Fundamenta Informaticae XXI (2001) 1001–1041
DOI 10.3233/FI-2016-0000
IOS Press

Efficient approaches for solving a multiobjective energy-aware job shop scheduling problem

Miguel A. González * C
Department of Computing, University of Oviedo
Campus de Gijón, 33203 Gijón, Spain
mig@uniovi.es

Angelo Oddi†
ISTC-CNR
Via San Martino della Battaglia, 44, 00185 Rome, Italy
angelo.oddi@istc.cnr.it

Riccardo Rasconi†
ISTC-CNR
Via San Martino della Battaglia, 44, 00185 Rome, Italy
riccardo.rasconi@istc.cnr.it

Abstract. One of the most recent and interesting trends in intelligent scheduling is trying to reduce the energy consumption in order to obtain lower production costs and smaller carbon footprint. In this work we consider the energy-aware job shop scheduling problem, where we have to minimize at the same time an efficiency-based objective, as is the total weighted tardiness,

*Miguel A. González has been supported by the Spanish Government under research project TIN2016-79190-R and by the Principality of Asturias under grant FC-GRUPIN-IDI/2018/000176. We also thank Ying Liu for sending us the detailed results of his work.
CCorresponding author
†ISTC-CNR authors were supported by the ESA Contract No. 4000112300/14/D/MRP "Mars Express Data Planning Tool MEXAR2 Maintenance".
Address for correspondence: Miguel A. González. Campus de Viesques. Edificio Departamental Oeste, módulo 1. Despacho 1.1.03. 33203 Gijón. Spain.

and also the overall energy consumption. We experimentally show that we can reduce the energy consumption of a given schedule by delaying some operations, and to this end we design a heuristic procedure to improve a given schedule. As the problem is computationally complex, we design three approaches to solve it: a Pareto-based multiobjective evolutionary algorithm, which is hybridized with a multiobjective local search method and a linear programming step, a decomposition-based multiobjective evolutionary algorithm hybridized with a single-objective local search method, and finally a constraint programming approach. We perform an extensive experimental study to analyze our algorithms and to compare them with the state of the art.

Keywords: Multiobjective optimization, job shop, energy, metaheuristics

1. Introduction

This work addresses an energy-aware variant of the classical Job Shop Scheduling Problem (JSP). Its study is highly relevant due to the fact that it finds numerous applications in manufacturing and is central to many supply chain problems that integrate production planning and scheduling [1]. The problem is NP-hard [2] thus very challenging, and so it has been studied for decades.

The makespan is clearly the objective function that received the most attention over the years. However, due date related objective functions are usually more important in real applications. In fact, several research papers conclude that meeting due dates is the most important scheduling objective in competitive markets [3, 4]. The total weighted tardiness (TWT) is a particularly interesting objective function, as we may assign different priorities to different jobs.

Several reasons motivate the study of energy considerations in scheduling problems, for example the increasing price of energy or the need for reducing carbon footprint. Energy-saving practices are remarkably important in modern industries, both for economical and environmental reasons.

Unfortunately, it may occur that the improvement in energy costs is obtained at the cost of losing solution quality, and therefore we face a multi-objective scheduling problem. As this is a problem often found in real environments, there is an increasing interest in multi-objective optimization in scheduling problems, and in the use of metaheuristic algorithms to solve them [5].

In the literature we can find many research papers proposing solving approaches for the JSP with TWT minimization and/or energy considerations (see Section 2).

In this work we propose the following approaches to minimize both the energy consumption and the TWT in a JSP.

1. A dominance-based evolutionary algorithm, using the NSGA-II framework [6]. We describe a mechanism to penalize repeated individuals, thus improving diversity. The genetic algorithm is hybridized with a multi-objective local search method.

2. A decomposition-based evolutionary algorithm, using the MOEA/D framework [7], which is hybridized with a single-objective local search method. MOEA/D variants are much less used than genetic algorithms in combinatorial optimization, but its study is also interesting as it is a very powerful framework.

3. A constraint programming approach that solves a model of the problem by using the ϵ-constraint method [8].

Both the NSGA-II and MOEA/D based approaches implement a couple of methods to optimize the energy consumption of a given solution: a fast, low-polynomial heuristic procedure to be used every time a solution is evaluated, and an exact, linear programming step, more computationally costly and thus only applied to the best solutions obtained at the end of the execution.

We present an experimental study to analyze the proposed approaches and to compare them with the state of the art.

Preliminary versions of this work have been already published in the Proceedings of the 27th International Conference on Automated Planning and Scheduling (ICAPS 2017), in the Proceedings of the 11th Workshop on Constraint Satisfaction Techniques for Planning and Scheduling (COPLAS 2016), and also presented in the 24th RCRA International Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion. In these preliminary works the NSGA-II based method and a simpler version of the constraint programming approach are described. The main contributions and improvements with respect to those previous works are enumerated in the following:

- We have proposed a new multi-objective evolutionary algorithm based on the MOEA/D framework.
- We have improved the experimental study by considering a larger number of instances with different characteristics, as in previous research we only considered four variants of the small FT10 instance.
- We introduce a modified version of the constraint programming model originally presented in [9] that improves performances.
- We report an extensive literature review that includes many recent papers about energy-aware scheduling problems.
- We also provide more comprehensive descriptions of the proposed approaches, graphical examples, and more detailed conclusions.

The structure of this paper is as follows. In Section 2 we present some recent papers about solving energy-aware scheduling problems. Then, Section 3 describes the scheduling problem while also providing some examples. In Sections 4 and 5 we define our NSGA-II and MOEA/D based evolutionary algorithms, respectively, whereas in Section 6 we describe the constraint programming approach. Section 7 reports the results of an extensive experimental study where we analyze our proposals. Finally, Section 8 summarizes the conclusions.

2. Literature review

The classical JSP has been studied for decades, but most works focused on the makespan objective function (see for example [10], [11] or [12]). However, the interest in other objective functions has

grown over the years. The first approaches that considered TWT minimization in the JSP were a branch and bound algorithm [13], shifting bottleneck [14] and a large step random walk heuristic [15]. More recent papers include genetic local search [16], local search [17], shifting bottleneck hybridized with tabu search [18] or genetic algorithm combined with tabu search [19]. The work [20] is also interesting, as the authors study some novel neighborhood structures for the problem.

In these last years, interest in energy-aware scheduling problems has exponentially increased. Many different scheduling problems are studied, for example single machine problems [21, 22, 23], flexible JSP [24, 25, 26, 27, 28, 29, 30, 31, 32, 33], flow shop [34, 35, 36], flexible flow shop [37], two-machine sequence dependent permutation flow shop [38], batch-processing machine scheduling problem [39], hybrid flow shop [39], flexible flow shop [40], uniform parallel machine [41], unrelated parallel machine [42], dynamic JSP [43], the dual-resource constrained JSP with interval processing time and heterogeneous resources [44] or the classical JSP [45, 46, 47, 48, 49, 50, 9, 51]. Another example is [52], where the authors study the Energy-Constrained Scheduling Problem with Continuous Resources, which is a generalization of the well-known cumulative scheduling problem.

There are also papers that report the utilization of intelligent scheduling approaches in real applications. For example, in [53] the authors implement their scheduling system at a manufacturing company located in Tucson, Arizona state of USA, and prove that it achieves energy cost savings without sacrificing the productivity. Other real applications are related to manufacturing of cast iron plates [35], recycling carbon fiber reinforced polymer [24], a metalworking workshop in a plant [37], animal feed manufacturing and home energy management systems [54] or a moulding industry, which involved a major producer of plastic dispensers [55].

About the objective functions, most papers try to optimize two objectives at the same time, one of them is efficiency-related (makespan [24, 25, 30, 32, 33, 38, 44, 42, 46, 51, 37], TWT [45, 48, 39, 50, 47, 9], total completion time [21, 22], total late work [29], or cycle time [34]) and the other objective is related to the energy consumption of the schedule. However, some papers optimize additional objectives, as for example the total setup [55] or the total availability of the system [28], while others impose a restriction on peak power consumption, in addition to the traditional time-based objectives [35]. Another example of a third objective function is the numbers of turning-on/off machines [27], total workload of machines [32], noise reduction [26] or peak power [34, 39]. Some papers also consider robustness objectives, as for example [43], where given a disruption, the main goal is to reschedule the minimum number of operations so that the energy consumption is minimized and the makespan does not increase.

As for how to tackle several objectives at once, some authors opt to use a lexicographical approach [44], whereas others impose an upper limit to some objective and minimize the other [35], but most papers propose to build the Pareto Front [45].

Regarding the energy model, some papers consider that the machine cannot be switched off when idle [45, 50, 9], whereas other papers consider the possibility of turning the machines on and off [21, 27, 48, 31], or even to choose the speed level of the machines, of course consuming more energy when working faster [34, 27, 26, 47, 49, 43, 38, 42]. Some papers also consider that machines can switch to stand-by state, aside from turning it on and off [46, 51]. Also, while most papers consider fixed energy costs, some papers consider the presence of time-of-use electricity prices [36], a time-of-use policy [33] or variable energy prices [54, 23].

As these scheduling problems are really complex, exact methods are able to tackle small instances, however metaheuristics are recommended when solving real-life sized instances. In the literature we can find all kinds of approaches to solve energy-aware scheduling problems, the most common are probably multi-objective genetic or memetic algorithms [45, 46, 27, 43, 29, 24, 26, 22, 39, 49, 50, 9]. However, many more different approaches can be found, as for example: ant colony optimization metaheuristic [36], particle swarm optimization algorithm [41], dynamical neighborhood search [44], constraint programming approach [9, 51], memetic differential evolution [42], hybrid fruit fly optimization algorithm [25], nested partitions algorithm [30], biogeography-based optimization algorithm combined with variable neighborhood search [33], genetic-simulated annealing [37], a dynamic game theory based two-layer scheduling method [32], dispatching rules [21], mixed integer programming model [40], mathematical programming model [21, 34]. In fact, some papers propose both exact (mathematical programming) and combinatorial approaches [35, 31, 23] in order to be able to optimally solve small instances and also give decent solutions to large instances.

As we have seen, although there are a huge number of works about energy-aware scheduling, each of them solves a specific problem with a specific energy model, set of constraints and number and type of objective functions. Therefore, most of them cannot naturally be compared with the approaches presented in this work, with the exception of [45] and [9], which solve exactly the same problem. In fact, [9] is the preliminary version of the research described in this paper, where the NSGA-II and a simpler version of the constraint programming approach are described.

In [45] the authors describe a basic NSGA-II to solve the problem. We have to remark that our NSGA-II approach has several differences with respect to it: for example its crossover operator, elimination of repeated solutions in the replacement strategy, hybridization with a local search method, the additional energy optimization procedure included in the schedule builder, and the final linear programming step. We will see that these additional components lead to a much better overall performance.

3. Problem formulation

The JSP consists of processing a set of N jobs, $J = \{J_1, \ldots, J_N\}$, in a set of M resources or machines, $R = \{R_1, \ldots, R_M\}$. A job J_i consists of a sequence of n_i tasks or operations (θ_{i1}, ..., θ_{in_i}), each requiring the uninterrupted and exclusive use of a given machine during all its processing time. The goal here is to minimize some objective functions subject to a set of constraints. The precedence constraints indicate the sequence of machines for each job, which is prescribed. Also, each machine can process at most one operation at a time, due to the capacity constraints. We define two additional parameters for each job: its due date, which is a time before which all its operations should be processed, and its weight, which represents its priority or importance. It is also necessary to define the idle power level of each machine, in order to calculate its energy consumption. We denote by:

- Ω : the set of all operations.
- d_i : due date of job J_i.

- w_i : weight of job J_i.
- P_k^{idle} : idle power level of machine or resource R_k.
- $m_{\theta_{ij}}$: machine or resource required by operation θ_{ij}.
- $p_{\theta_{ij}}$: processing time of operation θ_{ij}.
- $s_{\theta_{ij}}$: starting time of operation θ_{ij} (that we need to determine).

The goal of this scheduling problem is to obtain a feasible schedule, which can be formally defined as an assignment of a starting time $s_{\theta_{ij}} \geq 0$ for each $\theta_{ij} \in \Omega$ such that the following constraints are met:

$$s_{\theta_{ij}} + p_{\theta_{ij}} \leq s_{\theta_{ij+1}} \quad \forall i \in \{1, \ldots, N\} \quad \forall j \in \{1, n_i - 1\} \tag{1}$$

$$(s_{\theta_{ij}} + p_{\theta_{ij}} \leq s_{\theta_{kl}}) \vee (s_{\theta_{kl}} + p_{\theta_{kl}} \leq s_{\theta_{ij}}) \quad \forall i, j, k, l \text{ such that } m_{\theta_{ij}} = m_{\theta_{kl}} \text{ and } \theta_{ij} \neq \theta_{kl} \tag{2}$$

As we have seen, the JSP has two types of binary constraints. (i) *Precedence constraints* are defined by the sequential routings of the operations within a job, and translate into linear inequalities (Equation 1). (ii) *Capacity constraints* limit the use of each machine to one operation at a time, and translate into disjunctive constraints (Equation 2).

In the following, in order to simplify expressions, we will denote operations by a single letter whenever possible, instead of using θ_{ij}. Also, given a feasible schedule, we denote by PJ_v and SJ_v the predecessor and successor of v in the job sequence, respectively, whereas we denote by PM_v and SM_v the predecessor and successor of v in its machine sequence.

In particular, we are looking for feasible schedules that minimize two objective functions at the same time: the TWT and the energy consumption. The TWT is defined as follows:

$$\sum_{i=1,\ldots,N} w_i T_i \tag{3}$$

where T_i is the tardiness of job i, given by $T_i = \max\{C_i - d_i, 0\}$, being C_i the completion time of job i.

We borrow the energy consumption model from [45], where it is assumed that it is not possible to turn the machines off when idle. It is easy to prove that in a job shop environment minimizing the total consumption can be reduced to minimize the total non-processing energy (NPE), which is the time a machine is idle (i.e., not processing any job). The reason is that in every possible schedule, all machines have to process the same fixed set of operations, and therefore the total processing energy must be the same (that is, if we do not consider setup times between operations, which could be an interesting research line for future work).

Therefore, our energy objective function will be to minimize the sum of all the NPE consumed by all machines to process a given job schedule. The total NPE is defined as:

$$\sum_{k=1,\ldots,M} [P_k^{idle} \times (s_{\omega_k} + p_{\omega_k} - s_{\alpha_k} - \sum_{u \in M_k} p_u)] \tag{4}$$

where α_k and ω_k are respectively the first and last operations on machine R_k in the given schedule, and M_k is the set of all operations that must be executed in machine R_k.

Generally speaking, for a minimization problem with f_i, $i = 1, \ldots, n$ objective functions, a given solution S is said to be *dominated* by another solution S' (denoted $S' \succ S$) if and only if for each objective function f_i, $f_i(S') \leq f_i(S)$ and there exists at least one i such that $f_i(S') < f_i(S)$. Then, the goal of the described scheduling problem will be to find a set of non-dominated solutions with respect to TWT and NPE, the so-called *Pareto set*.

We have to remark that the described energy model is quite simple, and it is based on the assumption that it is not possible to turn off the machines when idle. In some real applications energy can be saved by turning off machines, but in others it will not be possible, either because the machines simply cannot be switched off, or because the process of switching them off and then on again consumes more energy than keeping them idle ([46, 51]).

In Sections 2 and 8 we discuss some possible more complicated energy models.

In [56] it is defined that a regular performance measure is one such that its value can only be increased by increasing at least one of the completion times in the schedule. In order to minimize a regular performance measure it is enough to consider "left-shift schedules", which are schedules built from a partial ordering of the operations, in such a way that we assign to each operation its earliest possible starting time, following the given ordering. It is easy to see that the TWT is a regular performance measure, but the NPE is not. Notice that the NPE can be increased if we are able to decrease the starting time of the first operation of a machine, while maintaining all other operations intact, and also it can be decreased by delaying the starting time of some operations. As stated in [57], research on non-regular objective functions has always been isolated and scattered, compared to regular objectives, as they are more difficult to minimize.

3.1. Example instance

For illustrative purposes, we are going to describe a small example instance with 3 jobs and 3 machines. Each job consists of 3 operations, and their processing times and machine requirements are detailed in Table 1.

Table 1. Processing times and machine requirements of the operations of the example instance.

	First operation		Second operation		Third operation	
	Proc. time	Machine	Proc. time	Machine	Proc. time	Machine
Job 1	4	R_1	4	R_2	2	R_3
Job 2	2	R_1	5	R_3	3	R_2
Job 3	3	R_2	7	R_1	3	R_3

The due dates are $d_1 = 16$, $d_2 = 12$ and $d_3 = 14$, whereas the weights are $w_1 = 3$, $w_2 = 2$ and $w_3 = 1$. Finally, the idle power level of the machines are $P_1^{idle} = 1$, $P_2^{idle} = 2$ and $P_3^{idle} = 3$.

Figure 1 shows the Gantt chart of one possible feasible schedule for the described example instance.

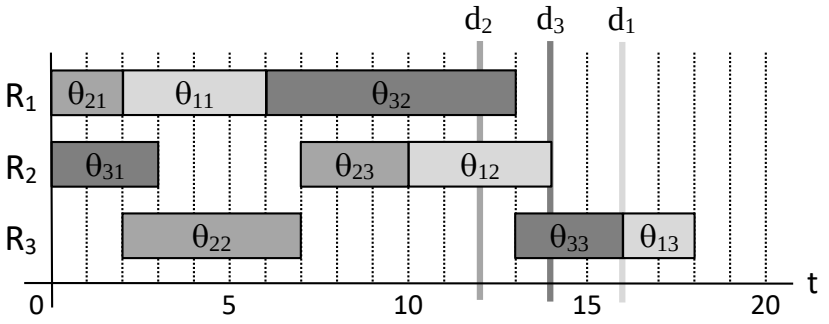


Figure 1. A feasible schedule for the instance described in Section 3.1, with TWT=8 and NPE=26.

Notice that the last task of job J_1 ends at time 18, and its due date is 16 whereas its weight is 3, therefore it adds $(18-16)\times 3 = 6$ to the TWT of the schedule. Job J_2 adds nothing to the TWT, as it ends two time units before its due date of 12, hence having zero tardiness. Finally, job J_3 ends at time 16, having a due date of 14 and a weight of 1, and so it adds $(16-14)\times 1 = 2$ to the TWT. In summary, the TWT of the schedule is $6+0+2=8$.

About the NPE calculation, machine R_1 adds nothing to it because it has no idle intervals between the start of its first operation and the end of its last operation. Machine R_2 has an idle interval of 4 time units between tasks θ_{31} and θ_{23}, so given that $P_2^{idle} = 2$ it adds $4\times 2 = 8$ to the NPE. Finally, machine R_3 has an idle interval of 6 time units between tasks θ_{22} and θ_{33}, and $P_3^{idle} = 3$, so it adds $6\times 3 = 18$ to the NPE. Overall, the NPE of the schedule is $0+8+18 = 26$.

3.2. The disjunctive graph model representation

A common representation in scheduling problems is the disjunctive graph. For the JSP with TWT minimization we follow a similar representation to that already used in several papers [15, 17, 16, 19, 20]. In particular, it is a directed graph $G = (V, A \cup D)$. Each node of set V represents an operation, with the exception of the dummy nodes *start* and end_i $1 \leq i \leq N$, which represent fictitious operations with zero processing time and that do not require any machine. A is a set of arcs (denoted *conjunctive arcs*) that represent precedence constraints between operations of each job. The set A also contains additional arcs from node *start* to the first operation of each job, and arcs from the last operation of each job i to its corresponding end_i node. D is a set of arcs (denoted *disjunctive arcs*) that represent capacity constraints. Set D is partitioned into subsets D_j, with $j = 1, \ldots, M$, where D_j corresponds to machine R_j and includes two directed arcs (v, w) and (w, v) for each pair v, w of operations that require machine R_j. Arcs (v, w) in A and in D are weighted with the processing time of the operation at the source node, p_v (as already defined, $p_{start} = 0$ and $p_{end_i} = 0$).

A feasible schedule S can be represented by an acyclic subgraph of G: $G_S = (V, A \cup H)$, where $H = \cup_{j=1\ldots M} H_j$. H_j is a minimal subset of arcs from D_j defining a processing order for all operations requiring machine R_j. Therefore, obtaining a solution to the problem can be reduced to finding compatible orderings H_j, or partial schedules for each machine, that translate into a solution graph G_S without any cycle. As an example, Figure 2 shows the disjunctive graph corresponding to

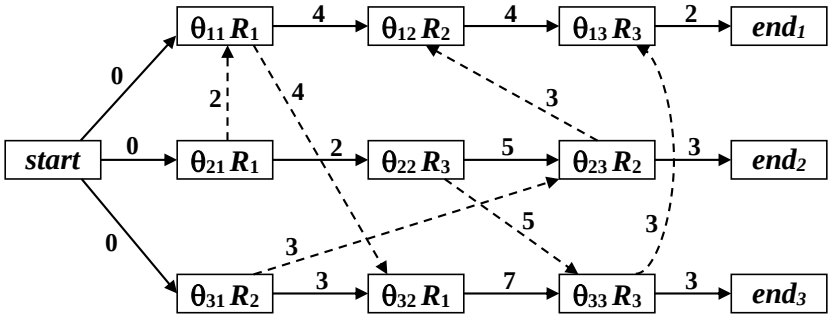


Figure 2. Disjunctive graph corresponding to the feasible solution depicted in Figure 1.

the feasible schedule depicted in Figure 1; dotted arcs belong to set H, while continuous arcs belong to set A.

Given the acyclic subgraph G_S, the TWT of the schedule is determined by a set of critical arcs. A critical path is a largest cost path from node $start$ to a node end_i $1 \leq i \leq N$. The length of these paths represents the completion time of operation end_i (and hence the completion time of job J_i), which can be used to calculate the contribution of job J_i to the TWT. Nodes and arcs in a critical path are also termed critical. A critical path may be decomposed as a sequence $start, B_1, \ldots, B_r, end_i$, $1 \leq i \leq N$, where each B_k, $1 \leq k \leq r$, is termed critical block, defined as a maximal subsequence of consecutive operations in the critical path that require the same machine.

These definitions are relevant, as many solution methods, formal properties and neighborhood structures proposed in the literature for the JSP are based on the concepts of critical path and critical block. In fact, the neighborhood structure used in this work relies on reversing the processing order of two consecutive operations in a critical block, as similarly done in [58] and many other works.

Although the disjunctive graph representation is useful for minimizing TWT, it is unfortunately not as useful for NPE minimization, as this performance measure is not directly related to finding largest cost paths in a graph representation.

4. Dominance-based multi-objective evolutionary algorithm

In this section we describe a dominance-based hybrid method, taken from [9], that combines a modified NSGA-II multi-objective genetic algorithm with a multi-objective hill climbing local search method and a linear programming approach.

Several papers have already tackled the minimization of non-regular objectives in the JSP. An interesting example is [59], where the authors propose to decompose the overall problem in two subproblems: sequencing and timing. In our work (both in the dominance-based approach described in this Section and also in the decomposition-based approach described in Section 5) we follow a similar approach, and so we represent the solutions as permutations in order to solve the sequencing subproblem. Then, to solve the timing subproblem we introduce two approaches: a low-polynomial energy post-optimization procedure when evaluating each and every solution, and a more computationally expensive optimal linear programming approach, applied only to the final set of non-dominated solutions

returned by the evolutionary algorithm.

4.1. Genetic algorithm

The basis of our dominance-based evolutionary algorithm is the well-known NSGA-II template [6]. Firstly, an initial population set Pop_0 of size $popSize$ is created at random and evaluated. Then, the algorithm iterates over $numGen$ generations. At each generation i a set of offspring solutions $\mathrm{Off}(Pop_i)$ is built from the current one Pop_i by applying selection, crossover and mutation operators, and finally a replacement strategy is applied to select the solutions that will form the next population Pop_{i+1}. At the end of the execution, the set of non-dominated solutions present in the population is returned as an approximation of the Pareto set.

4.1.1. Representation and evaluation of chromosomes

We codify solutions into chromosomes using permutations with repetitions [60]. This is a permutation of the set of operations, each denoted by its job number, which represents a linear ordering compatible with precedence constraints. As an example, consider a problem instance with 3 jobs: $J_1 = \{\theta_{11}, \theta_{12}, \theta_{13}\}$, $J_2 = \{\theta_{21}, \theta_{22}, \theta_{23}\}$, $J_3 = \{\theta_{31}, \theta_{32}, \theta_{33}\}$, then the ordering of operations $\pi = \{\theta_{21}, \theta_{11}, \theta_{22}, \theta_{31}, \theta_{23}, \theta_{32}, \theta_{33}, \theta_{12}, \theta_{13}\}$ is represented by the chromosome $v =$ (2 1 2 3 2 3 3 1 1).

To evaluate a given chromosome we generate its associated schedule and then we compute its TWT and NPE. In particular, we use an insertion strategy following the sequence given by the chromosome. Each operation is scheduled at the earliest possible starting time such that all constraint are met, leaving previously scheduled operations intact. The main advantage of using permutations with repetitions is that every "correct" permutation (i.e., as many repetitions of a given number as number of operations of the corresponding job) can be translated into a feasible solution.

As an example, in the instance described in Section 3.1, the chromosome $v_1 =$ (2 1 2 3 2 3 3 1 1) would be translated as the feasible schedule represented in Figure 1.

Notice that a single schedule might be represented by several different chromosomes, as actually the permutation is a linearization of the partial order of operations and there might be more linear orderings. For example, the chromosome $v_2 =$ (3 2 1 2 2 3 3 1 1) would also be translated as the feasible schedule represented in Figure 1. Although this is not a desirable property, the benefits of this representation more than compensate for it, and so it is used in most literature about the JSP.

4.1.2. Energy post-optimization procedure

The insertion strategy described for evaluating a chromosome would be enough to produce a Pareto optimal schedule (provided the appropriate chromosome) if the objective functions were regular. Unfortunately, the NPE is a non-regular performance measure, and hence scheduling each operation as soon as possible is sometimes not the best option. Clearly, we can improve the NPE of a machine R_k by delaying the starting time of its first operation (s_{α_k}) without delaying the starting time of its last operation (s_{ω_k}). Notice that it would also be possible if we decrease s_{ω_k} without increasing s_{α_k}, but

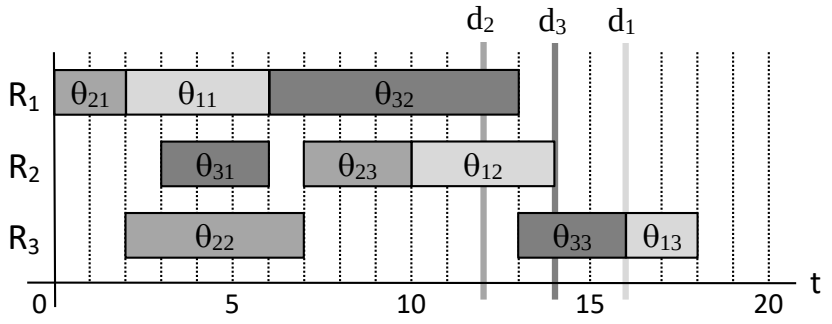


Figure 3. The solution of Figure 1 after applying the energy post-optimization procedure described in Section 4.1.2. Notice the delay of operation θ_{31}. The TWT is still 8 but the NPE is reduced from 26 to 20.

this approach cannot be used in our algorithm, as operations are initially scheduled as soon as possible by the insertion strategy.

In order to solve the timing subproblem we propose an energy post-optimization procedure that, maintaining the operation ordering of a given schedule, tries to reduce its NPE while not increasing its TWT. The procedure is detailed in Algorithm 1. The basic idea is to delay as much as possible all operations of each machine (except the last one), while at the same time taking care when delaying the last operation of each job, so that the TWT does not increase. Evidently, when delaying operations we always take into account the precedence constraints between operations of the same job, so that the final schedule is feasible.

Clearly, as we do not increase the starting time of the last operation of each machine, the resulting NPE after applying the procedure is always lower than or equal to the NPE of the original schedule. Additionally, the resulting TWT is not increased either, due to the fact that the completion time of the last operation of a job may only be delayed if it is lower than the due date of the corresponding job, and in this case it is delayed at most up to this due date.

The described procedure is embedded in the solution evaluation method, and is executed just after the insertion strategy builds the initial schedule. Hence, it is applied when evaluating every chromosome generated by the genetic algorithm and every neighbor considered in the local search. Adding this procedure to the scheduler increases the running time of the evolutionary algorithm between 10% and 25%, however the quality improvement of the reached solutions is very substantial and so it more than compensates the increase in running time, as we will see in Section 7.

As an example, applying this energy post-optimization procedure to the schedule represented in Figure 1 would lead to the schedule represented in Figure 3, which has the same TWT of 8 but the NPE is reduced from 26 to 20 (as machine R_2 adds 2 whereas machine R_3 adds 18), thanks to the delay of operation θ_{31}.

4.1.3. Genetic operators

The purpose of the selection phase is to select which chromosomes will undergo crossover and mutation. Here we use a tournament strategy: we randomly select $tSize$ chromosomes from the population,

Algorithm 1: The energy post-optimization procedure

input : A problem instance I and a feasible schedule (i.e., an ordering O and a set of starting times s)
output: A new set of starting times s' for the ordering O
begin
  $k \leftarrow |\Omega|$;
  while $k \geq 1$ **do**
    $a \leftarrow O[k]$;
    if *a is the last operation processed in a machine* **then**
      $s'_a \leftarrow s_a$;
    else
      if *a is the last operation of its job* **then**
        $j \leftarrow$ job of operation a;
        $s'_a \leftarrow \min\{\max\{d_j, s_a + p_a\}, s'_{SM_a}\} - p_a$;
      else
        $s'_a \leftarrow \min\{s'_{SJ_a}, s'_{SM_a}\} - p_a$;
      end
    end
    $k \leftarrow k - 1$;
  end
end

and the first parent will be the best of them (according to non-domination rank and crowding distance, as described in Section 4.1.4). The second parent is selected in the same way, and then the crossover operator is applied with probability $crProb$, producing two offspring solutions.

We choose as crossover operator the well-known Job Order Crossover (JOX) [60], that given two parents it produces two offspring solutions. It selects a random subset of jobs and copies their genes to the first offspring in the same positions as they are in the first parent, whereas the remaining genes are taken from the second parent maintaining their relative ordering. To create the second offspring the parents reverse their role. Finally, we mutate each offspring with probability $mutProb$ using the swap mutation operator, that consists on swapping two random positions of the chromosome.

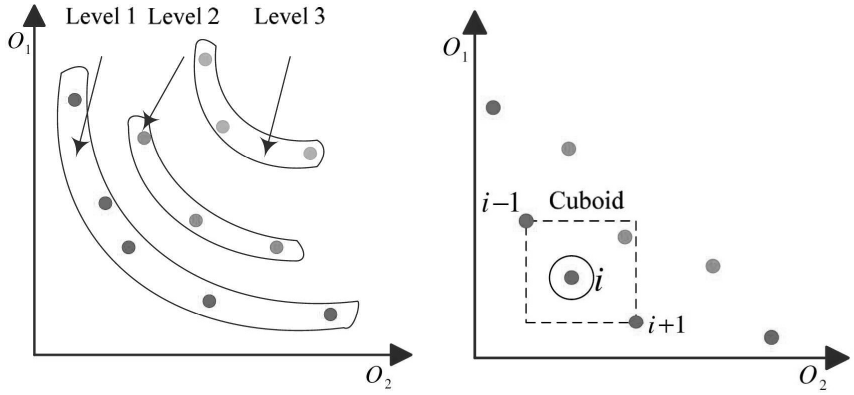
4.1.4. Replacement strategy

The new population Pop_{i+1} is generated by combining the previous population Pop_i and the population $\text{Off}(Pop_i)$ that results from applying selection, crossover and mutation operators to Pop_i. In this work we adopt a replacement strategy based on the non-dominated sorting approach with diversity preservation proposed in [6].

Firstly, we assign to each individual j in the pool $Pop_i \cup \text{Off}(Pop_i)$ a non-domination rank ($rank(j)$), thus sorting the pool of solutions into different non-domination levels. In order to do this, all non-dominated solutions in the pool are assigned level 1, then level 2 is formed by all the non-

dominated solutions of the pool after removing those solutions from level 1, and so on (see Figure 4(a)).

Then, we assign to each individual j in the pool a crowding distance ($dist(j)$) (see Figure 4(b)), which estimates the density of solutions "near" each individual from its same non-domination level. As described in [6], to get an estimate of the density of solutions surrounding a particular solution in its same non-domination level, we calculate the average distance of two points on either side of this point along each objective. This quantity serves as an estimate of the perimeter of the cuboid formed by using the nearest solutions as the vertices (call this the *crowding distance*). In Figure 4(b), the crowding distance of the ith solution in its front (marked with dark circles) is the average side length of the cuboid (shown with a dashed box). The crowding distance computation requires sorting the solutions in the non-domination level according to an objective function value in ascending order. Then, the boundary solutions (solutions with smallest and largest function values) are assigned an infinite distance value. All other intermediate solutions are assigned a distance value equal to the absolute normalized difference in the function values of two adjacent solutions. The overall crowding-distance value is calculated as the sum of individual distance values corresponding to each objective. Each objective function is normalized before calculating the crowding distance. After all solutions are assigned a distance metric, we can compare two solutions for their proximity with other solutions. A solution with a smaller value of this distance measure is more crowded by other solutions.



(a) Non-domination levels.

(b) Crowding distance calculation between solutions in the same non-domination level.

Figure 4. The replacement strategy of the multi-objective genetic algorithm.

Finally, the next population Pop_{i+1} is formed by the best $popSize$ solutions from the pool $Pop_i \cup \mathrm{Off}(Pop_i)$, according to the lexicographical ordering defined by $(rank, dist)$. Hence, we prefer solutions that belong to a lower (better) non-domination rank. When choosing between two solutions with the same non-domination rank, we prefer that located in the less crowded region, because it is more desirable to obtain a good spread of different solutions instead of many similar ones (see [6]).

We have experimentally observed that the performance of the algorithm can be substantially improved by adding an additional step to improve the diversity of the population. In particular, we propose to firstly remove from the pool $Pop_i \cup \mathrm{Off}(Pop_i)$ those solutions which are repeated, i.e.,

there exists in the pool at least another solution having the same values for all objective functions. The idea is to add to the next population Pop_{i+1} the best $popSize$ solutions based on $(rank, dist)$ after we perform the described elimination of repeated solutions.

In case that the elimination causes the pool to contain less than $popSize$ solutions, then all the unique individuals are added to Pop_{i+1}, which is then completed with some of the repeated solutions by recursively using the same strategy. This is, we consider the pool formed by the repeated solutions we have eliminated before, and we first remove the repeated solutions from that pool, and then add to Pop_{i+1} the best solutions from the resulting pool, according to $(rank, dist)$. If Pop_{i+1} still have less than $popSize$ solutions, we apply again the same strategy, and so on.

4.2. Local search

Local search methods are frequently hybridized with evolutionary metaheuristics, as the former provide exploitation while the latter focuses on efficient exploration of the search space. Local search is essentially a single-objective optimization method, and so the main issue when designing a multi-objective memetic algorithm is how to implement the local search. In fact, the amount of existing multi-objective local search algorithms is still reduced [61]. Some examples are PAES (Pareto Archived Evolution Strategy) [62] or Pareto Local Search [63], but these Pareto-based local search methods are too computationally costly to be hybridized with a genetic algorithm. In this work we propose a local search procedure that is much less time-consuming that, instead of building a Pareto front, it provides a single output solution, or "one-point iteration" [64].

The selection criterion for choosing the best neighbor is also a non-trivial issue when applying local search to a multi-objective problem, as the dominance relation $\succ$ only defines a partial order. Some authors propose to scalarize the objective function vector in order to guide the search [65, 66], whereas other authors define dominance-based acceptance criteria [62].

We propose a local search method based on hill climbing which is fast and efficient. The neighbor selection is based on dominance, but it also considers the solutions in the current non-dominated set of solutions present in the population of the genetic algorithm. In this way, we can select a very interesting neighbor even if it does not dominate the current solution.

Our proposal is detailed in Algorithm 2. It starts from an initial solution provided by the evolutionary algorithm. Then it generates neighbors of that initial solution one by one. They are evaluated using the scheduler and the energy post-optimization procedure described in Sections 4.1.1 and 4.1.2 respectively, until a neighbor that fulfills at least one of the following two requirements is found:

1. The neighbor dominates the current solution.

2. The neighbor would be included in the current set of non-dominated solutions of the population of the genetic algorithm (i.e., no solution of the population dominates the neighbor and also no solution has the same fitness values as the neighbor), while the current solution would not be included.

When one such neighbor is found, the current solution is swapped for the newly found solution and the process is repeated. In case that no such neighbor exists then the procedure ends and the

Algorithm 2: Multi-objective hill climbing local search

```
input  : A problem instance I and a feasible schedule S
output: The (hopefully) improved solution S' for I
begin
    S' ← S; continue ← True;
    while continue = True do
        NeighborSelected ← False;
        N(S') ← neighborhood of S';
        k ← 1;
        while NeighborSelected = False and k ≤ |N(S')| do
            S'' ← N(S')[k];
            Evaluate S'';
            if S'' dominates S', or S'' would enter in the current set of non-dominated
              solutions of the genetic algorithm and S' would not then
                NeighborSelected ← True;
            end
            k ← k + 1;
        end
        if NeighborSelected = False then
            continue ← False;
        else
            S' ← S'';
        end
    end
end
```

current solution is returned. When the procedure ends, the chromosome is rebuilt from the improved schedule returned, so its characteristics can be inherited to subsequent offspring solutions, an effect known as Lamarckian evolution.

The second requirement allows our local search to select interesting neighbors even if they do not dominate the current solution. Notice that if the current solution would already be included in the set of non-dominated solutions of the genetic algorithm, then we restrict the search to dominating neighbors only; it is easy to see that not doing so would result in an inefficient search process.

As this local search is based on hill climbing, it is actually not very computationally costly, and so it can be applied to all initial chromosomes and to all the generated offspring of the genetic algorithm. This would not be possible for other alternatives such as the Pareto Local Search [63], as their computational cost is much greater.

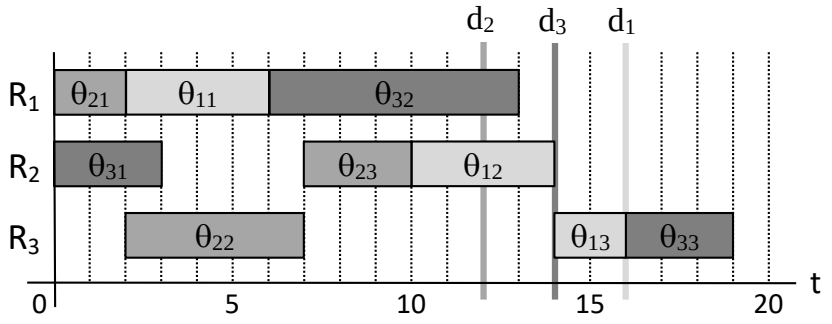


Figure 5. A neighbor of the schedule represented in Figure 1, created by reversing the critical arc (θ_{33}, θ_{13}). The TWT of 5 is better than that of the original schedule but the NPE of 29 is worse.

4.2.1. Neighborhood structure

Most neighborhoods that have been proposed in the literature for the JSP rely on the concepts of critical block and critical path, described in Section 3.2. In this work we adopt a neighborhood structure based on reversing a single critical arc of the schedule. It was initially proposed in [58] and it has some interesting properties, as for example that it always generates feasible neighbors, therefore avoiding the need to use expensive repairing procedures.

Notice that in TWT minimization the cost of a solution can be given by up to N critical paths. As indicated in [19], for each node end_i we consider a critical path from the node *start* to end_i when its length is greater than the due date of job J_i (i.e., d_i). It is well-known that the most time consuming part of a local search is the neighbor evaluation. As similarly proposed in [19], we opted to only consider the critical path of the job J_i that has the highest contribution to the TWT, in order to limit the number of neighbors and so limit the computational burden.

As an example, consider the schedule represented in Figure 2. Its TWT is 8, as job J_1 adds 6 and job J_3 adds 2 (see the Gantt chart of Figures 1 or 3). As job J_1 adds the most to the TWT, we choose the critical path that ends in node end_1, which is the following: (θ_{21}, θ_{11}, θ_{32}, θ_{33}, θ_{13}). There are two critical blocks in this critical path: (θ_{21}, θ_{11}, θ_{32}) and (θ_{33}, θ_{13}). Therefore, there are three possible neighbors: the first one created by reversing the critical arc (θ_{21}, θ_{11}), the second one reversing (θ_{11}, θ_{32}) and the third one reversing (θ_{33}, θ_{13}). As an example, Figures 5 and 6 show the schedule of the third neighbor, respectively before and after applying the energy post-optimization procedure. As the only late job is J_3, its TWT is 5, which is better than that of the original solution (8), but in this case the NPE is worse (23 versus 20 after applying the energy post-processing procedure or 29 versus 26 before applying it).

This neighborhood structure is specifically designed for TWT minimization, but we have empirically seen that many neighbors that improve the TWT of the original solution also improve its NPE as well (although this does not happen in the provided example).

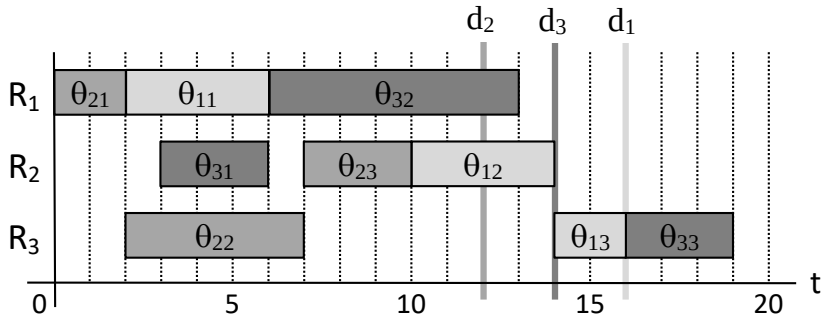


Figure 6. The schedule of Figure 5 after applying the energy post-optimization procedure to it. Notice the delay of operation θ_{31}, which is able to reduce the NPE from 29 to 23.

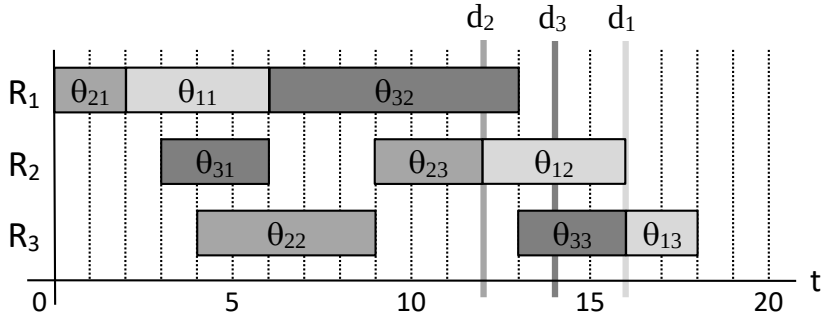


Figure 7. The NPE of the schedule represented in Figure 3 can be further improved if we allow delaying the last operation of a machine. By delaying operations θ_{12}, θ_{23} and θ_{22} the TWT is still 8 but the NPE is further reduced from 20 to 18.

4.3. The linear programming approach

The heuristic energy post-optimization procedure defined in Section 4.1.2 is fast and efficient, but it does not necessarily produce the optimal assignment of starting times. Notice that the NPE could be further improved in some cases even if we maintain the processing ordering of the operations on the machines. For example, the delay of the starting time of the last operation of some machine may allow the first operation of a different machine to be delayed as well, producing a solution with a lower NPE. This possibility is not taken into account by the proposed energy post-optimization procedure, as it does not allow to delay the last task of each machine.

As an example, we have seen that applying the energy post-optimization procedure to the schedule of Figure 1 produces the schedule of Figure 3 and the NPE is reduced from 26 to 20. But further improvement is possible if we allow to delay operation θ_{12}, as that allows us to also delay operations θ_{23} and θ_{22}, resulting in the schedule represented in Figure 7, which has the same TWT but an even lower NPE of 18 (machine R_2 adds 6 and machine R_3 adds 12).

However, checking for all these possibilities would significantly increase the computational burden

of the heuristic procedure, and so it could not be used to improve every solution. We have decided to propose a more computationally expensive procedure, but only use it to improve the final set of non-dominated solutions returned by the evolutionary algorithm in its last generation. In order to do that, given an input solution S and the problem definition of Section 3, the following relaxed Linear Programming (LP) problem is considered.

$$min \sum_{k=1,..,M} [P_k^{idle} \times (s_{\omega_k} + p_{\omega_k} - s_{\alpha_k} - \sum_{u \in M_k} p_u)]$$

$$s.t.: \quad s_v + p_v \leq s_{SJ_v} \quad v \in \Omega \setminus \{\theta_{1n_1}, .., \theta_{Nn_N}\} \tag{5a}$$

$$s_v + p_v \leq s_{SM_v} \quad v \in M_k \setminus \{\omega_k\}, k = 1, .., M \tag{5b}$$

$$0 \leq s_{\theta_{i1}} \quad i = 1, .., N \tag{5c}$$

$$s_{\theta_{in_i}} + p_{\theta_{in_i}} \leq \max\{C_i, d_i\} \quad i = 1, .., N \tag{5d}$$

The decision variables are the starting times of the operations s_v with $v \in \Omega$. The constraints (5a) represent the linear orderings imposed by the jobs J on the set of operations Ω, which hold for each operation $v \in \Omega$ except when v is the last operation of a job J_i. The constraints (5b) represent the processing orderings on the machines in S. Constraints (5c) impose to the first operation θ_{i1} of each job J_i to start after time 0. Constraints (5d) ensure that the final TWT value is less than or equal to that of the input solution S, by constraining the ending times of the last operations θ_{in_i} of each job J_i.

Clearly, all the imposed temporal constraints are of the form $x - y \leq c$. Therefore, in accordance with [67, 68] the coefficient matrix of the described LP is *totally unimodular* (TU), and hence all the optimal solutions of the LP problem remain discrete values and provide the optimal *NPE* given the processing order of the operations established by the initial solution S. A similar approach is described in [59], where the optimal timing problem for non-regular single objectives in a job-shop is reduced to a minimum cost flow problem.

We are applying this linear programming approach to further improve the NPE of all solutions of the final Pareto front obtained in the last generation of the memetic algorithm.

5. Decomposition-based multi-objective evolutionary algorithm

MOEA/D (multiobjective evolutionary algorithm based on decomposition) was initially proposed in [7]. It is a generic decomposition-based multiobjective optimization framework [69], which is simple and powerful, and has been successfully applied to solve many optimization problems. In this section we describe the approach taken in this paper, which is one of its contributions.

5.1. MOEA/D basics

The main idea of MOEA/D is to decompose a given multi-objective problem into several single-objective subproblems, each of them defined by a scalarizing function using a different weight vector. Each problem is solved using evolutionary operators as crossover or mutation or even local search, as in a single-objective setting, but a neighborhood relation is also defined between subproblems; in this way a solution for a subproblem may help when solving some neighboring subproblems.

To decompose the original multi-objective problem into a number of scalarized single-objective subproblems, several scalarizing functions have been proposed so far in the literature [70]. Probably the most used, for its proven efficiency, is the weighted Tchebycheff (g^{te}) function which is to be minimized:

$$g^{te}(x, \lambda) = \max_{i \in \{1,2\}} \lambda_i \cdot |z_i^* - f_i(x)| \tag{6}$$

where $\lambda = (\lambda_1, \lambda_2)$ is a positive weighting coefficient vector and $z^* = (z_1^*, z_2^*)$ is a reference point, which will be (0,0) in our case. Another commonly used function is the weighted sum, but it is well known that it usually performs worse than the Tchebycheff function. We have also empirically confirmed this on our particular problem.

Let $(\lambda^1, \ldots, \lambda^{NP})$ be a set of NP uniformly distributed weighting coefficient vectors defining NP subproblems that will be optimized. The goal is to approximate the solution x with the best scalarizing function value $g^{te}(x, \lambda^i)$ for each subproblem $i \in \{1, \ldots, NP\}$. To this end, MOEA/D maintains a population $P = (x^1, \ldots, x^{NP})$, in which each individual corresponds to a "good" solution for one of the subproblems.

As an example, if $NP = 6$, then MOEA/D will have a population of 6 solutions, and their weighting coefficient vectors will be $\lambda^1 = (0, 1)$, $\lambda^2 = (0.2, 0.8)$, $\lambda^3 = (0.4, 0.6)$, $\lambda^4 = (0.6, 0.4)$, $\lambda^5 = (0.8, 0.2)$ and $\lambda^6 = (0, 1)$.

Continuing with the example, now suppose we want to calculate the weighted Tchebycheff function of a given solution x for the third subproblem. In order to do it, we have to compute $\max(0.4 \cdot fn_1(x), 0.6 \cdot fn_2(x))$, where $fn_1(x)$ and $fn_2(x)$ are the normalized values of the first and second objective functions, respectively, of solution x. Hence, $fn_1(x) = (f_1(x) - f_1min)/(f_1max - f_1min)$, where $f_1(x)$ is the value of the first objective function of solution x, and f_1min and f_1max are the minimum and maximum values of the first objective function, respectively, found so far in the search. $fn_2(x)$ is defined in a similar way.

Furthermore, in MOEA/D a set of neighbors $B(i)$ is defined for each subproblem $i \in \{1, \ldots, NP\}$, considering the T closest weight vectors, including i. In the standard version of MOEA/D the subproblems are optimized iteratively in order to evolve the population. At a given iteration corresponding to a subproblem i, an offspring solution y is created by applying a crossover operator to a pair of solutions selected at random from $B(i)$ (i.e., from the neighboring subproblems of i), and then a mutation operator to the resulting solution. Then it performs the replacement: for every subproblem $j \in B(i)$, it checks if y improves over j's current solution x^j (of course considering λ^j), and in that case y replaces it. Notice that y might replace the current solution of several subproblems in a single iteration, which will cause diversity problems, as detailed in the following subsection. The algorithm continues iterating until a stopping condition is reached.

5.2. Advanced selection and replacement strategies

As stated in [71], the selection and replacement components in MOEA/D are critical when trying to achieve an intensification-diversification trade-off when evolving the population. This trade-off is a key ingredient in the performance of any evolutionary algorithm. In fact, in [71] it is proven that the standard MOEA/D suffers from a loss in population diversity and premature convergence when solving a bi-objective JSP with uncertainty. Some alternatives have been proposed in the literature to

address this problem and maintain the diversity, as for example MOEA/D-n_r [72], MOEA/D-xy [73] or MOEA/D-RO [71], among others. In this work we have decided to implement the generational approach proposed in MOEA/D-xy [73], because it is one of the few studies that efficiently tackle the combinatorial case, as most MOEA/D papers study continuous multi-objective problems.

The main idea developed in MOEA/D-xy [73] is that since subproblems are processed in an iterative way in conventional MOEA/D, as well as in its variants [72], as soon as a solution gets replaced in a given iteration, it is definitively discarded and gets no chance to produce any offspring solutions, even though it could be potentially beneficial. Accordingly, the authors proposed a generational approach in which all individuals in the population are evolved simultaneously before the replacement takes place. They consequently described four variants denoted MOEA/D-xy with x,y $\in \{s, c\}$. The components of the couple (x,y) correspond respectively to the selection and replacement strategies; the notation s (resp. c) refers to a so-called selfish (resp. collective) variant. The experimental study reported in [73] showed the superiority of MOEA/D-ss over conventional MOEA/D and its variants [7] [72]. Although we finally adopted MOEA/D-ss in our paper, we have also tried the variants MOEA/D-cc, MOEA/D-sc and MOEA/D-cs and the results were usually worse.

MOEA/D-ss introduces the following modifications to the conventional MOEA/D framework. First, for each subproblem i, a new offspring y^i is generated by fixing the first parent to be the current solution x^i and the second one is selected from the remaining subproblems in the T-neighborhood of i with probability δ, or from the whole population with probability $1 - \delta$. The use of the parameter $\delta \in [0, 1]$ in the selection phase was initially proposed in [72] and is typically set to a high value; it can be viewed as a way to increase diversity among parents which might potentially result in increasing the diversity of the generated offsprings.

As for the replacement, it is not performed until we have generated one offspring for every subproblem. Then, an offspring y^i can only replace the *current* solution x^i of the subproblem from where it was originated. Once the replacement is done, then the next generational round is performed and so until the termination criterion is met. This variant is termed selfish because each subproblem privileges its own solution in the selection phase (whereas in the variants with $x = c$ the first parent is not fixed), and it also does not include its neighbors in the replacement phase (in the variants with $y = c$ a solution can also replace solutions from neighboring subproblems).

5.3. Local search

One of the advantages of decomposition-based frameworks is that we do not need to define complicated multi-objective local search methods. Instead, we can use local search methods designed for single-objective optimization. To this end, we are using a standard hill climbing algorithm that works as follows: starting from an initial solution it generates solutions one by one, until we find a neighbor with a better scalarizing function value than the current solution. If we find such neighbor the procedure is repeated switching the current solution for the newly found solution, in other case the procedure ends returning the current solution. As for the neighborhood strategy, we use that previously defined in Section 4.2.1. This local search is applied to every individual of the initial population, and also to every generated offspring.

5.4. Archive of non-dominated solutions

Furthermore, we use an archive for storing all the non-dominated solutions found during the search process. In NSGA-II this is not really necessary, as all non-dominated solutions are stored in the population. In our MOEA/D based algorithm, however, during the single-objective local search it is quite common to evaluate a neighbor which can be great for a different subproblem but not good enough for the current subproblem we are trying to optimize. Therefore, we opt to maintain an external archive of non-dominated solutions and we check the inclusion in the archive of every generated solution in the MOEA/D framework and every neighbor generated during the local searches.

The size of the archive should not be very large at any time point, as every time a new solution is added to it, all solutions dominated by the newly added solution are removed from the archive. In our experimental study, the largest number of non-dominated solutions we found in any run was 163, in the LA31 instance, which has 300 tasks (30 jobs and 10 machines). However, we conjecture that the space of the archive is exponential in the worst case, and so difficult huge instances may have a very large number of non-dominated solutions, and in that case it might be necessary to impose a limit on the size of the archive and use the crowding distance criterion (see Section 4.1.4) in order to select them. In our experimental study, this was not necessary.

5.5. Outline of the proposed algorithm

The representation of solutions is the same as described in Section 4.1.1, whereas the evaluation is also the same: the insertion strategy described in Section 4.1.1 with the energy post-optimization procedure detailed in Section 4.1.2. We again use the JOX crossover operator and swap mutation, both described in Section 4.1.3, although this time we set the JOX operator to produce only one offspring solution instead of two.

Algorithm 3 shows an overview of the described approach. Firstly, a population of $popSize$ individuals (or μ) is randomly created, each one corresponding to a particular subproblem. Local search is then applied to every solution. Then, the algorithm iterates over $numGen$ iterations or generations. In each generation, for each subproblem the following process is applied: one parent will be the current solution of that particular subproblem, and the other parent is selected from its neighborhood (consisting of the T subproblems with the closest weighting coefficient vectors) with probability δ, or from the whole population with probability $1 - \delta$. Then the crossover operator is applied to these parents with probability $crProb$, creating one offspring solution, which is then applied the mutation operator with probability $mutProb$, and finally the single-objective local search to improve it. As soon as we have $popSize$ offspring solutions (one for each subproblem) then the generational replacement is performed: for each subproblem we select the best between its current solution and its corresponding offspring. After we perform the number of generations indicated by the stopping condition, the external archive will contain the final Pareto set. Then, we apply to those solutions the Linear Programming step described in Section 4.3 in order to further improve them, and we return the resulting solutions.

Algorithm 3: The improved MOEA/D-ss (based on that proposed in [73]) hybridized with local search. The archive of non-dominated solutions is updated every time a new solution is evaluated (that includes all neighbors generated during the local search)

input : $\{\lambda^1, \ldots, \lambda^\mu\}$: weight vectors w.r.t. sub-problems;
g: a scalarizing function;
$B(i)$: the neighbors of sub-problem $i \in \{1, \ldots, \mu\}$;
output: The archive of non-dominated solutions (which can then be further improved by the Linear Programming step)

```
begin
    Archive of non-dominated solutions ← ∅;
    Create a random initial population P ← {p^1, ..., p^μ};
    for i ∈ {1, ..., μ} do p^i ← localsearch(p^i) ;
    currentGen ← 0;
    while currentGen < numGen do
        currentGen ← currentGen + 1;
        for i ∈ {1, ..., μ} do
            if rand(0,1) < δ then B_i ← B(i) ;
            else B_i ← P ;
            l ← rand(B_i);
            while l = i do l ← rand(B_i) ;
            o^i ← p^i;
            if rand(0,1) < crProb then o^i ← crossover(p^i, p^l) ;
            if rand(0,1) < mutProb then o^i ← mutation(o^i) ;
            o^i ← localsearch(o^i);
        end
        for i ∈ {1, ..., μ} do
            if g(o^i, λ^i) < g(p^i, λ^i) then p^i ← o^i ;
        end
    end
end
```

6. A Constraint Programming approach

Constraint Programming (CP) is a declarative programming paradigm [74] which is appropriate for tackling constraint optimization problems. We have to first create a *model* of the problem, defined as a set of *decision variables* that range on a discrete domain of values, and a set of *constraints* that imposes limits on the possible combinations of variable-value assignments. Afterwards, the solver interleaves two steps: constraint propagation (where inconsistent values are removed from the domains of the variables) and search.

The described paradigm is particularly suited for efficiently solving scheduling problems where the decision variables correspond to the problem operations. Each operation variable u is characterized

by at least two features: s_u represents its start time, and p_u represents its processing time.

Several authors have developed a number of different *global constraints* for scheduling problems. The most important are probably the `unary-resource` constraint [75], which is useful for modelling simple machines, the `cumulative` resource constraint [76] that models cumulative resources, as for example a pool of workers, and the `reservoir` [77] for modelling consumable resources, as for example a fuel tank. In particular, the constraint `unary-resource(`A`)` holds if and only if all operations in the set A do not overlap at any time point. A number of propagation algorithms are embedded in the `unary-resource` constraint in order to remove provably inconsistent assignments of operation start-time variables.

Our starting CP model is taken from [9], where we introduce an additional set of decision variables that represent the switch *ON/OFF* events in the set of machines M_k. The purpose of these decision variables is to take into account the non-regularity of the *NPE* objective function. In addition, we consider a lexicographic optimization method in place of a single-objective optimization, which consider both the given objective functions, that for reader's convenience we report in the following:

$$NPE = \sum_{k=1,\ldots,M} [P_k^{idle}(s_{\omega_k} + p_{\omega_k} - s_{\alpha_k} - \sum_{u \in M_k} p_u)] \tag{7a}$$

$$TWT = \sum_{i=1,\ldots,N} w_i \max\{C_i - d_i, 0\} \tag{7b}$$

In this work, we consider two different CP models, targeted at lexicographically minimizing both pairs of objectives (NPE, TWT) and its *dual* (TWT, NPE). The reader should note that in the following we consider the OPL language [78] as reference for the description of the CP models.

$$\begin{aligned}
&\min && lex(NPE, TWT) && \text{(8a)}\\
&\text{s.t. :} && s_v + p_v \leq s_{SJ_v} \quad v \in \Omega \setminus \{\theta_{1n_1}, .., \theta_{Nn_N}\} && \text{(8b)}\\
& && \texttt{span}(OnOff_k, M_k) \quad k = 1, .., M && \text{(8c)}\\
& && \texttt{noOverlap}(M_k) \quad k = 1, .., M && \text{(8d)}\\
& && s_{\omega_i} + p_{\omega_i} \leq \lceil TWT_\epsilon / w_i \rceil + d_i \quad i = 1, .., N && \text{(8e)}
\end{aligned}$$

In the first model the decision variables are the starting times of operations $s_v \in \Omega$, extended with the starting times s_{OnOff_k} of operations $OnOff_k$, which represents the first instant when machine k is turned on, so that the machine will remain in this state for the entire duration p_{OnOff_k}. We assume that every decision variable s_v (s_{OnOff_k}) may respectively take values in the interval $[0, H - p_v]$ ($[0, H - p_{OnOff_k}]$), where H is the horizon value, representing the maximum value that each decision variable can assume. The (8a) statement represents the lexicographic minimization of the objective (NPE, TWT) with the energy *NPE* as primary key. The linear orderings imposed on the set of operations Ω by the set of jobs J are represented by constraints (8b), which hold for each operation $v \in \Omega$ except when v is the last operation of a job J_i. Constraints (8c) use the OPL function `span()`, which imposes that the set of operations M_k (the set of operations requiring machine k) be contained in the *spanning* operations $OnOff_k$, $k = 1, 2, \ldots, M$. Specifically, for each operation $u \in M_k$ the constraints $s_{OnOff_k} \leq s_u$ and $s_u + p_u \leq s_{OnOff_k} + p_{OnOff_k}$ must hold. Constraints (8d) represent the non-overlapping

constraints imposed by machines M_k by means of the global constraints `unary-resource()` introduced above, implemented through the OPL function `noOverlap()`. Finally, given a total bound TWT_ϵ on the secondary key *TWT* ($TWT \leq TWT_\epsilon$), constraints (8e) impose an upper bound on the end-times of each job J_i such that the contribution of each single job J_i to the objective TWT cannot be greater than the given global value TWT_ϵ. The use of this last set of constraints is explained at the end of this section, where we describe the ϵ-constraint method [8] for calculating an approximation of the Pareto set.

$$\text{min} \quad lex(TWT, NPE) \tag{9a}$$

$$\text{s.t. :} \quad s_v + p_v \leq s_{SJ_v} \quad v \in \Omega \setminus \{\theta_{1n_1}, .., \theta_{Nn_N}\} \tag{9b}$$

$$\texttt{span}(OnOff_k, M_k) \quad k = 1, 2, .., M \tag{9c}$$

$$\texttt{noOverlap}(M_k) \quad k = 1, .., M \tag{9d}$$

$$e_{OnOff_k} - s_{OnOff_k} \leq \lceil NPE_\epsilon / P_k^{idle} \rceil + \sum_{u \in M_k} p_u \quad k = 1, .., M \tag{9e}$$

The second CP model is similar to the previous one, with two differences: (i) it lexicographically minimizes the pair (TWT, NPE), see (9a)); (ii) it has a dual set of constraints (9e) on the duration of the $OnOff_k$ operations (e_{OnOff_k} is the end-time of the $OnOff_k$ operation), which indirectly bound the secondary objective *NPE*. In particular, given a total bound NPE_ϵ on the secondary key *NPE* ($NPE \leq NPE_\epsilon$), the (9e) constraints impose an upper bound on the duration of the $OnOff_k$ operations, such that the contribution of each single machine M_k to the objective NPE cannot be greater than the given global value NPE_ϵ. As for the constraint set (8e), its use will be described at the end of this section.

It is important to remark that both the above CP models remain perfectly feasible and solvable without the s_{OnOff_k} decision variables. Although such variables are not strictly necessary, we have observed in a preliminary study that their addition to the models is beneficial and helps the solver to minimize the NPE objective. In fact, due to the non-regularity of the NPE objective (i.e., NPE non-monotonically decreases as the start-times of the activity increase), a delay decision on the start-time s_{OnOff_k} influences the whole set of operations M_k and could have the effect of "grouping" them, with a consequent reduction of the NPE objective.

As previously introduced, we use the ϵ-constraint method [8] to generate the Pareto front. It is a well-known multi-objective optimization method that proceeds by choosing only one objective function as the only objective, while properly constraining all the remaining objectives during the optimization process. Different solutions of the Pareto front can be obtained by systematically varying the constraint bounds.

In Algorithm 4 we present the ϵ-constraint method for the particular case of a bi-criterion objective function $\mathbf{f} = (f^{(1)}, f^{(2)})$. The algorithm takes as inputs: (i) the objective $\mathbf{f}$, (ii) the bounds $f^{(2)}_{min}$ and $f^{(2)}_{max}$ on the second component of the objective, and (iii) the decrement value δ. As previously mentioned, the method iteratively leverages a procedure provided in input to solve constrained optimization problems, which in our case is the CP() procedure implementing the constraint programming models previously described. Note that in this work we consider an ϵ-constraint method that is slightly different w.r.t. the "canonical" one, as the given CP model considers a lexicographic min-

Algorithm 4: Bi-criterion ϵ-constraint method

input : the objective **f**, the bounds $f^{(2)}_{min}$ and $f^{(2)}_{max}$, and the decrement value δ
output: the Pareto set *P*
begin
  $P \leftarrow \emptyset$;
  $\epsilon \leftarrow f^{(2)}_{max}$;
  while $\epsilon \geq f^{(2)}_{min}$ **do**
    $S \leftarrow$ CP(**f**, ϵ);
    if $(S \neq nil) \wedge (\nexists S' \in P : S' \prec S)$ **then**
      $P \leftarrow (P \cup \{S\}) \setminus \{S' \in P : S \prec S'\}$;
    end
    $\epsilon \leftarrow \epsilon - \delta$;
  end
end

imization instead of single-objective minimization, with $f^{(1)}$ as primary key and $f^{(2)}$ as secondary one. In addition, according to the constraints (8e) ((9e)), we do not impose a direct bound on the *TWT* (*NPE*) objective; rather, we impose a bound on the end-times of the jobs (the duration of the $OnOff_k$ operations) through the constraints (8e) ((9e)), which indirectly affects the corresponding objective function *TWT* (*NPE*).

Algorithm 4 proceeds as follows: after initializing the constraint bound ϵ to the $f^{(2)}_{max}$ value, a new solution S is computed by calling CP() at each step of the *while* solving cycle. If S is not dominated by any of the existing solutions in the current Pareto front approximation P, then S is inserted in P, and all the solutions possibly dominated by S are removed from P. The rationale behind this method is to iteratively tighten the constraint bound by a pre-defined constant δ at each step of the solving cycle. Although we use a slightly different version of the ϵ-constraint method, in a preliminary study we have observed that the substitution of the constraint $TWT \leq TWT_\epsilon$ ($NPE \leq NPE_\epsilon$) with the combined action of the constraints (8e) ((9e)) and the lexicographic minimization of the objectives (NPE, TWT) $((TWT, NPE))$ yields results that bend in favor of the latter solution. This fact is probably due to the lack of a constraint mechanism that propagates the bounds imposed on the two objectives on the start-times of the operations (the problem decision variables).

7. Experimental results

We have conducted an experimental study to evaluate our proposed algorithms. We start describing the benchmark instances. Then, we briefly report about the performed parameter tuning. Finally we analyze the contribution of each of the components of our algorithms to its overall performance, and also compare them with the state-of-the-art method of the literature.

7.1. Benchmarks considered

The authors of the [45] paper that inspired our work about this problem, only consider the well-known FT10 instance in their experiments. In our work we also consider this instance, but we add eight more JSP instances from the literature of different sizes in order to perform a more exhaustive experimental analysis. In particular we take one well-known instance from a set of possible sizes (*number-of-jobs* $\times$ *number-of-machines*): LA01 (size 10×5), LA06 (size 15×5), FT10 (size 10×10), FT20 (size 20×5), LA21 (size 15×10), LA27 (size 20×10), LA31 (size 30×10), LA40 (size 15×15) and ABZ7 (size 20×15). All these instances are available in the OR-library ([79, 80]) and most of them are well-known for being very difficult to solve in the classical JSP with makespan minimization [81].

The due date d_i of each job J_i is assigned using the expression $d_i = k \times \sum_{j=1}^{n_i} p_{ij}$, where n_i is the number of activities related to the i-th job, and k is a parameter that controls the tightness of the due dates. For the FT10 instance we take the values $k = 1.5$, $k = 1.6$, and $k = 1.7$, as these are the values used in [45] (we omit the $k = 1.8$ case so as to have three k values for each instance, considering also that the $k = 1.8$ case has shown to be not very challenging). On all remaining instances we consider the values $k = 1.3$, $k = 1.5$ and $k = 1.6$, as these are common in the TWT literature (see for example [19, 16, 18]). Considering these values for the k parameter we have a total of 27 different instances.

This technique for creating due dates was originally proposed in [82], but it was used afterwards in most papers about TWT minimization in the JSP (see for example [83, 15, 16, 19, 18]). The selected values for the k parameter are known to produce a good range of instances, where $k = 1.3$ produces tight and very difficult to meet due dates, whereas $k = 1.7$ produces loose and quite easy to meet due dates. This is especially true in "square" instances (i.e., instances with the same number of jobs and machines), although in [16, 19, 18] these values of k are also used for "rectangular" instances.

The weights of the jobs are assigned in the same way as in other papers in TWT minimization: the first 20% of jobs are assigned weight $w = 4$ (high priority), the next 60% of jobs are assigned weight $w = 2$ (average priority) and the last 20% of jobs are assigned weight $w = 1$ (low priority). The exception is instance FT10, where we take the weights used in [45], which are the following: $w_1 = 1$, $w_2 = 2$, $w_3 = 3$, $w_4 = 1$, $w_5 = 3$, $w_6 = 2$, $w_7 = 3$, $w_8 = 2$, $w_9 = 1$ and $w_{10} = 1$.

Finally, the idle power consumption of the machines is based on that used in [45]: $P_1^{idle} = 2400$, $P_2^{idle} = 3360$, $P_3^{idle} = 2000$, $P_4^{idle} = 1770$, $P_5^{idle} = 2200$, $P_6^{idle} = 7500$, $P_7^{idle} = 2000$, $P_8^{idle} = 1770$, $P_9^{idle} = 2200$, $P_{10}^{idle} = 7500$, $P_{11}^{idle} = 2400$, $P_{12}^{idle} = 3360$, $P_{13}^{idle} = 2000$, $P_{14}^{idle} = 1770$ and $P_{15}^{idle} = 2200$. In instances with 5 or 10 machines we simply cut the extra machines.

7.2. Parameter tuning

The NSGA-II and MOEA/D based hybrid metaheuristics were fully implemented by the authors of this paper in C++ using a single thread. Besides, the Linear Programming (LP) and Constraint Programming (CP) approaches are implemented using IBM CPLEX Optimization Studio 12.7.1. Target machine is an Intel Core i5-2450M CPU at 2.5 GHz with 4 GB of RAM, running on Windows 10 Pro. Firstly, we have performed a parametric analysis in order to fix some of the parameters of our approaches, using the set of instances described in Section 7.1. For the NSGA-II based approach, Table 2 reports a summary of the tested values, in which bold numbers indicate the configuration that obtained the best average results. The stopping condition $numGen$ is modified accordingly in these tests so

Table 2. Values tested in the parameter tuning for the NSGA-II based approach. Bold values indicate the best configuration found.

Parameter	Values tested
$popSize$	500, **1000**, 2000
$tSize$	**2**, 4, 8, 16
$crProb$	0.6, 0.8, **1.0**
$mutProb$	0, 0.1, **0.2**, 0.3

that all possible configurations take similar computational times. Using the proposed configuration and the setting $numGen = 2000$, we have confirmed that the convergence pattern is appropriate and the running time is reasonable. Precisely, the computational time depends on the instance size, and in average is as follows: 23 minutes for ABZ7, 4 minutes for FT10, 6 minutes for FT20, 1.5 minutes for LA01, 3 minutes for LA06, 7 minutes for LA21, 12.5 minutes for LA27, 34 minutes for LA31, and 11 minutes for LA40.

The time taken by each component of the hybrid metaheuristic is in average 73% for the local search, 27% for the genetic algorithm and less than 1% for the linear programming final step. It is worth to remark that, even if the time taken by the LP step is less than 1% of the total time, it would be too computationally costly to apply the LP procedure in all generations of the algorithm. We have tried to do it in some preliminary experiments and we have confirmed that the huge increase in computational time does not compensate for the very slight improvement in solution quality.

For the MOEA/D based approach we decided to use a base configuration as similar as possible to the NSGA-II based approach. Therefore, we also choose $popSize = 1000$, $numGen = 2000$, $crProb = 1.0$ and $mutProb = 0.2$. Aside from those parameters, we also have to fix the δ parameter (that indicates the probability of choosing the second parent from the neighborhood) and the T parameter (that indicates the size of the neighborhood for each subproblem, including the subproblem). After some preliminary experiments we decided to set $\delta = 0.95$ (we have also tested $\delta = 0.9$ and $\delta = 1.0$) and $T = 21$ (we have also tested $T = 11$ and $T = 41$). Using this configuration, the running times are similar to those used by the other metaheuristic approach.

7.3. Analysis of our algorithms and comparison with the state of the art

To compare the performance of the algorithms we choose the hypervolume indicator (HV) [84], as it is one of the most popular indicators in multi-objective optimization. If solutions are considered as points in objective space, hypervolume is the n-dimensional volume of the set relative to some reference point. Figure 8 shows an example of Pareto set containing three solutions relatively to a bi-objective minimization problem; the hypervolume value is the area of the shaded portion, provided the reference point delimiting the area bounds. It is expected that a set of solutions with larger hypervolume presents better trade-offs than sets with lower hypervolume.

Tables 3, 4, and 5 report the hypervolume values for all the 27 previously mentioned instances. The first column of each table shows the problem instance, while the second column shows the general

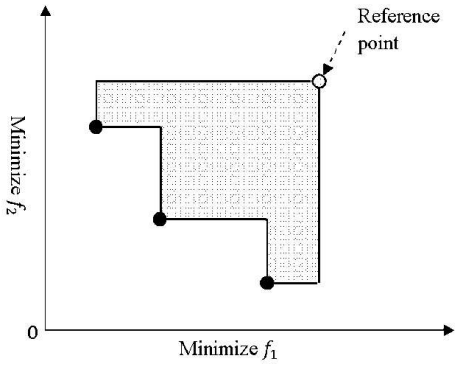


Figure 8. The hypervolume measure (shaded area) for a bi-objective minimization problem.

solving approach. In particular, *Genetic* is the algorithm described in Section 4.1, *MOEA/D* is the algorithm described in Section 5, *CP* is the algorithm described in Section 6, while *LIU* is the NSGA-II-based algorithm proposed in [45], which only appears in Table 3 because only the FT10 instance is analyzed in that paper. We note that, in order to run *LIU* in the remaining instances we could have tried to obtain the source code (it is not publicly available) or reimplement the method from scratch. However, we did not deem it necessary, due to the fact that its results in the FT10 instance with every value for the k parameter are considerably far away from those obtained by any of our methods. The third column shows the particular version of every approach (i.e., $\langle emptybox \rangle$ = plain version, $+LS$ = Local Search (see Section 4.2), $+PO$ = Post-Optimization (see Section 4.1.2), $+LS + PO$ = Local Search + Post-Optimization). For the $FT10$ instance only, we have added an extra row related to the *Genetic* case called $noDR$ (i.e., no duplicate removals), that reports the results obtained with the plain genetic version without removing duplicate solutions in the population. Comparing the results obtained without the duplication removal method with those obtained by using it (i.e., the row called $noDR$ with the previous row), it can be seen that removing duplicates has beneficial effects, as demonstrated by the larger hypervolume values.

The remaining columns divide the tables in three sections, one for each value of the k parameter. Lastly, for each k value there are two columns (labelled *no* LP and LP) that report the results of the solving approach without and with the final linear programming step, respectively.

For *Genetic + LS + PO* and *MOEA/D + LS + PO* we set $numGen = 2000$. For *Genetic + LS* and *MOEA/D + LS* we set $numGen = 2500$ for all instances. Finally, for *Genetic*, *Genetic+PO*, *Genetic+PO+LP*, *MOEA/D*, *MOEAD/D+PO*, *MOEA/D+PO+LP* the parameter $numGen$ varies between 4000 and 32000 depending on the instance size and on the particular method.

The proposal of [45] (labelled LIU) is a standard NSGA-II algorithm using OOX crossover operator and swap mutation. The crossover probability is set at 1.0 and the mutation probability at 0.6. The population size varies between 800 and 1000 depending on the instance, and the total number of generations vary between 25000 and 40000. As the authors do not report the computational time, we have implemented a version of their method and concluded that their running time is considerably larger than that of our approach: about 15 minutes per run.

Table 3. Hypervolumes computed for the $LA01$, $LA06$, $FT10$ instances.

Instance	Algorithm	Version	$k = 1.3$		$k = 1.5$		$k = 1.6$	
			no LP	LP	no LP	LP	no LP	LP
LA01	Genetic		0.42	0.45	0.54	0.58	0.60	0.65
		+LS	0.43	0.46	0.55	0.60	0.60	0.66
		+PO	0.46	0.46	**0.61**	**0.61**	0.66	0.66
		+LS+PO	**0.47**	**0.47**	**0.61**	**0.61**	**0.67**	**0.67**
	MOEA/D		0.42	0.45	0.53	0.58	0.60	0.65
		+LS	0.43	0.46	0.55	0.60	0.60	0.65
		+PO	0.46	0.46	0.59	0.59	0.66	0.66
		+LS+PO	**0.47**	**0.47**	**0.61**	**0.61**	0.66	0.66
	CP		0.43	0.43	0.55	0.55	0.61	0.61
LA06	Genetic		0.41	0.43	0.52	0.53	0.57	0.59
		+LS	0.43	0.44	0.53	0.55	0.57	0.59
		+PO	0.45	0.45	0.56	0.56	0.61	0.61
		+LS+PO	**0.46**	**0.46**	**0.57**	**0.57**	**0.62**	**0.62**
	MOEA/D		0.40	0.41	0.50	0.51	0.56	0.58
		+LS	0.41	0.43	0.53	0.55	0.57	0.59
		+PO	0.44	0.44	0.55	0.55	**0.62**	**0.62**
		+LS+PO	**0.46**	**0.46**	**0.57**	**0.57**	**0.62**	**0.62**
	CP		0.44	0.44	0.56	0.56	0.61	0.61
FT10			$k = 1.5$		$k = 1.6$		$k = 1.7$	
	Genetic		0.63	0.63	0.68	0.69	0.72	0.75
		no DR	0.59	0.60	0.64	0.67	0.72	0.73
		+LS	0.62	0.63	0.70	0.70	0.76	0.77
		+PO	0.62	0.64	0.72	0.73	0.75	0.76
		+LS+PO	0.65	**0.67**	0.73	**0.74**	0.76	0.76
	MOEA/D		0.59	0.61	0.70	0.71	0.75	0.76
		+LS	0.64	0.64	0.71	0.72	0.76	0.76
		+PO	0.64	0.65	0.71	0.72	**0.78**	**0.78**
		+LS+PO	0.65	0.66	0.73	**0.74**	**0.78**	**0.78**
	CP		0.59	0.59	0.69	0.70	0.70	0.71
	LIU		0.38		0.45		0.52	

About the CP approach, we generate an approximation of the Pareto set by using the *ϵ-constraint* algorithm (see Algorithm 4) by running the procedure twice: in the first run we lexicographically optimize the *(NPE,TWT)* pair, while in the second run we lexicographically optimize the *(TWT,NPE)* pair. Afterwards, we merge the two approximations of the Pareto sets into a single one, and calculate the hypervolume value of the Pareto approximation thus obtained. Each run of the *ϵ-constraint* algorithm is composed of 10 solving cycles. In each solving cycle, we leverage the random nature of the CP Optimizer algorithm (i.e., a Large Neighborhood Search strategy) running the CP solver 10 times, thus generally obtaining 10 different solutions. Each of the previous CP runs is allotted a maximum CPU time of 50 seconds; thus, each of the two runs of the *ϵ-constraint* algorithm is characterized by a total CPU time bound of 5000 seconds. About the intervals $[f^{(2)}_{min}, f^{(2)}_{max}]$ on the objective functions, we proceeded as follows. Relatively to the set of solved instances, when we optimize the *NPE* primary objective (*TWT* primary objective) in the CP model, the interval of *NPE* values $[NPE_{lb}, NPE_{ub}]$

Table 4. Hypervolumes computed for the $FT20$, $LA21$, $LA27$ instances.

Instance	Algorithm	Version	$k = 1.3$		$k = 1.5$		$k = 1.6$	
			no LP	LP	no LP	LP	no LP	LP
FT20	Genetic		0.43	0.43	0.52	0.52	0.54	0.54
		+LS	**0.45**	**0.45**	**0.54**	**0.54**	**0.57**	**0.57**
		+PO	0.43	0.43	0.51	0.51	0.54	0.54
		+LS+PO	**0.45**	**0.45**	**0.54**	**0.54**	**0.57**	**0.57**
	MOEA/D		0.41	0.41	0.50	0.50	0.54	0.54
		+LS	0.44	0.44	0.52	0.52	0.56	0.56
		+PO	0.41	0.41	0.48	0.48	0.52	0.52
		+LS+PO	0.44	0.44	0.53	0.53	0.56	0.56
	CP		0.41	0.41	0.49	0.49	0.53	0.53
LA21	Genetic		0.38	0.39	0.51	0.52	0.61	0.62
		+LS	0.38	0.39	0.55	0.56	0.62	0.63
		+PO	0.40	0.40	0.56	0.56	0.65	0.65
		+LS+PO	0.40	0.40	**0.60**	**0.60**	0.65	0.65
	MOEA/D		0.37	0.37	0.51	0.52	0.58	0.60
		+LS	0.39	0.40	0.56	0.57	0.64	0.65
		+PO	0.39	0.39	0.57	0.57	0.66	0.66
		+LS+PO	**0.41**	**0.41**	0.59	0.59	**0.67**	**0.67**
	CP		0.38	0.38	0.56	0.56	0.64	0.64
LA27	Genetic		0.32	0.32	0.47	0.50	0.54	0.57
		+LS	0.37	**0.38**	0.53	0.55	0.60	0.62
		+PO	0.33	0.33	0.51	0.51	0.59	0.59
		+LS+PO	0.36	0.36	0.55	0.55	**0.65**	**0.65**
	MOEA/D		0.32	0.33	0.47	0.49	0.53	0.56
		+LS	0.36	0.37	0.54	**0.56**	0.61	0.62
		+PO	0.34	0.34	0.55	0.55	0.59	0.59
		+LS+PO	0.37	0.37	**0.56**	**0.56**	0.63	0.63
	CP		0.34	0.34	0.50	0.51	0.60	0.60

(*TWT* values $[TWT_{lb}, TWT_{ub}]$) for each value of k is given in Table 6. Note that the second column of Table 6 reports the H_0 values that are imposed as fixed horizon constraints for all of our instances; such values are computed increasing each instance's optimal known makespan by 15%.

As a last step, we apply the Linear Programming (LP) algorithm described in Section 4.3 to the set of solutions obtained with the CP algorithm. It is worth noting that we tested additional ways to generate the Pareto set within the same total CPU time: a run with a single objective and/or a single run at each solving step of the ϵ-*constraint* algorithm.

All tables show that the hybridization with local search clearly improves the performance over the non-hybrid approaches: in general, *LS+PO* approaches are better than their LS, PO, and $Plain$ counterparts. In the FT10 case, we can also observe that even our plain $Genetic$ and $MOEA/D$ algorithms (that is, without local search, PO and LP) is much better than LIU. Note that both our genetic algorithm and the genetic algorithm used in [45] are NSGA-based. The main reason why our genetic algorithm is better is the procedure for eliminating repeated individuals to create each generation, which significantly improves the diversity of the population. Our crossover operator (JOX) also represents a slight improvement with respect to the OOX used in LIU. Also, note that

Table 5. Hypervolumes computed for the $LA31$, $LA40$, $ABZ7$ instances.

Instance	Algorithm	Version	$k = 1.3$		$k = 1.5$		$k = 1.6$	
			no LP	LP	no LP	LP	no LP	LP
LA31	Genetic		0.28	0.29	0.36	0.38	0.4	0.41
		+LS	0.33	0.33	0.42	0.42	0.46	0.47
		+PO	0.32	0.32	0.40	0.40	0.46	0.46
		+LS+PO	**0.33**	**0.33**	**0.43**	**0.43**	**0.48**	**0.48**
	MOEA/D		0.27	0.28	0.33	0.34	0.39	0.41
		+LS	0.31	0.32	0.38	0.39	0.42	0.44
		+PO	0.31	0.31	0.39	0.39	0.45	0.45
		+LS+PO	**0.33**	**0.33**	0.42	0.42	0.46	0.46
	CP		0.25	0.27	0.34	0.35	0.36	0.39
LA40	Genetic		0.11	0.14	0.22	0.27	0.24	0.30
		+LS	0.15	0.19	0.25	0.30	0.27	0.31
		+PO	0.15	0.15	0.35	0.35	0.35	0.35
		+LS+PO	**0.21**	**0.21**	**0.36**	**0.36**	**0.36**	**0.36**
	MOEA/D		0.11	0.14	0.21	0.28	0.25	0.30
		+LS	0.15	0.19	0.26	0.31	0.26	0.31
		+PO	0.17	0.17	0.34	0.34	**0.36**	**0.36**
		+LS+PO	**0.21**	**0.21**	0.35	0.35	**0.36**	**0.36**
	CP		0.13	0.13	0.27	0.28	0.31	0.31
ABZ7	Genetic		0.23	0.25	0.37	0.40	0.43	0.47
		+LS	0.28	**0.30**	0.43	0.45	0.49	0.52
		+PO	0.24	0.24	0.46	0.46	0.50	0.50
		+LS+PO	0.27	0.27	0.46	0.46	0.51	0.51
	MOEA/D		0.23	0.24	0.39	0.42	0.44	0.47
		+LS	0.26	0.28	0.42	0.45	0.47	0.50
		+PO	0.27	0.27	0.45	0.45	0.49	0.49
		+LS+PO	**0.30**	**0.30**	**0.47**	**0.47**	**0.53**	**0.53**
	CP		0.22	0.22	0.43	0.43	0.50	0.51

the CP-based algorithm clearly outperforms LIU, whereas the hybrid evolutionary algorithm further improves over the CP performance. Hence, even if the *ϵ-constraint* algorithm is a basic strategy to generate a Pareto set, the set of results generated with a state-of-the-art CP solver is clearly a further confirmation of the effectiveness of the proposed evolutionary approach.

In general, all tables show that the evolutionary approaches outperform the CP-based approach, even though the CP results always remain rather competitive. The tables also show that there exists an advantage in the exploitation of the final LP step (see the LP columns), with the exception of all the approaches that use the heuristic-based energy post-optimization procedure (PO) described in Section 4.1.2, where the LP advantage is negligible. This is not surprising, given that the PO procedure pursues the same goal as the LP procedure, even though in different phases of the solving process.

Despite the fact that in general the CP method obtains reasonable results in all instances (the CP hypervolumes in Tables 3, 4, and 5 are in general not exceedingly far to the best), it should be observed that the CP performance tends to decrease as the instance size increases (the distance between CP-related hypervolumes and the best hypervolumes progressively increases as the problem's size grows). This property is clearly shown in Figure 9, which compares the performances of the Genetic+LS+PO,

Table 6. CP model: imposed horizon constraint H_0, the interval of both the *NPE* values $[NPE_{lb}, NPE_{ub}]$ and *TWT* values $[TWT_{lb}, TWT_{ub}]$) for each value of k.

Instance	Horizon H_0	NPE_{lb}	NPE_{ub}	TWT_{lb}	TWT_{ub}
ABZ7	755	80	130	3000	8000
FT10	1070	0	200	0	5000
FT20	1340	0	30	0	20000
LA01	766	0	50	2000	5000
LA06	1023	0	80	4000	9000
LA21	1203	40	100	4000	8000
LA27	1420	30	130	12000	18000
LA31	2052	30	100	30000	55000
LA40	1405	180	240	2000	5000

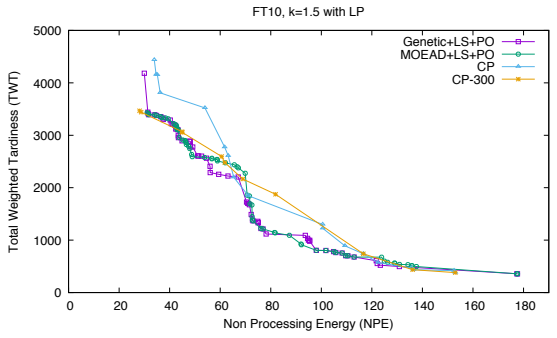
the MOEA/D+LS+PO and the CP algorithms for the instances of increasing size FT10, FT20, LA27, LA40, LA31, and ABZ7, showing that CP obtains the worst performance in the bigger instances ABZ7, LA31, and LA40. In order to deepen the analysis of the CP performance, we decided to perform some tests by providing the CP solver with a greater CPU time. Figure 9 also reports the plots related to the CP results obtained by extending each run from 50 to 300 seconds (*CP-300* plots). The test reveals that giving more time to the CP solver significantly pays off in terms of solution quality, showing that our CP-based ϵ-constraint method had not yet reached a "minimization plateau" after 50 seconds.

In terms of hypervolume improvement, in order to fully appreciate the performance difference, we provide the exact figures in the following. The values relative to the CP approach with CPU time equal to 50 seconds per run are already reported in Tables 3, 4, and 5. Passing from 50 to 300 seconds per run, for the FT10 instance we have a $0.59 \rightarrow 0.60$ improvement; for the FT20 instance we have a $0.41 \rightarrow 0.43$ improvement; for the LA27 instance we have a $0.34 \rightarrow 0.40$ improvement; for the LA31 instance we have a $0.27 \rightarrow 0.33$ improvement; for the LA40 instance we have a $0.13 \rightarrow 0.17$ improvement; finally, for the ABZ7 instance we have a $0.22 \rightarrow 0.27$ improvement.

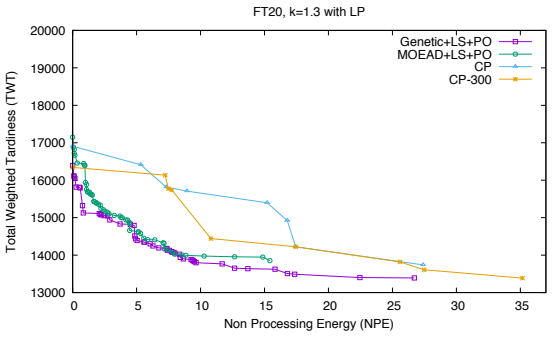
Finally, Figure 10 shows two solutions, respectively *before* and *after* the application of the post-optimization algorithm presented in Section 4.1.2, in the $k = 1.5$ case (the behavior is similar with other values of k). The TWT is the same, while the *NPE* value is significantly improved by introducing delays on the start times of some operations (the most evident delays are those related to machines R_5, R_6, and R_{10}). In general, such improvement is caused by the exploitation of the solution's temporal flexibility which allows to reduce the idle time of the machines that contribute the most to the NPE reduction, according to their respective P^{idle} values.

8. Conclusions and Future Work

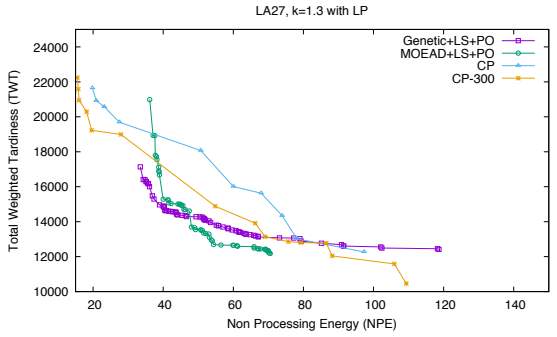
In this paper we have tackled the difficult problem of minimizing at the same time the TWT and the energy consumption in the JSP. We have described three methods: a dominance-based evolutionary metaheuristic based on the NSGA-II framework which is hybridized with a multi-objective local



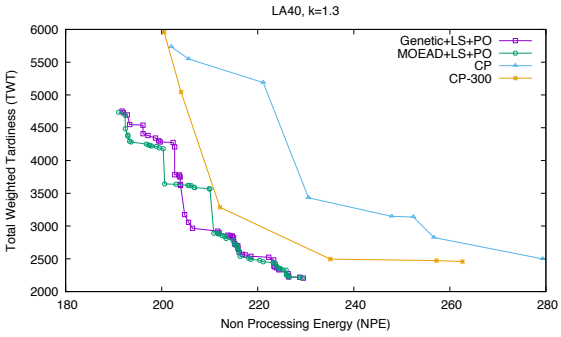
(a) FT10 instance (10×10)



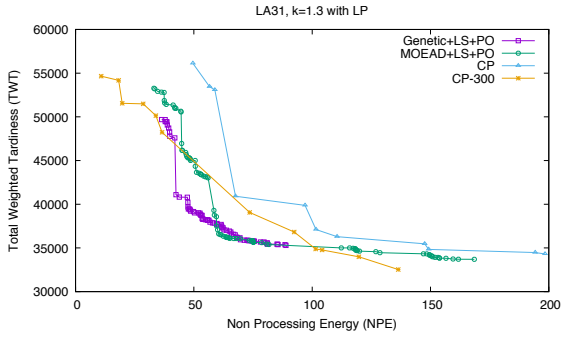
(b) FT20 instance (20×5)



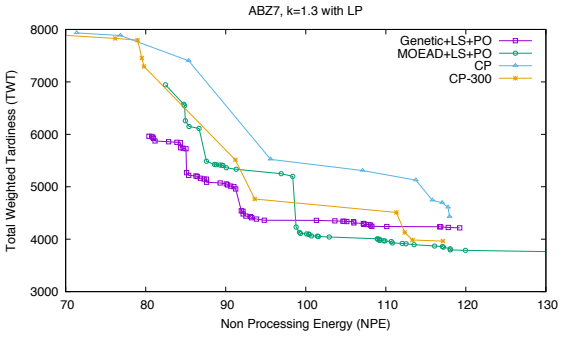
(c) LA27 instance (20×10)



(d) LA40 instance (15×15)



(e) LA31 instance (30×10)

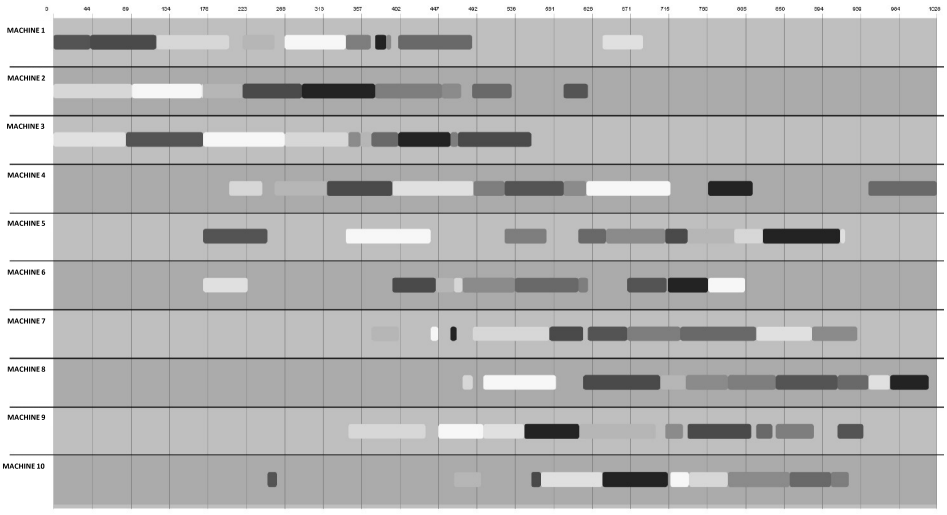


(f) ABZ7 instance (20×15)

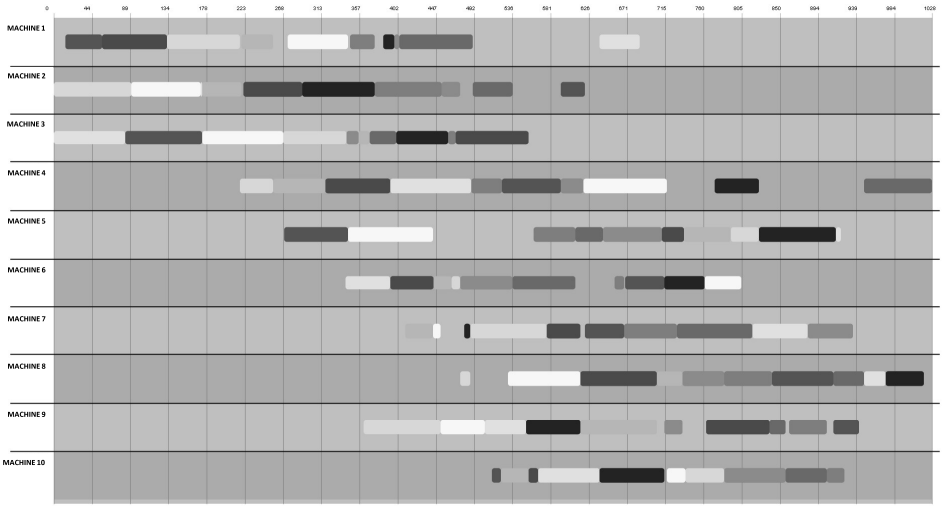
Figure 9. Scalability of the CP model: comparison with the best evolutionary strategies

search method, a decomposition-based evolutionary metaheuristic based on the MOEA/D framework which is hybridized with a single-objective local search method, and a constraint programming approach that uses the ϵ-constraint method. The approach based on the MOEA/D framework is a contribution of this paper, whereas the other two methods are taken from [9].

To minimize the energy-related objective function we propose two methods: a low-polynomial, fast heuristic procedure which is used when evaluating every solution produced by the two metaheuris-



(a) Before post-optimization ($TWT = 3347$, $NPE = 91.98$ KWh)



(b) After post-optimization ($TWT = 3347$, $NPE = 31.35$ KWh)

Figure 10. Improvement of the NPE value as a consequence of the application of the post-optimization procedure (Algorithm 1).

tic approaches, and also an optimal linear programming method which is more costly and thus only applied to the final set of non-dominated solutions returned by our approaches, in order to further improve them.

The results of the reported experimental study prove the efficiency of the proposed energy optimization procedures. We also show the superiority of the metaheuristic approaches over the constraint programming method and also over the state-of-the-art algorithm in the literature, which is the NSGA-II based approach proposed in [45]. The remarkable performance of the evolutionary algorithms is,

in our opinion, due to an adequate balance between the intensification provided by the local search and the diversification provided by the NSGA-II and MOEA/D frameworks. The design of fast local search methods is critical as it allowed us to apply it to every generated solution in a reasonable computational time. Also, the proposed energy optimization procedures were able to efficiently minimize a complicated non-regular objective function such as the energy consumption.

However, a careful observation of Figure 9 suggests that the CP-based ϵ-constraint method we apply in this work can be considered a complementary approach w.r.t. the evolutionary strategies, as it appears to be rather efficient in finding good solutions in zones of the (NPE, TWT) plane that the evolutionary approaches do not easily reach, i.e., at the extremes of the graphs, while the evolutionary strategies are particularly effective on the central zone of the Pareto set. We believe that the Pareto set obtained by merging all the solutions obtained by means of all methods is a rather challenging set to further improve.

As already stated in Section 3, in our approaches we have considered a quite simple energy model. Therefore, the presented methods are only efficient in practical applications where turning off the machines is not an appropriate method for saving energy. This can happen, for example, when turning the machines off and on again consumes more energy than keeping them in idle state. Hence, the most interesting line for future work is to consider more realistic and complicated energy consumption models; for example, models that permit varying the energy consumed by machines by varying their processing mode (i.e., we can reduce the processing time of an operation if we consume more energy) [47], models that consider shifting energy costs through the time horizon [54], models where the machines can be Turn off/Turn on [21], or even where they can also be switched to Stand-By state [46].

In order to extend the proposed metaheuristics to tackle energy models where machines can be turned off or stand-by, the main things that should be addressed are: 1) the schedule builder and 2) the energy optimization procedures. The schedule builder should decide, between every two consecutive tasks in a machine, if it is worth to switch the machine Off or to Stand-By state, or remain it in idle state. That decision would depend on the length of the idle interval, the energy consumed in each state, and the energy/time required for transitions between states. Hence, only if the interval is large enough it is probably worth to switch off the machine. Additionally, the energy optimization procedures should be completely modified. Delaying some tasks may, again, improve the schedule, but not in the same way as in our current problem because if, for example, in some idle interval the machine is already off, that interval could be increased without spending additional energy (assuming that the machine consumes no energy when off). In summary, this is a very interesting line for future work.

Another research direction is detailed in [85], where the authors try to minimize energy consumption and machine error using automated planning in order to vary the energy consumed by machine tools in between the working intervals. The machines can be in several different states, each consuming a different amount of energy. This work is interesting, as the use of automated planning in scheduling problems is a very promising approach, and also the machine error metric is another objective function which can be very relevant in real applications.

References

[1] Pinedo M. Planning and Scheduling in Manufacturing and Services. Springer, 2005.

[2] Garey M, Johnson D, Sethi R. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1976. **1**(2):117–129.

[3] Wisner J, Siferd S. A survey of US manufacturing practices in make-to-order machine shops. *Production and Inventory Management Journal*, 1995. **1**:1–7.

[4] Conner G. 10 questions. *Manufacturing Engineering Magazine*, 2009. pp. 93–99.

[5] Dabia S, Talbi EG, van Woensel T, De Kok T. Approximating multi-objective scheduling problems. *Computers & Operations Research*, 2013. **40**:1165–1175.

[6] Deb K, Pratap A, Agarwal S, Meyarivan T. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 2002. **6**(2):182–197.

[7] Zhang Q, Li H. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *Evolutionary Computation, IEEE Transactions on*, 2007. **11**(6):712–731. doi:10.1109/TEVC.2007.892759.

[8] Miettinen K. Nonlinear Multiobjective Optimization. International Series in Operations Research & Management Science. Springer US, 2012. ISBN 9781461555636. URL `https://books.google.it/books?id=bnzjBwAAQBAJ`.

[9] González MA, Oddi A, Rasconi R. Multi-Objective Optimization in a Job Shop with Energy Costs through Hybrid Evolutionary Techniques. In: Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS-2017). AAAI Press, 2017 pp. 140–148.

[10] Adams J, Balas E, Zawack D. The shifting bottleneck procedure for job shop scheduling. *Managament Science*, 1988. **34**:391–401.

[11] Beck JC, Feng T, Watson JP. Combining Constraint Programming and Local Search for Job-Shop Scheduling. *Informs Journal on Computing*, 2011. **23**:1–14. doi:10.1287/ijoc.1100.0388.

[12] Nowicki E, Smutnicki C. An Advanced Tabu Search Algorithm For The Job Shop Problem. *Journal of Scheduling*, 2005. **8**(2):145–159. doi:10.1007/s10951-005-6364-5.

[13] Singer M, Pinedo M. A computational study of branch and bound techniques for minimizing the total weighted tardiness in job shops. *IIE Transactions*, 1998. **30**:109–118.

[14] Singer M, Pinedo M. A shifting bottleneck heuristic for minimizing the total weighted tardiness in a job shop. *Naval Research Logistics*, 1999. **46**(1):1–17.

[15] Kreipl S. A large step random walk for minimizing total weighted tardiness in a job shop. *Journal of Scheduling*, 2000. **3**:125–138.

[16] Essafi I, Mati Y, Dauzère-Pérès S. A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Computers & Operations Research*, 2008. **35**:2599–2616.

[17] Mati Y, Dauzere-Peres S, Lahlou C. A General Approach for Optimizing Regular Criteria in the Job-Shop Scheduling Problem. *European Journal of Operational Research*, 2011. **212**:33–42.

[18] Bülbül K. A Hybrid shifting bottleneck-tabu serach heuristic for the job shop total weighted tardiness problem. *Computers & Operations Research*, 2011. **38**:967–783.

[19] González MA, González-Rodríguez I, Vela C, Varela R. An efficient hybrid evolutionary algorithm for scheduling with setup times and weighted tardiness minimization. *Soft Computing*, 2012. **16**:2097–2113.

[20] Kuhpfahl J, Bierwirth C. A study on local search neighborhoods for the job shop scheduling problem with total weighted tardiness objective. *Computers & Operations Research*, 2016. **66**:44–57.

[21] Mouzon G, Yildirim MB, Twomey J. Operational methods for minimization of energy consumption of manufacturing equipment. *International Journal of Production Research*, 2007. **45**(18–19):4247–4271.

[22] Yildirim MB, Mouzon G. Single-Machine Sustainable Production Planning to Minimize Total Energy Consumption and Total Completion Time Using a Multiple Objective Genetic Algorithm. *IEEE Transactions on Engineering Management*, 2012. **59**(4):585–597. doi:10.1109/TEM.2011.2171055.

[23] Shrouf F, Ordieres-Meré J, García-Sánchez A, Ortega-Mier M. Optimizing the production scheduling of a single machine to minimize total energy consumption costs. *Journal of Cleaner Production*, 2014. **67**:197–207. doi:10.1016/j.jclepro.2013.12.024.

[24] Liu Y, Tiwari A. An investigation into minimising total energy consumption and total completion time in a flexible job shop for recycling carbon fiber reinforced polymer. *Procedia CIRP*, 2015. **29**:722–727. doi:10.1016/j.procir.2015.01.063.

[25] Liu Q, Zhan M, Chekem FO, Shao X, Ying B, Sutherland JW. A hybrid fruit fly algorithm for solving flexible job-shop scheduling to reduce manufacturing carbon footprint. *Journal of Cleaner Production*, 2017. **168**:668–678. doi:10.1016/j.jclepro.2017.09.037.

[26] Yin L, Li X, Gao L, Lu C, Zhang Z. A novel mathematical model and multi-objective method for the low-carbon flexible job shop scheduling problem. *Sustainable Computing: Informatics and Systems*, 2017. **13**:15–30. doi:10.1016/j.suscom.2016.11.002.

[27] Wu X, Sun Y. A green scheduling algorithm for flexible job shop with energy-saving measures. *Journal of Cleaner Production*, 2018. **172**:3249–3264. doi:10.1016/j.jclepro.2017.10.342.

[28] Mokhtari H, Aliakbar H. An energy-efficient multi-objective optimization for flexible job shop scheduling problem. *Computers & Chemical Engineering*, 2017. **104**:339–352. doi:10.1016/j.compchemeng.2017.05.004.

[29] Piroozfard H, Wong KY, Wong WP. Minimizing total carbon footprint and total late work criterion in flexible job shop scheduling using an improved multi-objective genetic algorithm. *Resources, Conservation and Recycling*, 2018. **128**:267–283. doi:10.1016/j.resconrec.2016.12.001.

[30] He Y, Li Y, Wu T, Sutherland JW. An energy-responsive optimization method for machine tool selection and operation sequence in flexible machining job shops. *Journal of Cleaner Production*, 2015. **87**:245–254. doi:10.1016/j.jclepro.2014.10.006.

[31] Zhang L, Tang Q, Wu Z, Wang F. Mathematical modeling and evolutionary generation of rule sets for energy-efficient flexible job shops. *Energy*, 2017. **138**:210–227. doi:10.1016/j.energy.2017.07.005.

[32] Zhang Y, Wang J, Liu Y. Game theory based real-time multi-objective flexible job shop scheduling considering environmental impact. *Journal of Cleaner Production*, 2017. **167**:665–679. doi:10.1016/j.jclepro.2017.08.068.

[33] Zhang H, Dai Z, Zhang W, Zhang S, Wang Y, Liu R. A new energy-aware flexible job shop scheduling method using modified biogeography-based optimization. *Mathematical Problems in Engineering*, 2017. **Article ID 7249876**:12 pages. doi:10.1155/2017/7249876.

[34] Fang K, Uhan N, Zhao F, Sutherland JW. A new approach to scheduling in manufacturing for power consumption and carbon footprint reduction. *Journal of Manufacturing Systems*, 2011. **30**(4):234–240. doi:10.1016/j.jmsy.2011.08.004.

[35] Fang K, Uhan NA, Zhao F, Sutherland JW. Flow shop scheduling with peak power consumption constraints. *Annals of Operations Research*, 2013. **206**(1):115–145. doi:10.1007/s10479-012-1294-z.

[36] Luo H, Du B, Huang GQ, Chen H, Li X. Hybrid flow shop scheduling considering machine electricity consumption cost. *International Journal of Production Economics*, 2013. **146**(2):423–439. doi:10.1016/j.ijpe.2013.01.028.

[37] Dai M, Tang D, Giret A, Salido MA, Li W. Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm. *Robotics and Computer-Integrated Manufacturing*, 2013. **29**:418–429.

[38] Mansouri SA, Aktas E, Besikci U. Green scheduling of a two-machine flowshop: Trade-off between makespan and energy consumption. *European Journal of Operational Research*, 2016. **248**(3):772–788. doi:10.1016/j.ejor.2015.08.064.

[39] Liu CH, Huang DH. Reduction of power consumption and carbon footprints by applying multi-objective optimisation via genetic algorithms. *International Journal of Production Research*, 2014. **52**(2):337–352. doi:10.1080/00207543.2013.825740.

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

[41] Ji M, Wang JY, Lee WC. Minimizing resource consumption on uniform parallel machines with a bound on makespan. *Computers & Operations Research*, 2013. **40**(12):2970–2974. doi:10.1016/j.cor.2013.06.011.

[42] Wu X, Che A. A memetic differential evolution algorithm for energy-efficient parallel machine scheduling. *Omega*, 2019. **82**:155–165. doi:10.1016/j.omega.2018.01.001.

[43] Salido MA, Escamilla J, Barber F, Giret A. Rescheduling in job-shop problems for sustainable manufacturing systems. *Journal of Cleaner Production*, 2017. **162, Supplement**:S121–S132. doi:10.1016/j.jclepro.2016.11.002.

[44] Lei D, Guo X. An effective neighborhood search for scheduling in dual-resource constrained interval job shop with environmental objective. *International Journal of Production Economics*, 2015. **159**:296–303. doi:10.1016/j.ijpe.2014.07.026.

[45] Liu Y, Dong H, Lohse N, Petrovic S, Gindy N. An investigation into minimising total energy consumption and total weighted tardiness in job shops. *Journal of Cleaner Production*, 2014. **65**:87–96.

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

[47] Zhang R, Chiong R. 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*, 2016. **112**:3361–3375.

[48] Liu Y, Dong H, Lohse N, Petrovic S. A multi-objective genetic algorithm for optimisation of energy consumption and shop floor production performance. *International Journal of Production Economics*, 2016. **179**:259–272. doi:10.1016/j.ijpe.2016.06.019.

[49] Salido MA, Escamilla J, Giret A, Barber F. A genetic algorithm for energy-efficiency in job shop scheduling. *The International Journal of Advanced Manufacturing Technology*, 2016. **85**(5–8):1303–1314. doi:10.1007/s00170-015-7987-0.

[50] González MA, Oddi A, Rasconi R. A multi-objective memetic algorithm for solving job shops with a non-regular energy cost. In: Proceedings of the 11th Workshop on Constraint Satisfaction Techniques for Planning and Scheduling (COPLAS-2016). 2016 pp. 15–24.

[51] Oddi A, Rasconi R, González MA. A constraint programming approach for the energy-efficient job shop scheduling problem. In: Proceedings of the 8th Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA-2017). 2017 pp. 158–172.

[52] Artigues C, Lopez P. Energetic reasoning for energy-constrained scheduling with a continuous resource. *Journal of Scheduling*, 2015. **18**(3):225–241. doi:10.1007/s10951-014-0404-y.

[53] Kim S, Meng C, Son YJ. Simulation-based machine shop operations scheduling system for energy cost reduction. *Simulation Modelling Practice and Theory*, 2017. **77**:68–83. doi:10.1016/j.simpat.2017.05.007.

[54] Grimes D, Ifrim G, O'Sullivan B, Simonis H. Analyzing the impact of electricity price forecasting on energy cost-aware scheduling. *Sustainable Computing: Informatics and Systems*, 2014. **4**(4):276–291.

[55] Paolucci M, Anghinolfi D, Tonelli F. Facing energy-aware scheduling: a multi-objective extension of a scheduling support system for improving energy efficiency in a moulding industry. *Soft Computing*, 2017. **21**(13):3687–3698. doi:10.1007/s00500-015-1987-8.

[56] Baker K. Introduction to Sequencing and Scheduling. Wiley, 1974.

[57] Bürgy R, Bülbül K. The job shop scheduling problem with convex costs. *European Journal of Operational Research*, 2018. **268**(1):82–100. doi:10.1016/j.ejor.2018.01.027.

[58] Van Laarhoven P, Aarts E, Lenstra K. Job shop scheduling by simulated annealing. *Operations Research*, 1992. **40**:113–125.

[59] Brandimarte P, Maiocco M. Job shop scheduling with a non-regular objective: a comparison of neighbourhood structures based on a sequencing/timing decomposition. *International Journal of Production Research*, 1999. **37**(8):1697–1715.

[60] Bierwirth C. A Generalized Permutation Approach to Jobshop Scheduling with Genetic Algorithms. *OR Spectrum*, 1995. **17**:87–92.

[61] Liefooghe A, Humeau J, Mesmoudi S, Jourdan L, Talbi EG. On dominance-based multiobjective local search: design, implementation and experimental analysis on scheduling and traveling salesman problems. *Journal of Heuristics*, 2012. **18**(2):317–352. doi:10.1007/s10732-011-9181-3.

[62] Knowles JD, Corne DW. Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy. *Evolutionary Computation*, 2000. **8**(2):149–172. doi:10.1162/106365600568167.

[63] Paquete L, Schiavinotto T, Stützle T. On local optima in multiobjective combinatorial optimization problems. *Annals of Operations Research*, 2007. **156**:83–97. doi:10.1007/s10479-007-0230-0.

[64] Lara A, Sánchez G, Coello Coello CA, Schütze O. HCS: A New Local Search Strategy for Memetic Multiobjective Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 2010. **14**(1):112–132. doi:10.1109/TEVC.2009.2024143.

[65] Ishibuchi H, Hitotsuyanagi Y, Tsukamoto N, Nojima Y. Use of biased neighborhood structures in multiobjective memetic algorithms. *Soft Computing*, 2009. **13**(8–9):795–810. doi:10.1007/s00500-008-0352-6.

[66] Jaszkiewicz A. Do multiple-objective metaheuristics deliver on their promises? A computational experiment on the set-covering problem. *IEEE Transactions on Evolutionary Computation*, 2003. **7**(2):133–143. doi:10.1109/TEVC.2003.810759.

[67] Papadimitriou C, Steiglitz K. Combinatorial Optimization: Algorithms and Complexity. Dover Books on Computer Science. Dover Publications, 1982. ISBN 9780486402581. URL https://books.google.it/books?id=u1RmDoJqkF4C.

[68] Sakkout H, Wallace M. Probe Backtrack Search for Minimal Perturbation in Dynamic Scheduling. *Constraints*, 2000. **5**(4):359–388. doi:10.1023/A:1009856210543.

[69] Giagkiozis I, Purshouse RC, Fleming PJ. Generalized Decomposition. In: Purshouse RC, Fleming PJ, Fonseca CM, Greco S, Shaw J (eds.), Evolutionary Multi-Criterion Optimization. Springer Berlin Heidelberg, Berlin, Heidelberg. ISBN 978-3-642-37140-0, 2013 pp. 428–442. doi:10.1007/978-3-642-37140-0.

[70] Miettinen K. Nonlinear Multiobjective Optimization. International Series in Operations Research & Management Science. Springer US, 1998. ISBN 978-0-7923-8278-2.

[71] Palacios JJ, Derbel B. On Maintaining Diversity in MOEA/D: Application to a Biobjective Combinatorial FJSP. In: GECCO '15 Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation. ACM, 2015 pp. 719–726. doi:10.1145/2739480.2754774.

[72] Li H, Zhang Q. Multiobjective Optimization Problems With Complicated Pareto Sets, MOEA/D and NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 2009. **13**(2):284–302. doi:10.1109/TEVC.2008.925798.

[73] Marquet G, Derbel B, Liefooghe A, Talbi EG. Shake Them All! Rethinking Selection and Replacement in MOEA/D. In: Bartz-Beielstein T, Branke J, Filipič B, Smith J (eds.), Parallel Problem Solving from Nature – PPSN XIII, volume 8672 of *Lecture Notes in Computer Science*, pp. 641–651. Springer. ISBN 978-3-319-10761-5, 2014. doi:10.1007/978-3-319-10762-2_63.

[74] Apt K. Principles of Constraint Programming. Cambridge University Press, New York, NY, USA, 2003. ISBN 0521825830.

[75] Vilím P, Barták R, Cepek O. Unary Resource Constraint with Optional Activities. In: Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings. 2004 pp. 62–76.

[76] Le Pape C, Baptiste P, Nuijten W. Constraint-Based Scheduling: Applying Constraint Programming to Scheduling Problems. Springer Science+Business Media, New York, NY, USA, 2001. ISBN 978-1-4613-5574-8.

[77] Laborie P. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence*, 2003. **143**(2):151–188.

[78] IBM. IBM CPLEX Optimization Studio V12.7.1. URL https://www.ibm.com/support/knowledgecenter/SSSA5P_12.7.1/ilog.odms.studio.help/Optimization_Studio/topics/COS_home.html.

[79] Fisher H, Thompson G. Probabilistic learning combinations of local job-shop scheduling rules. In: Muth J, Thompson G (eds.), Industrial Scheduling, pp. 225–251. Prentice Hall, 1963.

[80] Lawrence S. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement). Graduate School of Industrial Administration, Carnegie-Mellon University, 1984.

[81] Applegate D, Cook W. A computational study of the job-shop scheduling problem. *ORSA Journal of Computing*, 1991. **3**:149–156.

[82] Baker KR. Sequencing rules and due-date assignments in a job shop. *Management Science*, 1984. **30**(9):1093–1104. doi:10.1287/mnsc.30.9.1093.

[83] Sabuncuoglu I, Bayiz M. Job shop scheduling with beam search. *European Journal of Operational Research*, 1999. **118**(2):390–412.

[84] Zitzler E, Thiele L. Multiobjective optimization using evolutionary algorithms — A comparative case study. In: Parallel Problem Solving from Nature — PPSN V Proceedings, pp. 292–301. Springer Berlin Heidelberg, 1998.

[85] Parkinson S, Longstaff A, Fletcher S, Vallati M. On the Exploitation of Automated Planning for Reducing Machine Tools Energy Consumption Between Manufacturing Operations. In: Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling (ICAPS-2017). AAAI Press, 2017 pp. 400–408.