our implementation choice. This analysis completes the motivation for the introduction of LLRBs in the implementation.

A simple example is useful to illustrate and clarify the update operations performed by ES. In this example, we simply denote an EPS as a subset $T' \subseteq T$, without specifying the machine where the EPS is located as it is clear from the context. In Fig. 4.9(a) we consider a schedule for a single machine h with $e_h = 1$. There are two EPS-Js in E_1^J, one ($E^{(1)}$) including only the slot assigned to $j^{(1)}$, and the other ($E^{(2)}$) including only the slot assigned to $j^{(2)}$; moreover, there is an EPS-J ($E^{(0)}$) in E_3^J, including only the slots assigned to $j^{(0)}$. As regards the EPS-Is, $E_1^I = \{\{1, 5, 7, 9\}\}$, $E_2^I = \{\{5, 6\}, \{6, 7\}, \{7, 8\}, \{8, 9\}\}$, and $E_3^I = \{\{5, 6, 7\}, \{6, 7, 8\}, \{7, 8, 9\}\}$. Let us consider the possible EPS moves for the only EPS in E_3^J, which we denote as $E^{(a)}$, and $EC_{E^{(a)}} = 12$. We denote the three EPSs {5, 6, 7}, {6, 7, 8}, and {7, 8, 9} in E_3^I as $E^{(b)}$, $E^{(c)}$, $E^{(d)}$, respectively. Therefore, $EC_{E^{(b)}} = 4$, $EC_{E^{(c)}} = 8$, and $EC_{E^{(d)}} = 4$. Moreover, $EC_{E^{(b)}}^{max} = 15$, $EC_{E^{(c)}}^{max} = 13$, and $EC_{E^{(d)}}^{max} = 15$. The cumulative energy costs for $E^{(a)}$ are $\mathcal{C}_{2,2,h} = 1$, $\mathcal{C}_{2,3,h} = 2$, and $\mathcal{C}_{2,4,h} = 12$. All the three EPS-Is in E_3^I satisfy the lower bound condition for a move involving any of them and $E^{(a)}$, since $EC_{E^{(b)}}^{max} + \mathcal{C}_{2,2,h} = 16$, $EC_{E^{(c)}}^{max} + \mathcal{C}_{2,3,h} = 15$, and $EC_{E^{(d)}}^{max} + \mathcal{C}_{2,2,h} = 16$ are all less than $EC_{E^{(0)}} + EC_{E^{(1)}} + EC_{E^{(2)}}$. However, only the move involving $E^{(0)}$ and $E^{(c)}$ can actually decrease the *TEC* from 20 to 15. The schedule after the execution of this move is shown in Fig. 4.9(b).

Another example may be helpful in highlighting why ES can be effective and thus improve the quality of the solutions. Fig. 4.10(a) depicts a feasible schedule S, generated by SGH, for two machines h and h', and six jobs $j^{(0)}$, $j^{(1)}$, $j^{(2)}$, $j^{(3)}$, $j^{(4)}$ and $j^{(5)}$, with $p_{j^{(0)}} = 3$, $p_{j^{(1)}} = p_{j^{(2)}} = 2$, and $p_{j^{(3)}} = p_{j^{(4)}} = p_{j^{(5)}} = 1$. The makespan and TEC of schedule S shown in Fig. 4.10(a) are $C^{max}(S) = 7$ and $TEC(S) = 145$, respectively. Applying ES to S, only one improving EPS move is identified and performed. In particular, the set of slots {5, 6, 7} on machine h is an EPS-J including only $j^{(0)}$ that is swapped with the EPS-I corresponding to the set of slots {1, 2, 3} on machine h', which includes $j^{(3)}$, $j^{(4)}$ and an idle slot. After the swap, $j^{(3)}$, $j^{(4)}$ are rearranged by SGH in the EPS-I {5, 6, 7} on machine h, and the resulting schedule S', with $TEC(S') = 114 < TEC(S)$ is shown in Fig. 4.10(b). In addition, note that, due to the rearrangement, $C^{max}(S') = 6 < C^{max}(S)$.

The SGS-ES algorithm (obtained combining Split-Greedy Heuristic with Exchange Search) is shown in Algorithm 4.4.

Similarly to CH, SGS-ES iterates over all the possible values for the schedule makespan. At each iteration, it imposes a value $\hat{K}$ for the maximum makespan, and then it tries to build a feasible schedule for the given jobs by means of SGH, constrained to a time horizon not greater than $\hat{K}$. If SGH returns a feasible schedule, SGS-ES performs a local search step by applying ES. Otherwise, if for a given $\hat{K}$ no feasible solution is found, the main loop (lines 3-13) is interrupted, since no further solutions can be found for values of makespan less than the current $\hat{K}$. At algorithm termination, the set of non-dominated solutions found is finally returned. The computational complexity of lines $3-13$ is dominated by the one of ES. Therefore, the complexity of SGS-ES corresponds to the one of ES, up to a multiplicative factor of $K^{max} - K^{min} + 1$.

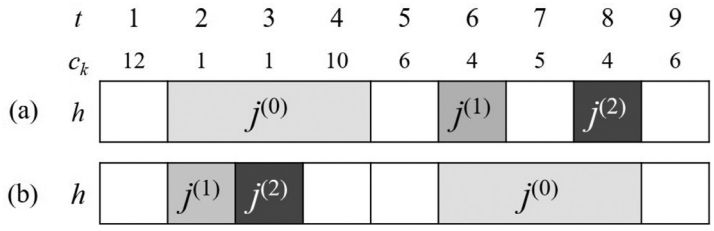

Fig. 4.9. An example of an EPS swap to illustrate the update operations performed.

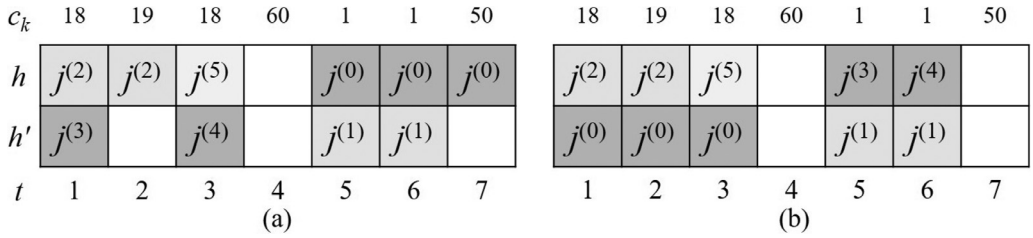

Fig. 4.10. An example of the application of ES.

Algorithm 4.4 Split-Greedy Scheduler with Exchange Search (SGS-ES)

Input A set $J = \{1, \ldots, N\}$ of jobs, with processing time p_j, $j \in J$.
A set $H = \{1, \ldots, M\}$ of machines, with unitary energy consumption rate e_h, $h \in H$.
A set $T = \{1, \ldots, K^{max}\}$ of time slots, with energy price c_k, $k \in T$.

Output A set of efficient solutions, each of which characterized by a pair (C^{max}, TEC).

$K^{min} \leftarrow max\{\lfloor \sum_{j \in J} p_j / M \rfloor, max_{j \in J}\{p_j\}\}$
Initialize the set of solutions $F \leftarrow \emptyset$ and makespan $\hat{K} \leftarrow K^{max}$
while $\hat{K} \geq K^{min}$ **do**
  $\hat{T} \leftarrow \{1, \ldots, \hat{K}\}$
  $S \leftarrow$ *Split-Greedy-Heuristic*$(J, H, \hat{T})$
  if S is a feasible schedule **then**
    $S \leftarrow$ *Exchange-Search*$(J, H, \hat{T}, S)$
  else
    break
  end if
  Compute C^{max} and TEC of S and add (C^{max}, TEC) to F
  $\hat{K} \leftarrow \hat{K} - 1$
end while
return the non-dominated solutions in the solution set F

5. Experimental results

In this section, the results of the set of experimental tests performed to validate the effectiveness of the proposed approach are shown. In order to provide more comprehensive results, 30 new very large-scale (VLS) instances (numbered from 61 to 90) were added to the set of 30 small-scale instances (numbered from 1 to 30) and the medium- and large-scale (MLS) instances (numbered from 31 to 60) proposed in Wang et al. (2018). The instances and the results obtained with our approach are available at https://github.com/ORresearcher/A-bi-objective-heuristic-approach-for-green-identical-parallel-machine-scheduling. For generating the VLS instances, the following combinations of parameters N, M, and K were used: $M \in \{25, 30, 40\}$, $N \in \{250, 300, 350, 400, 500\}$, $K \in \{350, 500\}$. Moreover, the processing times p_j, $j \in J$, were randomly generated from $U[1, 12]$. In order to reflect the tendencies of a highly volatile electricity market, the values of e_h, $h \in H$, and c_k, $k \in T$, were randomly drawn from the uniform distributions $U[1, 6]$ and $U[1, 8]$, respectively. Such choice fostered the generation of smaller time intervals with respect to the ones in Wang et al. (2018).

The experimental tests were carried out on a Windows 10 system equipped with a 2.2 gigahertz, 6 core Intel(R) Core i7-8750H CPU and 16 gigabytes of RAM. SGS-ES was implemented in Java 11, and the MATLAB implementation of CH kindly provided by Wang et al. (2018) was re-implemented in Java 11 to guarantee a fair comparison of the computational times. In addition, the Java version of CH was modified so that it does not return an unfeasible solution whenever it is not able to produce a feasible one, as discussed in Section 4.1. In order to validate the correctness of the CH re-implementation, the results obtained by the Java version of CH were compared with the ones produced by the MATLAB version. Such comparisons were all successful (i.e., their results coincide), except for the instances for which the MATLAB version generated an unfeasible solution.

Instead of using NSGA-II as a reference method for comparison, as it is done in Wang et al. (2018), we considered two more recent multi-objective approaches: NSGA-III (Deb & Jain, 2013) and the multi-objective evolutionary algorithm based on decomposition (MOEA/D) proposed in Zhang and Li (2007). Both algorithms were implemented in Java 11.

Section 5.1 briefly introduces the performance metrics used to compare the multi-objective methods. Then, Section 5.2 analyzes the improvements obtained with the introduction of ES. The performance analysis performed on the small-scale instances is reported in Section 5.3. Section 5.4 briefly describes NSGA-III and MOEA/D, the two approaches compared to SGS-ES other than CH. Finally, Section 5.5 presents the experimental results obtained in testing the different methods on the MLS and VLS instances.

5.1. Performance Analysis

Two classes of performance metrics are considered in Wang et al. (2018): the first aims at assessing the quality of a Pareto front, whereas the second evaluates the distribution of the efficient solutions on the Pareto front. To make the comparison with the results in Wang et al. (2018) more appropriate, we refer to the main indices reported in Wang et al. (2018), adding the *Hypervolume* (Zitzler & Thiele, 1999), and also considering the *Empirical Attainment Function* (EAF) (Da Fonseca, Fonseca, & Hall, 2001).

Of the three quality measures adopted in Wang et al. (2018), only two of them are evaluated in this paper:

- D_R (Ishibuchi, Yoshida, & Murata, 2003) is computed for a front F with respect to a reference Pareto front S^* as

$$D_R(F) = \frac{1}{|S^*|} \sum_{y \in S^*} min\{d_{xy} : x \in F\} \tag{11}$$

where $d_{xy} = \sqrt{\sum_{k=1}^{n} (f_k^*(x) - f_k^*(y))^2}$ is the Euclidean distance between two points x and y included in two distinct fronts, n is the dimension of the objective space, $f_k^*(a) = \frac{f_k(a) - f_k^{min}}{f_k^{max} - f_k^{min}}$ is the normalized k-th objective value, and f_k^{max} and f_k^{min} are the maximum and minimum values of the k-th objective function in the reference front S^*, respectively. If the optimal Pareto front is unknown, then S^* is obtained as the set of non dominated solutions in the union of the compared fronts. Smaller values of D_R denote higher quality fronts.

- *Purity* (Bandyopadhyay, Pal, & Aruna, 2004), defined as

$$P(F) = \frac{N_d(F^* \cap F)}{N_d(F)} \tag{12}$$

where $N_d(F)$ is the number of efficient solutions of front F. Larger values of Purity denote higher quality fronts.

Opposite to (Wang et al., 2018), the number of efficient solutions of a front is not reported since it is a not relevant quality index for comparing fronts. For instance, a front F with a larger number of points can be completely dominated by another front F' including far less points.

The second class includes the two following distribution and spread metrics used in Wang et al. (2018):

- *Minimal Spacing* (Bandyopadhyay et al., 2004) given by

$$MS(F) = \left(\frac{1}{N_d(F)} \sum_{i=1}^{N_d(F)} (\delta_i - \overline{\delta})^2 \right)^{1/2} \tag{13}$$

where

$$\delta_i = \min_{j \neq i} \sum_{k=1}^{n} |f_k(x_i) - f_k(x_j)| \tag{14}$$

and

$$\overline{\delta} = \frac{1}{N_d(F) - 1} \sum_{i=1}^{N_d(F)-1} \delta_i \tag{15}$$

- *Spread* (Deb et al., 2002), here denoted D_2 as in Wang et al. (2018), given by

$$D_2(F) = \frac{d_f + d_l + \sum_{i=1}^{N_d(F)-1} |d_i - \overline{d}|}{d_f + d_l + (N_d(F) - 1)\overline{d}} \tag{16}$$

where

$$d_i = \left(\sum_{k=1}^{n} \left(f_k(x_i) - f_k(x_{i+1}) \right)^2 \right)^{1/2} \tag{17}$$

and

$$\overline{d} = \frac{1}{N_d(F) - 1} \sum_{k=1}^{n} d_i \tag{18}$$

being x_i and x_{i+1} two consecutive points in front F, and d_f and d_l the Euclidean distances between the extreme solutions and the boundary solutions of front F, i.e.,

$$d_f = \left(\sum_{k=1}^{n} \left(f_k^{max} - \max_i f_k(x_i) \right)^2 \right)^{1/2} \tag{19}$$

and

$$d_l = \left(\sum_{k=1}^{n} \left(f_k^{min} - \min_i f_k(x_i) \right)^2 \right)^{1/2} \tag{20}$$

The smaller are the values for all the indices of this second class, the better is the distribution of the solution points in the front. However, we consider the distribution and spread indices only secondary to quality indices. In fact, we believe that the indices in the second class should be used to discern among fronts with a comparable degree of quality. As an example, we can consider a well distributed front that is totally dominated by another front which is instead poorly distributed: in such a situation we deem not appropriate to compensate the worse quality with the better distribution performance.

Besides the above indices used in Wang et al. (2018), we report the values of the Hypervolume (Zitzler & Thiele, 1999). This index provides a comprehensive evaluation of a Pareto front estimation since it measures the volume (or area, as in the two dimensional case dealt with in this paper) covered by a front with respect to a reference point in the objectives space. Therefore, it can be used to compare two or more fronts assuming a common reference point. Larger Hypervolume values denote a better approximation of the actual Pareto front, both in terms of quality and distribution of the solution points.

The comparison between two fronts can be performed also using EAF introduced in Da Fonseca et al. (2001). EAF provides a graphic representation of the probability of an algorithm to produce solutions that dominate a given point of the objective space in a single run (Minella, Ruiz, & Ciavotta, 2011). In particular, the *Differential Empirical Attainment Function* (Diff-EAF) proposed in López-Ibánez, Paquete, and Stützle (2006) shows the differences between a pair of EAFs graphically, highlighting the regions of the objective space in which an algorithm outperforms another in terms of produced solutions. However, since EAF and Diff-EAF do not provide an overall score, differently from the other indicators previously introduced, with these metrics the analysis can only be performed by observing the obtained diagrams. For this reason, in this paper we only report some examples of Diff-EAF.

Finally, we define two metrics, FM_1 and FM_2, to evaluate the feasibility of a front produced by an algorithm. In particular, these metrics are introduced to measure the incidence of the presence of unfeasible solution points in the fronts generated by CH since, as pointed out in Section 4.1, for some instances CH fails to schedule all the jobs. The feasibility metric $FM_1(F)$ computes the percentage of unfeasible points in a front F produced by an algorithm as

$$FM_1(F) = \frac{|U(F)|}{|F|} \tag{21}$$

where $U(F)$ denotes the set of unfeasible points of F. FM_2 provides the average percentage of non-scheduled jobs in the unfeasible instances produced by an algorithm and it is computed as

$$FM_2(F) = \begin{cases} \frac{|J_{ns}|}{|U(F)|N} & \text{if } FM_1(F) \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{22}$$

where J_{ns} is the set of non-scheduled jobs in the unfeasible solutions points of front F.

5.2. Effectiveness of the ES procedure

The first performance evaluation aims at comparing SGS and SGS-ES to determine whether ES is influential in obtaining better results. Table 5.1 reports the Hypervolume, Purity and Spread (D_2) as quality and distribution indices, and the computational times (*CPU*) in seconds for the two proposed heuristics, for both the MLS and the VLS instances. In addition, the rows *Avg* and *N.Best* in Table 5.1 show the averages and the number of instances for which a method has the best index value. In all the rows, the best values are in boldface. For each pair of compared methods, the last row p in Table 5.1 reports the probability value returned by the Friedman non-parametric test that we used to assess the significance

Table 5.1
Comparison of SGS and SGS-ES for the MLS and VLS instances.

Instance	Hypervolume		Purity		D_2		CPU		Istance	Hypervolume		Purity		D_2		CPU	
	SGS-ES	SGS	SGS-ES	SGS	SGS-ES	SGS	SGS-ES	SGS		SGS-ES	SGS	SGS-ES	SGS	SGS-ES	SGS	SGS-ES	SGS
31	**0.8813**	0.8790	**0.9232**	0.4475	**0.9173**	0.9292	0.0518	**0.0104**	61	**0.8198**	0.8130	**0.9994**	0.0068	1.0602	**1.0461**	21.7450	0.8068
32	**0.8090**	0.8082	**0.9742**	0.8478	0.9482	**0.9458**	0.0738	**0.0181**	62	**0.8295**	0.8220	**1.0000**	0.0032	**1.0119**	1.0160	62.3640	1.7699
33	**0.7791**	0.7777	**0.9876**	0.5517	**0.8728**	0.8882	0.0937	**0.0254**	63	**0.7660**	0.7599	**1.0000**	0.0078	0.8898	**0.8866**	25.8207	0.8211
34	**0.6960**	0.6901	**0.9510**	0.2159	0.7688	**0.7570**	0.1220	**0.0299**	64	**0.7862**	0.7805	**0.9996**	0.0033	0.9226	**0.9076**	57.0123	1.8385
35	**0.6011**	0.6009	**1.0000**	0.9541	0.5066	**0.5027**	0.0586	**0.0251**	65	**0.7245**	0.7116	**1.0000**	0.0044	0.8744	**0.8567**	30.4711	0.8903
36	**0.8542**	0.8518	**0.9496**	0.6613	0.7243	**0.7106**	0.0810	**0.0171**	66	**0.8014**	0.7910	**1.0000**	0.0075	**0.9823**	0.9981	75.7831	2.0618
37	**0.8709**	0.8695	**0.9463**	0.4795	0.8880	**0.8757**	0.1202	**0.0214**	67	**0.7581**	0.7518	**1.0000**	0.0060	0.8073	**0.7831**	25.9146	0.8793
38	**0.8409**	0.8399	**0.8638**	0.4692	**1.1245**	1.1427	0.1838	**0.0317**	68	**0.7391**	0.7266	**0.9998**	0.0046	0.8607	**0.8409**	84.8752	2.1309
39	**0.7830**	0.7814	**0.9895**	0.6211	**0.8684**	0.8748	0.2712	**0.0430**	69	**0.7368**	0.7289	**1.0000**	0.0188	0.8101	**0.8012**	25.9769	0.9363
40	**0.7827**	0.7730	**0.9553**	0.2247	**0.8447**	0.8456	0.3673	**0.0548**	70	**0.7508**	0.7378	**0.9999**	0.0064	0.8576	**0.8521**	90.3612	2.4123
41	**0.9282**	0.9282	**1.0000**	0.9938	**0.8120**	0.8125	0.0815	**0.0161**	71	**0.8057**	0.7987	**1.0000**	0.0000	**0.9612**	0.9786	25.2129	0.8829
42	**0.8529**	0.8523	**0.9943**	0.8179	**0.9458**	0.9489	0.1289	**0.0252**	72	**0.8735**	0.8691	**1.0000**	0.0000	1.0414	**1.0401**	65.3719	2.0116
43	**0.8613**	0.8599	**0.8563**	0.3881	**1.0217**	1.0239	0.1995	**0.0338**	73	**0.7955**	0.7878	**1.0000**	0.0041	0.9906	**0.9669**	30.7839	0.9326
44	**0.8312**	0.8301	**0.7902**	0.4311	1.0712	**1.0695**	0.3085	**0.0481**	74	**0.8508**	0.8456	**0.9992**	0.0048	1.0319	**1.0230**	77.8728	2.1324
45	**0.7900**	0.7871	**0.9491**	0.3861	0.8651	**0.8599**	0.5005	**0.0622**	75	**0.7888**	0.7825	**1.0000**	0.0043	0.9003	**0.8973**	34.7806	0.9713
46	**0.8242**	0.8194	**0.9145**	0.4845	**0.5920**	0.6045	0.4204	**0.1037**	76	**0.8508**	0.8438	**1.0000**	0.0059	0.9747	**0.9655**	92.2981	2.2562
47	**0.8832**	0.8811	**0.8538**	0.4783	**0.7928**	0.7968	0.7459	**0.1384**	77	**0.7478**	0.7404	**0.9999**	0.0050	0.8656	**0.8538**	37.2449	1.0383
48	**0.8724**	0.8676	**0.9573**	0.1964	0.7753	**0.7696**	0.9804	**0.1999**	78	**0.7890**	0.7781	**1.0000**	0.0058	**0.9621**	0.9639	107.1587	2.4133
49	**0.8025**	0.7972	**0.9686**	0.1526	0.8234	**0.8118**	1.4556	**0.2815**	79	**0.7464**	0.7371	**1.0000**	0.0128	0.8274	**0.7982**	45.8857	1.1171
50	**0.8275**	0.8226	**0.9864**	0.3545	**0.6961**	0.7183	1.8296	**0.3628**	80	**0.8158**	0.8099	**1.0000**	0.0035	0.9324	**0.9251**	101.0368	2.6189
51	**0.8457**	0.8427	**0.9286**	0.6523	0.8498	**0.8278**	0.7754	**0.1494**	81	**0.8201**	0.8147	**0.9987**	0.0062	**0.9468**	0.9510	37.5365	1.0809
52	**0.8657**	0.8642	**0.8922**	0.5680	**0.7515**	0.7907	1.1850	**0.1941**	82	**0.8646**	0.8596	**0.9970**	0.0066	**1.1053**	1.1441	92.0232	2.4982
53	**0.8875**	0.8864	**0.9411**	0.4613	0.8235	**0.8185**	1.6191	**0.2666**	83	**0.8199**	0.8135	**0.9992**	0.0024	**1.0663**	1.0665	42.3040	1.1566
54	**0.8836**	0.8825	**0.9019**	0.4204	**0.6813**	0.7081	2.1862	**0.3516**	84	**0.8282**	0.8230	**0.9998**	0.0041	**0.9831**	0.9930	113.6161	2.5701
55	**0.8445**	0.8417	**0.9874**	0.4762	0.8538	**0.8484**	3.3929	**0.4477**	85	**0.8237**	0.8175	**1.0000**	0.0030	**1.0316**	1.0450	49.3493	1.2331
56	**0.8867**	0.8852	**0.9792**	0.7035	0.8028	**0.7363**	1.1292	**0.1762**	86	**0.8308**	0.8248	**0.9999**	0.0007	**0.9742**	0.9781	129.1932	2.7451
57	**0.7435**	0.7377	**0.9392**	0.4743	0.6317	**0.6155**	1.7694	**0.2853**	87	**0.8091**	0.8034	**1.0000**	0.0003	0.9481	**0.9412**	52.3694	1.2788
58	**0.9057**	0.9047	**0.9960**	0.5480	**0.6884**	0.7099	2.4266	**0.3276**	88	**0.8372**	0.8308	**1.0000**	0.0033	**1.0088**	1.0124	149.7154	2.9976
59	**0.9012**	0.9002	**0.8873**	0.4814	0.9665	**0.9515**	4.6183	**0.4209**	89	**0.7629**	0.7485	**1.0000**	0.0004	0.9555	**0.9426**	72.6986	1.3990
60	**0.8169**	0.8150	**0.9478**	0.5063	0.7862	**0.7788**	4.2915	**0.5277**	90	**0.8093**	0.8004	**0.9998**	0.0066	1.0377	**1.0320**	161.0501	3.2595
Avg	**0.8318**	0.8292	**0.9404**	0.5149	0.8231	**0.8225**	1.0489	**0.1565**		**0.7994**	0.7918	**0.9997**	0.0050	0.9541	**0.9502**	67.2609	**1.7047**
N.Best	**30**	0	**30**	0	14	16	0	**30**		**30**	0	**30**	0	11	19	0	30
p	7.24E-04		4.32E-04		0.715		4.32E-04			4.32E-04		4.32E-04		0.1441		4.32E-04	

Table 5.2
Comparison between SGS-ES and MIP solver for the small-scale instances.

Instance	Hypervolume		Purity		D_R		D_2		Min Spacing		CPU	
	SGS-ES	MIP	SGS-ES	MIP	SGS-ES	MIP	SGS-ES	MIP	SGS-ES	MIP	SGS-ES	MIP
1	0.7460	0.7460	1.0000	1.0000	0.0000	0.0000	0.7161	0.7161	0.0465	0.0465	**0.0078**	5.5390
2	0.7744	**0.7756**	0.8919	**1.0000**	0.0018	**0.0000**	0.7251	**0.7175**	0.0389	**0.0387**	**0.0114**	10.9800
3	0.7398	**0.7587**	0.2321	**1.0000**	0.0230	**0.0000**	0.8110	**0.7460**	0.1053	**0.0993**	**0.0093**	3.0620
4	0.7563	**0.7566**	0.9444	**1.0000**	0.0004	**0.0000**	0.9532	**0.9428**	**0.0932**	0.0944	**0.0197**	11.5170
5	0.8016	**0.8019**	0.9400	**1.0000**	0.0004	**0.0000**	0.9381	0.9422	0.1101	0.1101	**0.0099**	6.5380
6	0.8234	**0.8247**	0.7882	**1.0000**	0.0024	**0.0000**	0.9323	0.9230	0.0903	**0.0902**	**0.0226**	10.2230
7	0.7257	**0.7259**	0.9815	**1.0000**	0.0003	**0.0000**	0.7267	0.7275	**0.0508**	0.0509	**0.0055**	6.7180
8	0.8138	**0.8153**	0.8938	**1.0000**	0.0012	**0.0000**	0.8283	0.8272	**0.0427**	0.0430	**0.0134**	13.2710
9	0.7561	**0.7687**	0.2367	**1.0000**	0.0128	**0.0000**	0.8176	0.8343	**0.0606**	0.0635	**0.0087**	4.1010
10	0.7722	**0.7732**	0.8704	**1.0000**	0.0014	**0.0000**	0.7425	0.7413	0.0593	0.0593	**0.0210**	28.3040
11	0.8077	0.8077	1.0000	1.0000	0.0000	0.0000	1.1115	1.1115	0.0724	0.0724	**0.0121**	12.5770
12	0.8485	**0.8506**	0.6032	**1.0000**	0.0028	**0.0000**	0.9929	0.9912	**0.0671**	0.0680	**0.0288**	22.0210
13	0.6900	0.6900	1.0000	1.0000	0.0000	0.0000	0.8796	0.8796	0.0382	0.0382	**0.0060**	5.1190
14	0.7532	**0.7548**	0.7588	**1.0000**	0.0016	**0.0000**	**0.7130**	0.7172	**0.0324**	0.0327	**0.0170**	13.9760
15	0.7508	**0.7520**	0.8356	**1.0000**	0.0012	**0.0000**	**0.9246**	0.9317	**0.0393**	0.0395	**0.0119**	10.8840
16	0.7987	**0.7993**	0.8560	**1.0000**	0.0005	**0.0000**	**0.8275**	0.8541	0.0328	**0.0317**	**0.0339**	33.5990
17	0.7082	**0.7086**	0.8591	**1.0000**	0.0006	**0.0000**	**0.5138**	0.5159	**0.0697**	0.0706	**0.0151**	12.5390
18	0.8107	**0.8156**	0.3508	**1.0000**	0.0061	**0.0000**	0.8817	**0.8804**	0.0371	**0.0365**	**0.0400**	37.1280
19	0.5785	**0.5794**	0.8419	**1.0000**	0.0010	**0.0000**	0.4309	**0.4210**	0.0521	**0.0520**	**0.0068**	5.5510
20	0.7294	0.7294	1.0000	1.0000	0.0000	0.0000	0.6909	0.6909	0.0338	0.0338	**0.0142**	42.6370
21	0.7455	**0.7458**	0.9292	**1.0000**	0.0004	**0.0000**	**0.7395**	0.7396	0.0517	0.0517	**0.0144**	7.4410
22	0.7968	**0.7970**	0.9245	**1.0000**	0.0008	**0.0000**	0.8954	**0.8872**	**0.0349**	0.0362	**0.0230**	65.0460
23	0.7704	**0.7743**	0.6714	**1.0000**	0.0043	**0.0000**	**0.9491**	0.9876	**0.0380**	0.0383	**0.0128**	13.8730
24	0.8403	**0.8413**	0.7955	**1.0000**	0.0013	**0.0000**	1.0476	**1.0415**	**0.0402**	0.0403	**0.0365**	31.3870
25	0.6976	0.6976	1.0000	1.0000	0.0000	0.0000	0.5976	0.5976	0.1111	0.1111	**0.0028**	2.7050
26	0.7683	0.7683	1.0000	1.0000	0.0000	0.0000	0.8472	0.8472	0.1042	0.1042	**0.0063**	5.0450
27	0.7425	0.7425	1.0000	1.0000	0.0000	0.0000	0.6324	0.6324	0.1558	0.1558	**0.0044**	3.0710
28	0.7438	0.7438	1.0000	1.0000	0.0000	0.0000	0.8421	0.8421	0.0977	0.0977	**0.0100**	4.6290
29	0.4531	0.4531	1.0000	1.0000	0.0000	0.0000	0.1677	0.1677	0.1830	0.1830	**0.0061**	1.8440
30	0.7697	0.7697	1.0000	1.0000	0.0000	0.0000	0.6432	0.6432	0.1486	0.1486	**0.0126**	4.9150
Avg	0.7504	**0.7522**	0.8402	**1.0000**	0.0021	**0.0000**	0.7840	**0.7832**	0.0713	0.0713	**0.0148**	14.5413
N.Best	0	20	0	20	0	20	9	11	11	6	30	0
p	7.74E-02		7.74E-02		7.74E-02		0.6547		0.2253		4.32E-08	

of the difference between the average values. More specifically, we assume that if p is smaller than 1%, then we can reject the null hypothesis that the two compared methods generate not significantly different results. Since both SGS and SGS-ES include random choices used to break ties (as detailed in Sections 4.1 and 4.2), the comparison between the two methods for the considered indices was carried out by averaging all possible comparisons between 10 runs of one method with 10 runs of the other. For the sake of compactness, the results regarding D_R and *Minimal Spacing* are not shown. Observing Table 5.1, we can note that SGS-ES prevailed in the Hypervolume and Purity indices on both the sets of instances, whereas SGS got slightly better average values for Spread. However, the statistical tests reveal that SGS-ES is significantly better than SGS for Hypervolume and Purity, whereas the results for Spread are statistically comparable. As regards the CPU times, SGS-ES, that includes the local search ES, is obviously more demanding than SGS, which in turn is a constructive heuristics. However, the average CPU time required by SGS-ES, that for the VLS instances is about 67 seconds, can be considered acceptable, especially in view of the achieved improvement in the performance results.

5.3. Analysis for the small-scale instances

In the second performance evaluation, the results produced by SGS-ES for the small-scale instances were compared with the ones generated by the MIP model presented in Section 3. To this end, we implemented an ϵ-constraint algorithm in Java 11 with CPLEX 12.10 as a MIP solver. In particular, the algorithm iteratively solves the MIP model of Section 3, minimizing only the objective (2), and imposing a decreasing upper bound for C^{max} from K down to K^{min} in constraint (8) at each iteration, terminating as soon as no feasible solution is found for a given upper bound. The obtained results

Table 5.3
Feasibility metrics applied to CH-M.

Instance	FM1	FM2	Instance	FM1	FM2
31			61	0.81%	2.00%
32			62	0.58%	2.80%
33			63	0.83%	1.50%
34			64	0.85%	3.00%
35			65	0.41%	0.57%
36			66		
37			67	1.81%	3.50%
38			68	1.17%	3.06%
39			69	0.51%	1.60%
40			70	1.23%	2.40%
41			71	1.28%	2.00%
42			72	0.78%	1.40%
43	1.54%	3.00%	73	1.72%	3.42%
44			74		
45			75	1.73%	3.93%
46			76	1.44%	4.14%
47			77	0.83%	3.25%
48			78	0.55%	4.25%
49	0.95%	2.67%	79	1.30%	2.47%
50			80	1.37%	2.00%
51			81	1.72%	3.40%
52			82	0.31%	0.80%
53			83	0.43%	1.00%
54	0.79%	2.67%	84	0.30%	0.33%
55			85	1.69%	3.43%
56			86		
57			87	0.45%	0.50%
58			88	0.61%	3.75%
59	0.72%	2.00%	89	0.43%	3.40%
60			90	0.60%	1.80%
Avg	1.00%	2.58%		0.95%	2.43%
Non-feasible	13.33%			90.00%	

Table 5.4
Comparison of Hypervolume and quality indices of SGS-ES, CH-J, NSGA-III and MOEA/D for the MLS instances.

Instance	Hypervolume				Purity				D_R			
	SGS-ES	CH-J	NSGA-III	MOEA/D	SGS-ES	CH-J	NSGA-III	MOEA/D	SGS-ES	CH-J	NSGA-III	MOEA/D
31	**0.8816**	0.8717	0.8717	0.8717	**0.99556**	0.0540	0.0540	0.0540	**0.0000**	0.0117	0.0116	0.0117
32	**0.8135**	0.8006	0.8006	0.8006	**1.00000**	0.0981	0.0983	0.0981	**0.0000**	0.0103	0.0102	0.0103
33	**0.7772**	0.7736	0.7736	0.7736	**1.00000**	0.3702	0.3702	0.3702	**0.0000**	0.0032	0.0032	0.0032
34	**0.6962**	0.6838	0.6839	0.6839	**0.99349**	0.1082	0.1082	0.1082	**0.0000**	0.0120	0.0119	0.0119
35	**0.6011**	0.5875	0.5875	0.5875	**1.00000**	0.3889	0.3889	0.3889	**0.0000**	0.0115	0.0115	0.0115
36	**0.8868**	0.8771	0.8773	0.8771	**1.00000**	0.5043	0.5304	0.5043	**0.0000**	0.0079	0.0076	0.0079
37	**0.8709**	0.8658	0.8658	0.8658	**0.98307**	0.2972	0.2992	0.2972	**0.0001**	0.0069	0.0068	0.0069
38	**0.8405**	0.8380	0.8380	0.8380	**0.84029**	0.4113	0.4113	0.4113	**0.0006**	0.0039	0.0039	0.0039
39	**0.7793**	0.7742	0.7742	0.7742	**1.00000**	0.3333	0.3333	0.3333	**0.0000**	0.0052	0.0052	0.0052
40	**0.7827**	0.7687	0.7687	0.7687	**0.98297**	0.1188	0.1188	0.1188	**0.0001**	0.0109	0.0109	0.0109
41	**0.9282**	0.9276	0.9280	0.9280	0.93750	0.9267	**0.9933**	**0.9933**	0.0016	0.0016	**0.0000**	**0.0000**
42	**0.8569**	0.8541	0.8541	0.8541	**1.00000**	0.7143	0.7143	0.7143	**0.0000**	0.0031	0.0031	0.0031
43	**0.8612**	0.8573	0.8573	0.8573	**0.96216**	0.1918	0.1918	0.1918	**0.0002**	0.0045	0.0045	0.0045
44	**0.8312**	0.8285	0.8285	0.8285	**0.80172**	0.3508	0.3508	0.3508	**0.0004**	0.0029	0.0029	0.0029
45	**0.7900**	0.7800	0.7800	0.7800	**0.86024**	0.2411	0.2411	0.2411	**0.0006**	0.0084	0.0084	0.0084
46	**0.8173**	0.8096	0.8114	0.8096	**0.89985**	0.3596	0.4207	0.3596	**0.0006**	0.0072	0.0068	0.0072
47	**0.8949**	0.8903	0.8903	0.8903	**0.94731**	0.3377	0.3377	0.3377	**0.0001**	0.0134	0.0134	0.0134
48	**0.8724**	0.8654	0.8654	0.8654	**0.87835**	0.2500	0.2500	0.2500	**0.0004**	0.0076	0.0076	0.0076
49	**0.8059**	0.7865	0.7865	0.7865	**1.00000**	0.0000	0.0000	0.0000	**0.0000**	0.0158	0.0158	0.0159
50	**0.8275**	0.8119	0.8120	0.8119	**0.99789**	0.3222	0.3222	0.3222	**0.0000**	0.0134	0.0134	0.0134
51	**0.8492**	0.8397	0.8420	0.8420	**0.81529**	0.4771	0.5600	0.5486	**0.0051**	0.0107	0.0056	0.0058
52	**0.8681**	0.8657	0.8658	0.8658	**0.79728**	0.5429	0.5429	0.5429	**0.0022**	0.0101	0.0101	0.0101
53	**0.8875**	0.8847	0.8847	0.8847	**0.93060**	0.3286	0.3286	0.3286	**0.0002**	0.0058	0.0058	0.0058
54	**0.8836**	0.8803	0.8803	0.8803	**0.97846**	0.1765	0.1765	0.1765	**0.0001**	0.0056	0.0056	0.0056
55	**0.8445**	0.8411	0.8411	0.8411	**0.97501**	0.4133	0.4133	0.4133	**0.0000**	0.0026	0.0026	0.0026
56	**0.8867**	0.8833	0.8841	0.8841	**0.87550**	0.4033	0.5278	0.5144	**0.0016**	0.0082	0.0054	0.0055
57	**0.7435**	0.7353	0.7363	0.7363	**0.85679**	0.5326	0.6011	0.5853	**0.0009**	0.0040	0.0031	0.0031
58	**0.9057**	0.9046	0.9046	0.9046	**1.00000**	0.4483	0.4483	0.4483	**0.0000**	0.0039	0.0039	0.0039
59	**0.9017**	0.8995	0.8995	0.8995	**0.96119**	0.3170	0.3170	0.3170	**0.0000**	0.0040	0.0040	0.0040
60	**0.8184**	0.8162	0.8162	0.8162	**0.93429**	0.3607	0.3607	0.3607	**0.0003**	0.0037	0.0037	0.0037
Avg	**0.8335**	0.8267	0.8270	0.8269	**0.94016**	0.3460	0.3604	0.3560	**0.0005**	0.0073	0.0070	0.0070
N.Best	30	0	0	0	29	0	1	1	29	0	1	1
p		1.87E-17				4.49E-16				2.05E-15		
Mean rank	4	1.75	2.2	2.05	3.9333	1.8	2.2833	1.9833	1.0833	3.2167	2.6667	3.0333
Rank position	1	2	2	2	1	2	2	2	1	2	2	2

are shown in Table 5.2, which reports all the considered indices and the CPU time. Table 5.2 shows that, on average, the MIP model produced better results for Hypervolume and the other two quality indices, i.e., Purity and D_R, whereas there is no statistical difference for the two distribution indices, i.e., Spread and Minimal Spacing. On the other hand, we can also note that SGS-ES found the same average result of the MIP model for Hypervolume, Purity and D_R for 10 out of 30 instances. As regards the CPU time, SGS-ES was able to solve the small-scale instances with an average time of three order of magnitude less than the MIP model. Taking into account this observation, we can consider the results for the small-scale instances produced by SGS-ES acceptable. However, as we show in the following section, the effectiveness of SGS-ES emerges for larger instances that cannot be solved in acceptable times by the MIP model.

5.4. The compared algorithms

As discussed in Section 3, the version of CH provided by Wang et al. (2018) may accept unfeasible solutions in the estimated Pareto front, since CH fails to recognize that the upper bound imposed by the maximum makespan at each iteration may not allow to schedule some jobs, producing a partial schedule instead of reporting unfeasibility. The version of CH implemented in Java, in the following denoted as CH-J, differs from the original algorithm, denoted as CH-M, since CH-J does not generate unfeasible points in the Pareto front. We tested CH-M on both MLS and VLS instances, reporting the feasibility metrics FM_1 and FM_2 in Table 5.3. From this table we can observe that about 13% of the MLS instances includes unfeasible solution points in the produced Pareto fronts, and on average, in such instances, 1% of solution points are unfeasible since an average of 2.58% of the jobs are not scheduled. If we consider the new set of VLS instances, Table 5.3 shows that most of the instances (90%) include solution points with unscheduled jobs, even if the average number of points and the average number of unscheduled jobs are similar to the ones for the MLS instances. Therefore, in the experimental tests presented in the following, we consider the results produced by CH-J.

As previously stated, we compared SGS-ES, with CH-J, NSGA-III, and MOEA/D. Both NSGA-III (Deb & Jain, 2013) and NSGA-II (Deb et al., 2002) are two well known and similar multi-objective evolutionary algorithms. NSGA-III differs from NSGA-II in the operator used to select the solutions to be included in the population since, to ensure the diversity of the obtained solutions, NSGA-III adopts a predefined set of reference points instead of the niche-preservation operator based on the crowding distance exploited by NSGA-II. Also MOEA/D (Zhang & Li, 2007) is an evolutionary multi-objective method based on a similar idea, since a set of weight vectors is defined a priori, and each generated solution is uniquely associated with a single vector to guarantee the solutions diversity.

We implemented NSGA-III converting the MATLAB version of NSGA-II used in Wang et al. (2018), and kindly shared by the authors. Therefore, we preserved the same design of NSGA-II, i.e., the same solution representation, initialization procedure, crossover and mutation operators, as well as the same parameters setting. The readers interested in the details of this algorithm, omitted here for the sake of brevity, can refer to (Wang et al., 2018). Similarly, we implemented the standard MOEA/D algorithm presented in Zhang and Li (2007), and we adapted it to the considered problem, exploiting again some parts of the designed NSGA-III, i.e., the same

Table 5.5
Comparison of distribution indices and CPU time of SGS-ES. CH-J. NSGA-III and MOEA/D for the MLS instances.

Instance	D_2				Min Spacing				CPU			
	SGS-ES	CH-J	NSGA-III	MOEA/D	SGS-ES	CH-J	NSGA-III	MOEA/D	SGS-ES	CH-J	NSGA-III	MOEA/D
31	**0.9173**	0.9638	0.9662	0.9638	0.0453	**0.0443**	0.0445	**0.0443**	**0.0518**	0.2630	14.0060	2.8346
32	0.9482	0.8792	**0.8782**	0.8792	0.0304	**0.0294**	0.0295	**0.0294**	**0.0738**	0.1037	14.4933	3.3755
33	0.8728	**0.8562**	**0.8562**	**0.8562**	**0.0232**	0.0238	0.0238	0.0238	**0.0937**	0.1145	15.1787	4.9112
34	**0.7687**	0.7887	0.7887	0.7887	**0.0169**	0.0171	0.0170	0.0170	0.1220	**0.0928**	16.0313	6.4969
35	**0.5066**	0.6053	0.6053	0.6053	**0.0205**	0.0232	0.0232	0.0232	**0.0586**	0.0918	17.3539	8.2203
36	0.7243	**0.6877**	0.6927	**0.6877**	0.0770	**0.0694**	0.0696	**0.0694**	**0.0810**	0.2132	15.1890	3.0904
37	**0.8880**	0.9034	0.9006	0.9034	0.0515	**0.0509**	0.0509	**0.0509**	**0.1202**	0.1646	15.3972	4.3526
38	1.1245	1.0910	**1.0910**	1.0910	0.0303	**0.0298**	0.0298	**0.0298**	**0.1838**	0.2798	16.1686	5.9278
39	**0.8684**	0.8764	0.8764	0.8764	0.0244	**0.0243**	**0.0243**	**0.0243**	0.2712	0.1295	17.0252	8.3602
40	0.8447	0.8119	**0.8118**	0.8119	**0.0250**	0.0258	0.0258	0.0258	0.3673	**0.1424**	17.6690	10.3599
41	0.8159	0.7885	**0.7667**	**0.7667**	0.1247	0.1259	**0.1234**	**0.1234**	**0.0815**	0.1231	14.7031	3.4379
42	**0.9458**	0.9669	0.9668	0.9669	**0.0682**	0.0713	0.0712	0.0713	0.1289	**0.1181**	15.3369	4.8556
43	1.0217	**0.9456**	**0.9456**	**0.9456**	**0.0376**	0.0389	0.0389	0.0389	0.1995	**0.1376**	15.7598	6.4588
44	**1.0712**	1.0759	1.0759	1.0759	0.0261	**0.0261**	0.0261	0.0261	0.3085	**0.1456**	16.7421	8.9836
45	0.8651	**0.8328**	**0.8328**	**0.8328**	**0.0237**	0.0237	0.0237	0.0237	0.5005	**0.1736**	18.4610	12.3467
46	**0.5925**	0.6452	0.6491	0.6452	**0.0487**	0.0510	0.0509	0.0510	**0.4204**	2.7928	16.9153	5.2404
47	0.7928	**0.7753**	0.7753	**0.7753**	0.0282	**0.0264**	0.0264	**0.0264**	**0.7459**	2.8224	17.0613	6.1811
48	0.7753	**0.7632**	0.7636	**0.7632**	0.0223	**0.0219**	0.0220	**0.0219**	**0.9804**	2.9008	17.7563	7.5760
49	**0.8234**	0.8301	0.8435	0.8508	**0.0088**	0.0089	0.0091	0.0093	**1.4556**	5.8274	21.1909	11.8035
50	0.6961	**0.6925**	0.6925	**0.6925**	0.0172	0.0155	**0.0155**	0.0155	**1.8296**	2.8851	19.4386	10.7617
51	0.8559	0.8635	**0.8297**	0.8297	**0.1124**	0.1550	0.1522	0.1524	**0.7754**	5.3334	19.7377	8.4114
52	0.7515	0.6922	**0.6921**	0.6922	**0.0848**	0.0903	0.0892	0.0892	**1.1850**	5.3997	20.4408	9.6556
53	0.8235	0.7922	**0.7922**	0.7922	0.0388	**0.0379**	**0.0379**	**0.0379**	**1.6191**	5.4256	20.8525	11.3704
54	0.6813	0.6765	**0.6764**	0.6765	**0.0338**	0.0341	0.0341	0.0341	**2.1862**	5.4775	22.1008	12.8782
55	0.8538	**0.8308**	**0.8308**	**0.8308**	**0.0144**	0.0148	0.0148	0.0148	**3.3929**	5.7765	22.7618	15.9638
56	0.8049	**0.7689**	0.7824	0.7821	**0.1013**	0.1122	0.1116	0.1115	**1.1292**	8.5075	23.0410	12.5976
57	0.6317	0.6023	**0.6018**	0.6023	**0.0835**	0.0993	0.0996	0.0996	**1.7694**	8.4910	23.5395	13.0080
58	0.6884	**0.6690**	**0.6690**	**0.6690**	0.0558	**0.0557**	**0.0557**	**0.0557**	**2.4266**	8.4105	24.5284	15.3027
59	0.9664	**0.8668**	0.8668	**0.8668**	0.0232	**0.0203**	0.0203	**0.0203**	**4.6183**	8.5497	25.4301	18.2997
60	0.7862	**0.7681**	**0.7681**	**0.7681**	**0.0228**	0.0238	0.0238	0.0238	**4.2915**	8.5050	26.9437	20.8700
Avg	0.8236	0.8103	0.8096	**0.8096**	**0.0440**	0.0464	0.0462	0.0462	**1.0489**	2.9799	18.7085	9.1311
N.Best	9	13	15	13	16	12	5	12	23	7	0	0
p	0.0138				0.8082				5.28E-18			
Mean rank	3.0667	2.3833	2.2	2.35	2.3667	2.6333	2.5333	2.4667	1.2333	1.7667	4	3
Rank position	2	1, 2	1	1, 2	1	1	1	1	1	1	3	2

solution representation, initialization procedure, crossover operator, and termination condition used in Wang et al. (2018). In addition, at each iteration of MOEA/D, we generate the candidate solutions to possibly update the ones currently associated with each weight vector, by means of the crossover operator in Wang et al. (2018). This operator is applied to a pair of parent solutions randomly extracted from the set including the *CW* solutions associated with the closest weight vectors, where the parameter *CW* is fixed to 10 as in Zhang and Li (2007).

5.5. Comparing SGS-ES with CH, NSGA-III and MOEA/D

Tables 5.4, 5.5, 5.6 and 5.7 show the comparisons among SGS-ES, CH-J, NSGA-III and MOEA/D. Since these tables report a comparison of four methods, we insert two last rows, which allow to discriminate the rank position of each method, besides the rows giving the average (*Avg*), the number of instances for which an algorithm produces the best value (*N.Best*), and the *p* value from the Friedman test. In particular, the row *Mean rank* gives the corresponding value returned by the multiple comparison test based on the Friedman test result, and the row *Rank position* shows the relative ranking for each index, considering the 99% confidence interval (note that rank 1 denotes the best method, and that methods that produced not statistically different results have the same rank).

Tables 5.4 and 5.5 provide the average values of the performance indices for the MLS instances. The SGS-ES approach clearly prevails as far as Hypervolume and the two quality indices in Table 5.4 are concerned, since it produced the best average value and it has the best rank. As far as distribution indices are concerned, in Table 5.5 we show that SGS-ES yielded the best average results for Minimal Spacing, whereas MOEA/D has the best average for Spread (D_2). For both Spread and Minimal Spacing, the *p* values denote that the results for the tested methods are statistically equivalent. In addition, all the methods share the same rank for Minimal Spacing, whereas only NSGA-III has a better rank than SGS-ES for Spread. For this set of instances, SGS-ES and CH-J are also the best methods as regards the computational times, since, even if SGS-ES needed on average less than half the CPU time of CH-J, they share the same rank.

The results for the VLS instances are reported in Tables 5.6 and 5.7. Even in this case, SGS-ES outperformed the other methods for the indices in Table 5.6. Observing Table 5.7, we can note that CH-J produced the best result for Spread and Minimal Spacing. However, the *p* values for such indices reveal the statistical equivalence of the methods. Finally, the computational times required by SGS-ES were significantly better than the ones of all the other methods.

The results obtained by SGS-ES for the sets of MLS and VLS instances show the effectiveness of SGS-ES, that was able to produce higher quality results in far less time compared to all the other tested heuristic methods. In Fig. 5.1, we also finally report an example of three pairs of comparisons of SGS-ES with the competitor methods (i.e., CH-J, NSGA-III and MOEA/D) for a MLS instance, given by means of the Diff-EAFs obtained using the visualization tool proposed in López-Ibánez, Paquete, and Stützle (2010). Apart from CH-J, which is a deterministic algorithm, the comparisons for SGS-ES, NSGA-III and MOEA/D were obtained from the results produced by these methods in 10 runs. Each diagram plots a lower and an upper line that respectively connect the best and worst points attained by the compared methods in all the runs. The plots also show a dashed line corresponding to the median attained by

Table 5.6
Comparison of Hypervolume and quality indices of SGS-ES, CH-J, NSGA-III and MOEA/D for the VLS instances.

Instance	Hypervolume				Purity				D_R			
	SGS-ES	CH-J	NSGA-III	MOEA/D	SGS-ES	CH-J	NSGA-III	MOEA/D	SGS-ES	CH-J	NSGA-III	MOEA/D
61	**0.8197**	0.8047	0.8047	0.8047	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0118	0.0117	0.0118
62	**0.8295**	0.8172	0.8172	0.8172	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0097	0.0096	0.0097
63	**0.7660**	0.7498	0.7498	0.7498	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0113	0.0113	0.0113
64	**0.7862**	0.7695	0.7695	0.7695	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0110	0.0110	0.0110
65	**0.7245**	0.7015	0.7015	0.7015	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0171	0.0171	0.0171
66	**0.8015**	0.7835	0.7835	0.7835	**1.0000**	0.0058	0.0058	0.0058	**0.0000**	0.0153	0.0153	0.0153
67	**0.7581**	0.7414	0.7416	0.7415	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0105	0.0106	0.0105
68	**0.7390**	0.7136	0.7138	0.7137	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0176	0.0177	0.0177
69	**0.7368**	0.7177	0.7177	0.7177	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0131	0.0131	0.0131
70	**0.7525**	0.7282	0.7282	0.7282	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0172	0.0173	0.0172
71	**0.8098**	0.7951	0.7952	0.7951	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0106	0.0107	0.0107
72	**0.8734**	0.8629	0.8630	0.8630	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0075	0.0075	0.0075
73	**0.7979**	0.7810	0.7812	0.7811	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0111	0.0112	0.0112
74	**0.8509**	0.8415	0.8416	0.8416	**0.9964**	0.0038	0.0038	0.0038	**0.0000**	0.0067	0.0067	0.0067
75	**0.7879**	0.7710	0.7712	0.7711	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0114	0.0115	0.0115
76	**0.8505**	0.8394	0.8396	0.8395	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0072	0.0073	0.0072
77	**0.7478**	0.7223	0.7223	0.7223	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0176	0.0176	0.0176
78	**0.7890**	0.7716	0.7716	0.7716	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0136	0.0136	0.0136
79	**0.7465**	0.7253	0.7254	0.7253	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0141	0.0142	0.0141
80	**0.8157**	0.8011	0.8012	0.8011	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0096	0.0097	0.0096
81	**0.8205**	0.8066	0.8069	0.8069	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0090	0.0089	0.0089
82	**0.8650**	0.8562	0.8563	0.8563	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0084	0.0084	0.0084
83	**0.8200**	0.8034	0.8036	0.8036	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0111	0.0109	0.0109
84	**0.8292**	0.8169	0.8169	0.8169	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0090	0.0089	0.0089
85	**0.8239**	0.8096	0.8097	0.8096	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0102	0.0102	0.0102
86	**0.8317**	0.8202	0.8202	0.8202	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0082	0.0082	0.0082
87	**0.8090**	0.7974	0.7974	0.7974	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0081	0.0081	0.0081
88	**0.8372**	0.8251	0.8251	0.8251	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0089	0.0089	0.0089
89	**0.7629**	0.7428	0.7429	0.7428	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0147	0.0146	0.0147
90	**0.8093**	0.7948	0.7949	0.7949	**1.0000**	0.0000	0.0000	0.0000	**0.0000**	0.0102	0.0101	0.0102
Avg	**0.7997**	0.7837	0.7838	0.7838	**0.9999**	0.0003	0.0003	0.0003	**0.0000**	0.0114	0.0114	0.0114
N.Best	30	0	0	0	30	0	0	0	30	0	0	0
p		1.66E-17				2.19E-19				2.39E-15		
Mean rank	4	1.5323	2.4335	2.0323	4	2	2	2	1	2.95	3.05	3
Rank position	1	3	2	2, 3	1	2	2	2	1	3	3	3

Table 5.7
Comparison of distribution indices and CPU time of SGS-ES. CH-J. NSGA-III and MOEA/D for the VLS instances.

Instance	D_2				Min Spacing				CPU			
	SGS-ES	CH-J	NSGA-III	MOEA/D	SGS-ES	CH-J	NSGA-III	MOEA/D	SGS-ES	CH-J	NSGA-III	MOEA/D
61	1.0602	1.0473	**1.0459**	1.0471	0.0077	0.0073	0.0073	**0.0073**	**21.7450**	50.1929	58.8327	66.7168
62	**1.0119**	1.0496	1.0464	1.0496	0.0055	**0.0053**	0.0054	0.0053	**62.3640**	472.9695	327.7931	488.7253
63	0.8898	0.8261	0.8256	**0.8255**	**0.0055**	0.0056	0.0056	0.0056	**25.8207**	38.8928	51.8122	58.2162
64	0.9226	**0.8744**	0.9106	0.8880	0.0051	**0.0043**	0.0059	0.0051	**57.0123**	457.8403	312.8328	476.2181
65	**0.8744**	0.9017	0.9005	0.9017	0.0059	**0.0056**	0.0056	**0.0056**	**30.4711**	46.5792	61.1699	67.6016
66	**0.9823**	1.0137	1.0132	1.0137	0.0056	**0.0053**	0.0053	**0.0053**	**75.7831**	294.5378	215.1681	314.6727
67	0.8073	**0.7758**	0.7919	0.7892	**0.0053**	0.0054	0.0062	0.0062	**25.9146**	61.7673	73.8247	86.6843
68	0.8607	**0.8158**	0.8333	0.8314	0.0039	**0.0037**	0.0041	0.0043	**84.8752**	321.0555	244.1829	344.4683
69	0.8101	0.7642	**0.7625**	0.7626	**0.0054**	0.0056	0.0055	0.0055	**25.9769**	69.9960	81.8046	99.3700
70	0.8576	**0.8543**	0.8780	0.8794	0.0043	**0.0040**	0.0049	0.0051	**90.3612**	333.5820	254.6311	362.0777
71	0.9612	**0.9179**	0.9734	0.9788	0.0076	**0.0071**	0.0097	0.0101	**25.2129**	77.2715	77.8492	94.0222
72	1.0414	1.0363	**1.0323**	1.0335	0.0084	0.0084	0.0084	**0.0084**	**65.3719**	129.7969	108.7024	147.3442
73	0.9906	**0.9181**	0.9536	0.9519	**0.0068**	0.0072	0.0083	0.0084	**30.7839**	55.2623	64.4688	75.9908
74	1.0319	**1.0087**	1.0087	1.0087	0.0079	**0.0078**	0.0078	0.0078	**77.8728**	186.2354	145.0289	206.7596
75	0.9003	**0.8709**	0.8948	0.8977	0.0064	**0.0061**	0.0070	0.0072	**34.7806**	53.0671	65.9283	77.1399
76	**0.9747**	0.9768	0.9945	0.9922	**0.0074**	0.0075	0.0081	0.0081	**92.2981**	100.0133	94.9817	124.1663
77	0.8656	0.8515	**0.8510**	0.8511	0.0057	0.0055	0.0055	**0.0055**	**37.2449**	79.5971	86.5105	105.8310
78	0.9621	**0.9338**	0.9338	**0.9338**	0.0052	**0.0047**	0.0047	**0.0047**	**107.1587**	378.6946	267.8694	405.3375
79	0.8274	**0.7902**	0.8018	0.7966	0.0053	**0.0051**	0.0058	0.0055	**45.8857**	52.1700	71.6422	83.8278
80	0.9324	**0.9018**	0.9218	0.9127	**0.0054**	0.0055	0.0062	0.0059	**101.0368**	228.2797	187.8734	261.1875
81	0.9468	**0.9066**	0.9523	0.9528	0.0096	**0.0084**	0.0115	0.0116	**37.5365**	54.1488	60.3279	75.1196
82	**1.1053**	1.1475	1.1439	1.1441	0.0081	0.0077	**0.0077**	0.0077	**92.0232**	369.0146	254.4615	389.2578
83	1.0663	1.0155	**1.0145**	1.0146	**0.0077**	0.0078	0.0079	0.0079	**42.3040**	74.4537	77.4552	98.9680
84	0.9831	**0.9649**	0.9666	0.9669	0.0061	**0.0058**	0.0058	0.0058	**113.6161**	351.1298	248.0501	375.9355
85	1.0316	**1.0017**	1.0243	1.0186	0.0071	**0.0069**	0.0079	0.0077	**49.3493**	62.4446	72.6152	89.6216
86	**0.9742**	0.9830	0.9828	0.9830	0.0058	**0.0055**	0.0055	0.0055	**129.1932**	180.4264	145.6146	207.6757
87	0.9481	**0.9275**	0.9276	**0.9275**	0.0072	0.0068	**0.0068**	0.0068	**52.3694**	126.2987	119.3945	156.4167
88	1.0088	1.0092	**1.0047**	1.0055	0.0061	0.0059	0.0059	**0.0059**	**149.7154**	300.2133	229.4651	330.7565
89	0.9555	0.9294	**0.9275**	0.9293	0.0070	0.0066	0.0066	**0.0066**	72.6986	**46.6116**	68.8165	83.1229
90	1.0377	1.0098	**1.0089**	1.0098	**0.0052**	0.0054	0.0054	0.0054	**161.0501**	285.6970	220.1420	323.1561
Avg	0.9541	**0.9341**	0.9442	0.9432	0.0063	**0.0061**	0.0066	0.0066	**67.2609**	177.9413	144.9750	202.5463
N.Best	6	15	8	3	8	15	2	8	29	1	0	0
p		0.0056				0.0049				1.02E-16		
Mean rank	3.1667	2.1667	2.1667	2.5	2.6333	1.8667	2.75	2.75	1.0667	2.5	2.4333	4
Rank position	3	2	1	1	2	1	2	2	1	2	2	3

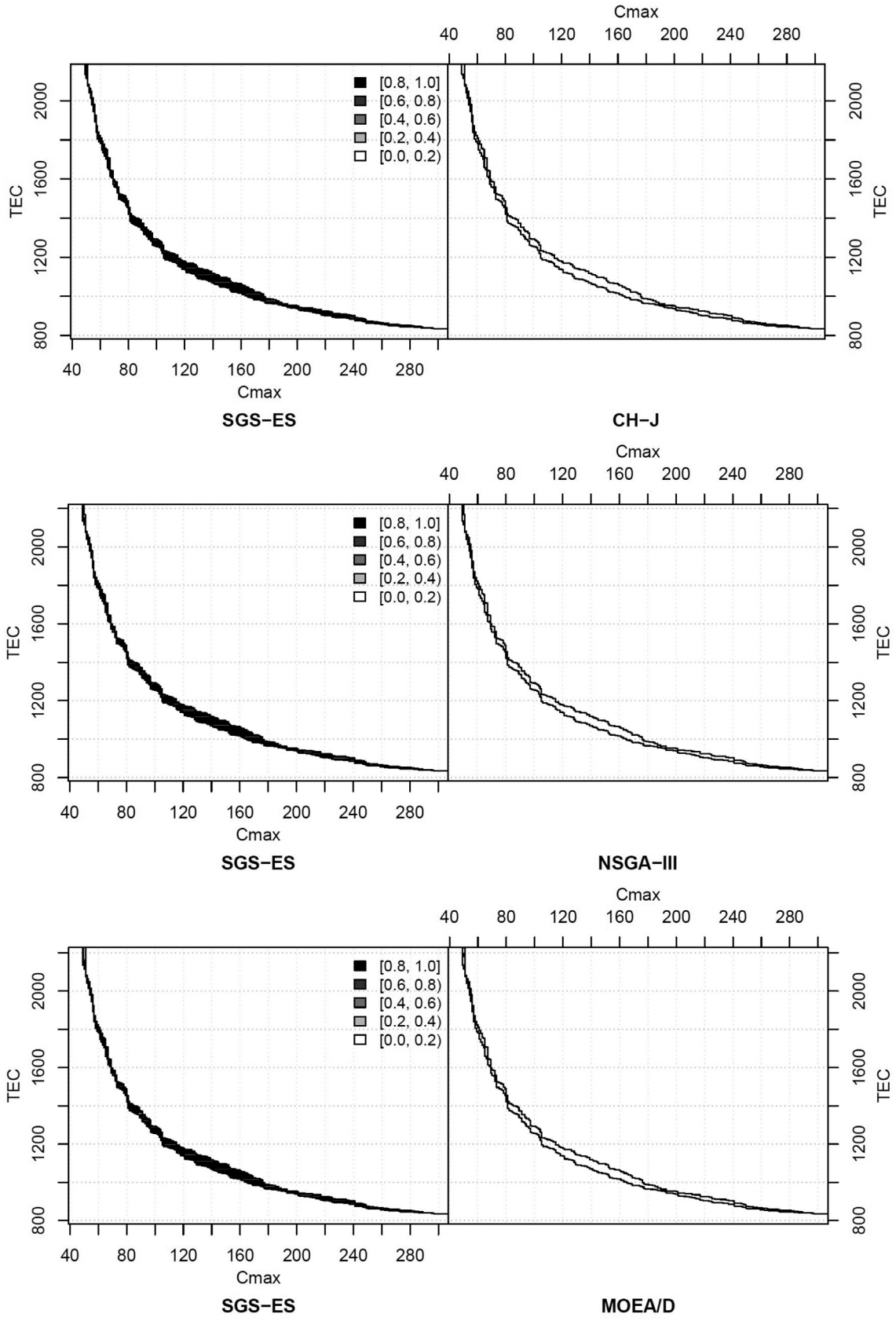

Fig. 5.1. The Diff-EAF plots comparing SGS-ES with CH-J, NSGA-III and MOEA/D for the medium-large instance 49.

each algorithm. The areas highlighted with different grey levels show the points where the EAF of an algorithm is larger than the one of the compared algorithm of at least 20% (the darker is the area, the larger is the difference). As we can observe in all the pairs of diagrams, SGS-ES always produced better results than the compared algorithms, confirming the conclusions about its effectiveness. As already pointed out, a thorough analysis using Diff-EAFs is not practical; however, we can report that the results obtained considering a sample of several instances produced outcomes similar to the ones in Fig. 5.1.

6. Conclusions

In this paper we present an approach to solve the bi-objective scheduling problem proposed by Wang et al. (2018). This problem belongs to the class of green scheduling or energy-aware scheduling, since it aims at reducing the impact of the associated manufacturing production on the environment due to the energy consumption. To this end, we defined SGH as the extended and revised version of the constructive heuristic proposed in Wang et al. (2018), and we employed it in a multi-objective scheduling algorithm, SGS, that we significantly enhanced by means of ES, a novel local search. The resulting algorithm, SGS-ES, was compared to CH and two multi-objective evolutionary approaches, NSGA-III and MOEA/D, with tests on the MLS instances proposed by Wang et al. (2018), and on a new set of VLS instances. The obtained results show that SGS-ES is significantly better than the other considered approaches. The results of CH were obtained by reimplementing it, while correcting the feasibility problem afflicting the original implementation. As the experimental results show, such problem is highly relevant for the set of very large-scale instances. The experimental campaign leads us to conclude that SGS-ES is an effective approach for energy-aware and green scheduling on parallel machines in presence of TOU energy prices, since it allows to reduce energy consumption while yielding overall better solutions. Finally, future developments may consist in including additional features to the problem in order to model some other realistic situations. In particular, we prospect the possibility of considering non-identical machines, sequence and machine-dependent setup times, as well as job-dependent energy consumption.

References

Alhadi, G., Kacem, I., Laroche, P., & Osman, I. M. (2020). Approximation algorithms for minimizing the maximum lateness and makespan on parallel machines. *Annals of Operations Research, 285*(1-2), 369–395.

Bandyopadhyay, S., Pal, S. K., & Aruna, B. (2004). Quantitative indices, and pattern classification. *IEEE Transactions On Systems, Man, And Cybernetics, 34*(5), 2088–2099. https://doi.org/10.1109/TSMCB.2004.834438.

Branke, J., Deb, K., Miettinen, K., & Slowiński, R. (2008). *Multiobjective Optimization: Interactive and Evolutionary Approaches*. Berlin Heidelberg: Springer-Verlag.

Bruzzone, A. A. G., Anghinolfi, D., Paolucci, M., & Tonelli, F. (2012). Energy aware scheduling for improving manufacturing process sustainability: a mathematical model for flexible flow shops. *CIRP Annals - Manufacturing Technology, 61*(1), 459–462. https://doi.org/10.1016/j.cirp.2012.03.084.

Che, A., Wu, X., Peng, J., & Yan, P. (2017a). Energy-efficient bi-objective single-machine scheduling with power-down mechanism. *Computers & Operations Research, 85*, 172–183. https://doi.org/10.1016/j.cor.2017.04.004.

Che, A., Zhang, S., & Wu, X. (2017b). Energy-conscious unrelated parallel machine scheduling under time-of-use electricity tariffs. *Journal of Cleaner Production, 156*, 688–697. https://doi.org/10.1016/j.jclepro.2017.04.018.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms*. MIT Press.

Cota, L. P., Guimarães, F. G., Ribeiro, R. G., Meneghini, I. R., de Oliveira, F. B., Souza, M. J. F., & Siarry, P. (2019). An adaptive multi-objective algorithm based on decomposition and large neighborhood search for a green machine scheduling problem. *Swarm and Evolutionary Computation, 51*, 100601.

Da Fonseca, V. G., Fonseca, C. M., & Hall, A. O. (2001). Inferential performance assessment of stochastic optimisers and the attainment function. In *Proceedings of the international conference on evolutionary multi-criterion optimization* (pp. 213–225). Springer.

Deb, K., & Jain, H. (2013). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: solving problems with box constraints. *IEEE Transactions on Evolutionary Computation, 18*(4), 577–601.

Deb, K., & Jain, H. (2014). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *IEEE Transaction on Evolutionary Computation, 18*(4), 577–601. https://doi.org/10.1109/TEVC.2013.2281535.

Deb, K., Pratap, A., Agarwal, S., & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-II. *IEEE Transaction on Evolutionary Computation, 6*(2), 182–197. https://doi.org/10.1109/4235.996017.

Despeisse, M., Ball, P. D., Evans, S., & Levers, A. (2012). Industrial ecology at factory level - a conceptual model. *Journal of Cleaner Production, 31*, 30–39. https://doi.org/10.1016/j.jclepro.2012.02.027.

Ding, J., Song, S., Zhang, R., Chiong, R., & Wu, C. (2016). Parallel machine scheduling under time-of-use electricity prices: new models and optimization approaches. *IEEE Transactions on Automation Science and Engineering, 13*(2), 1138–1154. https://doi.org/10.1109/TASE.2015.2495328.

Fang, K., & Lin, B. (2013). Parallel-machine scheduling to minimize tardiness penalty and power cost. *Computers & Industrial Engineering, 64*(1), 224–234. https://doi.org/10.1016/j.cie.2012.10.002.

Fang, K., Uhan, N. A., Zhao, F., & Sutherland, J. W. (2013). Flow shop scheduling with peak power consumption constraints. *Annals of Operations Research, 206*, 115–145. https://doi.org/10.1007/s10479-012-1294-z.

Gahm, C., Denz, F., Dirr, M., & Tuma, A. (2016). Energy-efficient scheduling in manufacturing companies: a review and research framework. *European Journal of Operational Research, 248*(3), 744–757. https://doi.org/10.1016/j.ejor.2015.07.017.

Giglio, D., Paolucci, M., & Roshani, A. (2017). Integrated lot sizing and energy efficient job shop scheduling problem in manufacturing/remanufacturing systems. *Journal of Cleaner Production, 148*, 624–641. https://doi.org/10.1016/j.jclepro.2017.01.166.

Giret, A., Trentesaux, D., & Prabhu, V. (2015). Sustainability in manufacturing operations scheduling: a state of the art review. *Journal of Manufacturing Systems, 37*(1), 126–140.

Ishibuchi, H., Yoshida, T., & Murata, T. (2003). Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. *IEEE Transactions on Evolutionary Computation, 7*(2), 204–223.

Ji, M., Wang, J., & Lee, W. (2013). Minimizing resource consumption on uniform parallel machines with abound on makespan. *Computers & Operations Research, 40*, 2970–2974.

Jia, Z.-h., Li, Y.-j., Li, K., & Chen, H.-p. (2020). Weak-restriction bi-objective optimization algorithm for scheduling with rejection on non-identical batch processing machines. *Applied Soft Computing, 86*, 105914.

Li, K., Shi, Y., Yang, S., & Cheng, B. (2011). Parallel machine scheduling problem to minimize the makespan with resource dependent processing times. *Applied Soft Computing, 11*(8), 5551–5557.

Li, K., Zhang, X., Leung, J. Y.-T., & Yang, S. L. (2016). Parallel machine scheduling problems in green manufacturing industry. *Journal of Manufacturing Systems, 38*, 98–106.

Liu, Y., Dong, H., Lohse, N., & Petrovic, S. (2016). A multi-objective genetic algorithm for optimisation of energy consumption and shop floor production performance. *International Journal of Production Economics, 179*, 259–272.

López-Ibánez, M., Paquete, L., & Stützle, T. (2006). Hybrid population-based algorithms for the bi-objective quadratic assignment problem. *Journal of Mathematical Modelling and Algorithms, 5*(1), 111–137.

López-Ibánez, M., Paquete, L., & Stützle, T. (2010). Exploratory analysis of stochastic local search algorithms in biobjective optimization. In *Experimental methods for the analysis of optimization algorithms* (pp. 209–222). Springer.

Lu, C., Gao, L., Li, X., Pan, Q., & Wang, Q. (2017). Energy-efficient permutation flow shop scheduling problem using a hybrid multi-objective backtracking search algorithm. *Journal of Cleaner Production, 144*, 228–238.

Lu, C., Gao, L., Li, X., Zheng, J., & Gong, W. (2018). A multi-objective approach to welding shop scheduling for makespan, noise pollution and energy consumption. *Journal of Cleaner Production, 196*, 773–787.

Mansouri, S., Aktas, E., & Besikci, U. (2016). Green scheduling of a two-machine flowshop: Trade-off between makespan and energy consumption. *European Journal of Operational Research, 248*(3), 772–788.

May, G., Stahl, B., Taisch, M., & Prabhu, V. (2015). Multi-objective genetic algorithm for energy-efficient job shop scheduling. *International Journal of Production Research, 53*(23), 7071–7089.

Meng, R., Rao, Y., & Luo, Q. (2020). Modeling and solving for bi-objective cutting parallel machine scheduling problem. *Annals of Operations Research, 285*(1-2), 223–245.

Minella, G., Ruiz, R., & Ciavotta, M. (2011). Restarted iterated Pareto greedy algorithm for multi-objective flowshop scheduling problems. *Computers & Operations Research, 38*(11), 1521–1533.

Mori, M., Fujishima, M., Inamasu, Y., & Oda, Y. (2011). A study on energy efficiency improvement for machine tools. *CIRP Annals - Manufacturing Technology, 60*(1), 145–148.

Mouzon, G., Yildirim, M. B., & Twomey, J. (2007). Operational methods for minimization of energy consumption of manufacturing equipment. *International Journal of Production Research, 45*(18-19), 4247–4271.

Paolucci, M., Anghinolfi, D., & Tonelli, F. (2017). Energy-aware scheduling: a multi-objective extension of a scheduling support system for improving energy efficiency in a moulding industry. *Soft Computing, 21*, 3687–3698.

Pinedo, M. L. (2012). *Scheduling: Theory, Algorithms, and Systems* (4th). Springer.

Queiroz, T. A. d., & Mundim, L. R. (2020). Multiobjective pseudo-variable neighbor-

hood descent for a bicriteria parallel machine scheduling problem with setup time. *International Transactions in Operational Research, 27*(3), 1478–1500.
Safarzadeh, H., & Niaki, S. T. A. (2019). Bi-objective green scheduling in uniform parallel machine environments. *Journal of Cleaner Production, 210*, 559–572.
Sedgewick, R., & Wayne, K. (2011). *Algorithms* (4th ed.).
Shao, W., Pi, D., & Shao, Z. (2019). A pareto-based estimation of distribution algorithm for solving multiobjective distributed no-wait flow-shop scheduling problem with sequence-dependent setup time. *IEEE Transactions on Automation Science and Engineering, 16*(3), 1344–1360.
Shrouf, F., Ordieres-Meré, J., García-Sanchez, A., & Ortega-Mier, M. (2014). Optimizing the production scheduling of a single machine to minimize total energy consumption costs. *Journal of Cleaner Production, 67*, 197–207.
Tang, D., Dai, M., Salido, M., & Giret, A. (2016). Energy-efficient dynamic scheduling for a flexible flow shop using an improved particle swarm optimization. *Computers in Industry, 81*, 82–95.
Taticchi, P., Tonelli, F., & Pasqualino, R. (2013). Performance measurement of sustainable supply chains: a literature review and a research agenda. *International Journal of Productivity and Performance Management, 62*(8), 782–804.
Wang, S., Wang, X., Yu, J., Ma, S., & Liu, M. (2018). Bi-objective identical parallel machine scheduling to minimize total energy consumption and makespan. *Journal of Cleaner Production, 193*, 424–440. https://doi.org/10.1016/j.jclepro.2018.05.056.
Wang, Y., Wang, M., & Lin, S. (2017). Selection of cutting conditions for power constrained parallel machine scheduling. *Robotics and Computer-Integrated Manufacturing, 43*, 105–110.
Yeh, W.-C., Chuang, M.-C., & Lee, W. C. (2015). Uniform parallel machine scheduling with resource consumption constraint. *Applied Mathematical Modelling, 39*, 2131–2138.
Zeng, Y., Che, A., & Wu, X. (2018). Bi-objective scheduling on uniform parallel machines considering electricity cost. *Engineering Optimization, 50*(1), 19–36.
Zhang, H., Zhao, F., Fang, K., & Sutherland, J. W. (2014). Energy-conscious flow shop scheduling under time-of-use electricity tariffs. *CIRP Annals - Manufacturing Technology, 63*(1), 37–40.
Zhang, Q., & Li, H. (2007). MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on Evolutionary Computation, 11*(6), 712–731.
Zhang, R., & Chiong, R. (2016). Solving the energy-efficient job shop scheduling problem: a multi-objective genetic algorithm with enhanced local search for minimizing the total weighted tardiness and total energy consumption. *Journal of Cleaner Production, 112*, 3361–3375.
Zitzler, E., & Thiele, L. (1999). Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation, 3*(4), 257–271.