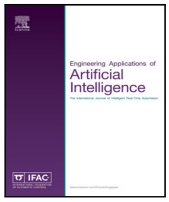
Engineering Applications of Artificial Intelligence 115 (2022) 105263

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

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



Metaheuristics for multiobjective optimization in energy-efficient job shops

Miguel A. González [a,*], Riccardo Rasconi [b], Angelo Oddi [b]

[a] Department of Computing, University of Oviedo, Spain
[b] Institute of Cognitive Sciences and Technologies, ISTC-CNR, Italy

ARTICLE INFO

Keywords:
Evolutionary algorithms
Heuristic search
Constraint programming
Mixed-Integer Linear Programming
Job-shop scheduling
Energy aware scheduling

ABSTRACT

Energy awareness is one of the most relevant research directions in scheduling problems. In this paper we consider the minimization of both the makespan and the energy consumption in the classical job shop scheduling problem. The energy model considered allows several possible states for the machines: *off*, *stand-by*, *idle*, *setup* and *processing*. To solve this multi-objective problem we propose an NSGA-II based evolutionary algorithm combined with local search and a heuristic procedure to improve the energy consumption of a given schedule. We also propose an advanced constraint programming (CP) approach as well as a Mixed-Integer Linear Programming (MILP) model, to the aim of comparing their performances against those obtained with the NSGA-II. The experimental study is performed against a benchmark set that extends by 41 instances of increasing size, the set tackled in the previous literature against the same problem. The experiments demonstrate the superiority of the NSGA-II algorithm over all other methods, despite the utilization of CP and MILP allows to draw interesting conclusions on the overall solution optimality, revealing that there is still room for further optimization.

1. Introduction

The main motivation for studying the job shop is that it is a model close to many real environments. However, its complexity is proven to be NP-hard, and therefore all kinds of resolution methods were proposed in the literature in the last decades, from exact methods to metaheuristic algorithms. The classical job shop with makespan minimization is the most studied variant, by far (see Adams et al. (1988), Beck et al. (2011) or Nowicki and Smutnicki (2005)). However, many different constraints and characteristics have been considered to make the problem even more realistic, for example setup times, flexibility in the machine selection, minimum and maximum time lags between operations or uncertainty. In the last years, energy considerations are becoming a very relevant research topic, both for environmental and economical reasons.

In Gao et al. (2019), Li and Wang (2022) we can read a couple of recent reviews of papers related to energy-efficient scheduling. Usually, these are multiobjective problems, minimizing at the same time a performance related objective (for example the makespan) and an energy related objective. The most interesting approaches for solving multi-objective problems are those based on the Pareto set, as they provide more flexibility to the decision makers in the real environments.

Many different energy-aware scheduling problems are considered in the literature, as for example single machine (Mouzon et al., 2007) or flexible flow shop (Dai et al., 2013). There are also some papers tackling real-life problems, as for example (Faria et al., 2019), where the authors minimize the energy consumption of a yeast production factory using a genetic algorithm. In some simulations, they estimate a 2.29% reduction in electricity cost, which represents about 7500 euro savings to the factory each year and a reduction of its environmental impact.

The energy-efficient job shop is probably the most interesting; in fact, according to Gao et al. (2019), 41% of the papers about green scheduling solve some variant of the job shop. We can cite (Liu et al., 2014), where a quite simplistic energy model is considered where the resources can only be idle or processing. Their results are improved in González et al. (2019) by using NSGA-II and MOEA/D multiobjective evolutionary metaheuristics hybridized with local search, and also a constraint-programming method. This last work is an extension of a previous work (González et al., 2017). Another interesting paper is (Jiang et al., 2018), where the authors consider a single-objective minimization of the sum of the energy-consumption cost and the completion-time cost. They consider varying speeds for processing the operations; obviously faster processing implies more energy consumed. In Escamilla and Salido (2018) the authors also consider three different processing modes for the operations, although in this work the main objective of their memetic algorithm is to obtain robust schedules, capable of absorbing possible incidences. A job shop considering flexibility and transportation times is tackled in Zhang et al. (2019) and a NSGA-II based evolutionary metaheuristic is proposed to minimize

* Corresponding author.
E-mail addresses: mig@uniovi.es (M.A. González), riccardo.rasconi@istc.cnr.it (R. Rasconi), angelo.oddi@istc.cnr.it (A. Oddi).

https://doi.org/10.1016/j.engappai.2022.105263
Received 30 March 2021; Received in revised form 30 June 2022; Accepted 21 July 2022
Available online 4 August 2022

both energy and makespan. In Zhang and Chiong (2016) the weighted tardiness and energy consumption are both minimized, considering an energy model in which the processing mode of operations can be modified. In He et al. (2022) the authors study an energy-efficient job-shop scheduling problem with sequence-dependent setup times where machines have three different speeds for performing operations. They aim to simultaneously minimize the makespan, total tardiness and total energy consumption. A novel fitness evaluation mechanism based on fuzzy relative entropy was developed to effectively evaluate and select solutions. This mechanism is exploited within a general multiobjective optimization framework that uses an adaptive local search strategy and a hybrid genetic algorithm approach. Experiments show that their proposal outperforms other multiobjective metaheuristics, as NSGA-II, SPEA-II, NSGA-III, MOEA/D and MOPSO.

In this paper we are particularly interested in job shop scenarios in which the machines can be in several states. This multiple-state energy model provides more possibilities and flexibility for some real environments, but its resolution can also be more difficult than other simpler models. The importance of such models for the realization of realistic scenarios is proven by their recent utilization in the scheduling literature. One recent example of such model is (Li and Lei, 2021) in which the authors analyze a flexible job shop floor where the machines exist in three modes: *Processing* mode, *Stand-by* mode and *Setup* mode, and the machine consumptions while in *Processing* mode depend on the speed selected for that particular job. In general, the importance of "energy-aware" (or *green*) scheduling problems is also underscored in the very recent survey (Li and Wang, 2022), where the problems like the one tackled in the present work belong to the important class of green scheduling problems related to the minimization of the non-value energy consumption. In fact, there are recent papers tackling real-world problems, as in the work (Jiang et al., 2022) where a energy-efficient flexible job shop is analyzed by using a case study for the aerospace industry complex components in China. In particular, the most relevant work for the present paper is (May et al., 2015), where it is proposed an energy model where the machines can be in five states: *Idle*, *Processing*, *Setup*, *Off*, or *Stand-by* and the machine speeds are fixed. The authors solve the problem with a multiobjective genetic algorithm that minimizes both the makespan and the energy consumption by building a Pareto set of solutions. However, their proposal always schedules tasks at the earliest possible time, and we will see in Section 2.2 that this is not the best approach to tackle the problem. In Oddi et al. (2017) their results are improved by a constraint-programming approach that also uses a piecewise linear programming step to further improve the energy consumption of the schedules. The authors generate the Pareto set by applying the ϵ-constraint method (Miettinen, 2012). Afterwards, in Oddi et al. (2018) the same authors propose an upgrade to their constraint-programming method that allows improvements in some of the instances of the benchmark. The upgrade consists in adding an additional set of *energy aware constraints* that allow to prune some of the decisions on variables.

In this paper we further study the problem proposed in May et al. (2015) and develop several solving methods for it, with the hopes of improving the results reported in May et al. (2015), Oddi et al. (2017, 2018). In particular we propose the following methods:

- A hybrid evolutionary algorithm based on NSGA-II combined with local search and a heuristic procedure for reducing the energy consumption of a solution.
- A constraint programming (CP) approach, slightly modified with respect to that proposed in Oddi et al. (2017, 2018).
- A novel Mixed-Integer Linear Programming (MILP) model.

It is expected that the hybrid evolutionary algorithm will perform the best. In fact, in Gao et al. (2019) it is stated that 59% of the papers about energy-aware scheduling problems are solved by means of swarm and evolutionary algorithms, and among those, 52% are genetic algorithms of different types (including NSGA-II). Therefore, it is a widely used approach due to its effectiveness in tackling optimization problems with vast search spaces.

The remaining of this paper is organized as follows: in Section 2 we formally define the problem, whereas in Sections 3–5 we describe the proposed solving methods, respectively the evolutionary algorithm, the constraint programming approach and the mixed-integer programming approach. Then, Section 6 reports the details of the experimental study and the comparison with the state of the art. Finally, Section 7 summarizes the conclusions and proposes some ideas for future research.

2. Problem formulation

In the classical job shop scheduling problem we must schedule a set of N jobs $J = \{J_1, \ldots, J_N\}$ in a set of M resources, $R = \{R_1, \ldots, R_M\}$. Each job J_i is composed by n_i operations $(\theta_{i1}, \ldots, \theta_{in_i})$ that must be scheduled in that order. The set of all operations is denoted as Ω. Each operation θ_{ij} requires the uninterrupted and exclusive use of a particular resource $r_{ij} \in R$ during all its processing time, denoted p_{ij}. The goal is to establish the starting time s_{ij} of all operations such that all constraints are fulfilled, i.e. we want to determine a feasible schedule.

Therefore, the constraints are:

- Precedence constraints, that represent the routing of the operations within each job: $s_{\theta_{ij}} + p_{\theta_{ij}} \leq s_{\theta_{ij+1}} \forall i \in J, \forall j \in \{1, \ldots, n_i - 1\}$
- Capacity constraints, that represent the fact that each machine can only process one operation at a time: $(s_{\theta_{ij}} + p_{\theta_{ij}} \leq s_{\theta_{kl}}) \vee (s_{\theta_{kl}} + p_{\theta_{kl}} \leq s_{\theta_{ij}}) \; \forall i, j, k, l$ such that $r_{ij} = r_{kl}$

In order to simplify notation, in the remaining of the paper we will denote operations by a single letter instead of θ_{ij} whenever possible. Also, given a feasible schedule, PJ_v and SJ_v will respectively denote the predecessor and successor of operation v in its job sequence, whereas PM_v and SM_v will respectively denote the predecessor and successor of operation v in its resource sequence. Additionally, the first and last operations on resource R_k in the considered schedule will be denoted α_k and ω_k, respectively.

As in many papers in the literature, we propose to find a schedule that minimizes two objective functions at the same time: the makespan and the energy consumption.

The makespan is the overall completion time and it is defined as $\max_{u \in \Omega}\{s_u + p_u\}$. It is the most studied objective function in scheduling literature.

In order to define the energy consumption, we must first describe the considered energy model, taken from May et al. (2015). Each machine can be in five different states: *Off*, *Stand-by*, *Idle*, *Setup* or *Processing*. For simplicity, the *Setup* and *Processing* states are considered together, which is a reasonable assumption often taken in the literature, and it is particularly suitable when setup times are not sequence-dependent.

When a resource R_k is in *Off* state it consumes no energy, whereas when in *Idle*, *Stand-by* or *Processing* states its power consumption is, respectively, P_k^{idle}, $P_k^{stand\text{-}by}$ and $P_k^{processing}$. Regarding the transitions, for simplicity we assume that there are some transitions which consume no time and no energy: from *Idle* to *Stand-by*, *Off* or *Processing*, and also from *Processing* to *Idle*. On the contrary, the transition from *Off* to *Idle* requires a power consumption of $P_k^{ramp\text{-}up}$ during $T_k^{ramp\text{-}up\text{-}off}$ time units, and the transition from *Stand-by* to *Idle* requires the same power consumption $P_k^{ramp\text{-}up}$ during $T_k^{ramp\text{-}up\text{-}stand\text{-}by}$ time units. Notice that $T_k^{ramp\text{-}up\text{-}stand\text{-}by}$ should be lower than $T_k^{ramp\text{-}up\text{-}off}$, as it is easier to prepare a resource for processing when in *Stand-by* state than when it is *Off*.

The main motivation for assuming that some transitions require no time and no energy is simply to use exactly the same model as in May et al. (2015). However, these assumptions do not limit the usefulness

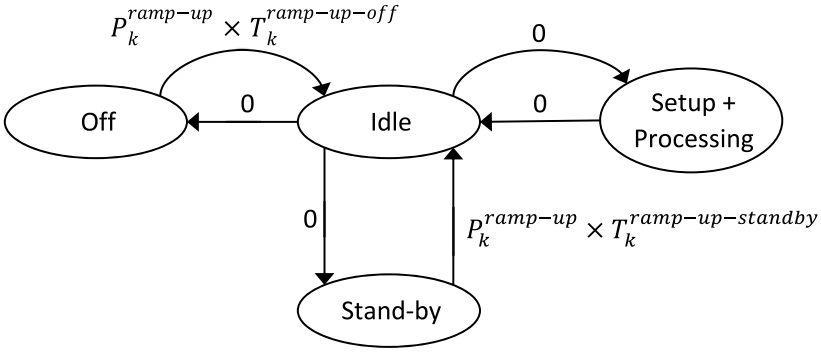


Fig. 1. State diagram of a machine; each transition is labeled with the energy consumed in it.

of the model, as explained in the following: the transitions from *Idle* to *Off* and from *Idle* to *Stand-by* can be simplified and assumed to be zero because the time/energy required can be included in the opposite transitions (i.e. from *Off* to *Idle* and from *Stand-by* to *Idle*), therefore modifying the parameters $T_k^{ramp\text{-}up\text{-}off}$, $T_k^{ramp\text{-}up\text{-}stand\text{-}by}$ and $P_k^{ramp\text{-}up}$. As for the transitions from *Idle* to *Processing* and vice versa, we can obtain similar results by adding them to the processing time of the corresponding operations. The described states of each machine and the transitions between them are summarized in Fig. 1.

For the sake of simplicity, we also assume that machines do not have any power consumption before the processing of their first operation assigned and after the processing of their last operation assigned.

As in this paper we are considering a job shop scheduling problem with no flexibility in resource selection, we can remark that each resource must always process the same set of operations, and so the total energy consumption of each resource when in *Processing* state must be the same in every feasible schedule. For this reason, in order to minimize the energy consumption, following (May et al., 2015), we must minimize the following measure, denoted WEC (Worthless Energy Consumption):

$$WEC = \sum_{k=1,\dots,M} [P_k^{idle}\, t_k^{idle} + P_k^{stand\text{-}by}\, t_k^{stand\text{-}by}] + \sum_{k=1,\dots,M} P_k^{ramp\text{-}up} (n_k^{ramp\text{-}up\text{-}standby}\, T_k^{ramp\text{-}up\text{-}standby} + n_k^{ramp\text{-}up\text{-}off}\, T_k^{ramp\text{-}up\text{-}off}) \quad (1)$$

where t_k^{idle} and $t_k^{stand\text{-}by}$ are the total time spent by the resource R_k in *Idle* and *Stand-by* states, respectively, and $n_k^{ramp\text{-}up\text{-}standby}$ and $n_k^{ramp\text{-}up\text{-}off}$ are the number or transitions done by R_k from *Stand-by* to *Idle* and from *Off* to *Idle*, respectively.

Following (Baker, 1974), in any given schedule a regular performance measure can be increased only by increasing at least one of the completion times of an operation. To minimize a regular measure it is sufficient to consider the so-called "left-shift schedules", in which, given a partial ordering of the operations, each one starts at the earliest possible time allowed by the previous operations in the partial ordering. In this paper we want to minimize the makespan, which is a regular measure, and the WEC, which is not regular. Notice that the WEC can sometimes be decreased by increasing the completion time of some operations, as we will see in Section 2.2.

2.1. Definition of multi-objective optimization problems

As pointed in He et al. (2019) or (Zhu et al., 2018), a multi-objective optimization problem with q objectives can be defined as follows:

$$Minimize\{f_1(\mathbf{x}), f_2(\mathbf{x}), \dots, f_q(\mathbf{x})\} \text{ subject to } \mathbf{x} \in X, \quad (2)$$

where $f_i, i = 1, \dots, q$ $(q \geq 2)$, are the possibly conflicting objective functions that must be minimized simultaneously, and X is the set of all feasible solutions. In our particular work, $q = 2$, as we consider two objective functions.

Table 1
Data for the operations of the toy instance.

	Job 1			Job 2			Job 3		
	θ_{11}	θ_{12}	θ_{13}	θ_{21}	θ_{22}	θ_{23}	θ_{31}	θ_{32}	θ_{33}
Processing time	4	5	2	2	5	3	4	7	3
Required resource	R_1	R_2	R_3	R_1	R_3	R_2	R_2	R_1	R_3

Some papers tackle multiobjective problems in a simple way, by means of a lexicographic approach or a weighted sum approach, but the dominance-based methods are more complex and interesting. In particular, when we have two objective functions f_1 and f_2 that we want to minimize, a solution S is Pareto dominated, or simply *dominated* by a solution S' (denoted $S' \prec S$), if and only if $f_1(S') \leq f_1(S)$ and $f_2(S') < f_2(S)$, or $f_1(S') < f_1(S)$ and $f_2(S') \leq f_2(S)$, i.e. S' is better in at least one objective function and it is never worse in any objective function.

It usually happens that a unique solution cannot be optimal with respect to both objectives. In this work we are looking for the set of all non-dominated solutions or *Pareto optimal* solutions. These solutions are not dominated by any solution $S' \in X$, and so the improvement of one objective necessarily implies the worsening of the other objective. The *Pareto front* PS^* is defined as the set of all objective function values corresponding to the solutions in the Pareto optimal set.

2.2. Solution example

The toy example presented in this section will help to better illustrate the described problem. Consider 3 jobs with 3 operations in each job, and 3 resources. Table 1 illustrates the data of these operations:

Consider also the following values for all resources R_k with $k \in \{1, 2, 3\}$: $P_k^{processing} = 10kW$, $P_k^{idle} = 6kW$, $P_k^{stand\text{-}by} = 4kW$, $P_k^{ramp\text{-}up} = 8kW$, $T_k^{ramp\text{-}up\text{-}off} = 3$ and $T_k^{ramp\text{-}up\text{-}stand\text{-}by} = 1$.

In Fig. 2 we show a feasible schedule for this toy instance, with a WEC of 40 (24 from R_3 plus 16 from R_2) and a makespan of 18. Between each pair of consecutive operations in the same resource we have to decide if we leave the resource in *Idle* state, or if we switch it to either *Off* or *Stand-by* state. For example, in the 3 time units between operations θ_{31} and θ_{23} we switch R_2 to *Stand-by* state because in this way it adds 16 to the WEC, whereas switching it *Off* would add 24 and if *Idle* would add 18. The same reasoning is used to decide to switch R_3 to the *Off* state between operations θ_{22} and θ_{33}. Notice that this feasible schedule is a "left-shift schedule", as no operation can start earlier without modifying its partial ordering.

Delaying operations can sometimes lead to improved schedules, as it can be seen in Fig. 3, in which we delay operation θ_{31}. Now, the best option between the end of operation θ_{31} and the beginning of operation

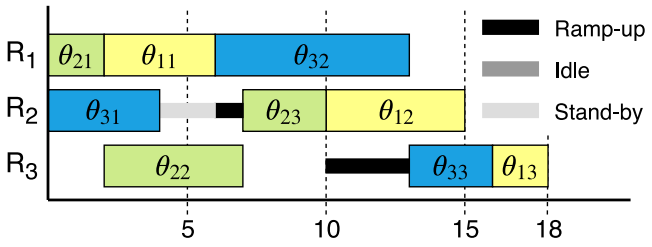


Fig. 2. Feasible solution for the toy instance using a "left-shift schedule".

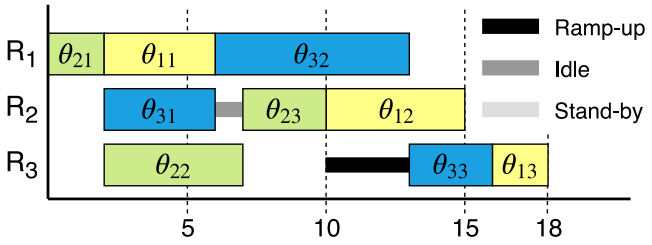


Fig. 3. Delaying one operation in order to improve the solution of Fig. 2.

θ_{23} is to leave resource R_2 in *Idle* state. In this way the WEC is lowered from 40 to 30, whereas the makespan is still 18.

Hence, the WEC is a non-regular performance measure. It is worth to remark that the paper (May et al., 2015) only considers "left-shift schedules", and so there is clearly a lot of room for improvement by delaying operations. The papers (Oddi et al., 2017) and Oddi et al. (2018) are a first step in considering the non-regularity of the WEC measure in this problem.

2.3. The disjunctive graph model

In this section we will define a disjunctive graph model, which is the most common representation in scheduling problems. It will be used in the neighborhood structure described in Section 3.5.1.

We follow a similar representation as other papers in the literature, as for example (Van Laarhoven et al., 1992). A problem instance can be represented by a directed graph $G = (V, A \cup D)$. Each node of set V is an operation of the problem, except dummy nodes *start* and *end* which are fictitious operations with processing time 0. The arcs of set A are called *conjunctive arcs* and represent precedence constraints, whereas arcs of set D are denoted *disjunctive arcs* and represent capacity constraints. Set A contains some additional arcs from the fictitious operation *start* to the first operation of each job, all of them weighted with value 0, and arcs from the last operation of each job to the fictitious operation *end*, all of them weighted with the processing time of the operation at the source node, p_v. The remaining arcs (v, w) of A and D are also weighted with the processing time of the operation at the source node, p_v. Set D can be partitioned into subsets D_j, with $j = 1, \ldots, M$, where D_j corresponds to resource R_j and includes two directed arcs (v, w) and (w, v) for each pair of operations v, w such that $r_v = r_w = R_j$.

A feasible schedule is represented by an acyclic subgraph G_S of G, $G_S = (V, A \cup H)$ where $H = \cup_{j=1\ldots M} H_j$, H_j being a Hamiltonian selection of D_j, i.e. a minimal subset of arcs from D_j that define a processing order for all operations that require R_j. Hence, finding a solution can be reduced to discovering compatible orderings H_j, or partial schedules, that translate into a solution graph G_S without cycles.

The makespan of the schedule is the cost of a *critical path*, defined as a longest path from node *start* to node *end*. Nodes and arcs in a critical path are termed *critical*. A critical path can be decomposed into a sequence $start, B_1, \ldots, B_r, end$ where each B_k, $1 \leq k \leq r$ is called a *critical block*, defined as a maximal subsequence of consecutive operations of the critical path requiring the same resource.

Many solving methods of the job shop literature rely on the concepts of critical path and critical block, including the local search neighborhood proposed in Van Laarhoven et al. (1992), which we adopt and is described in Section 3.5.1. We remark that this representation is useful for neighborhood structures devoted to makespan minimization, but it is not that relevant for WEC minimization, as this objective function is not directly related to finding largest cost paths in a graph representation.

Finally, we present an example in Fig. 4, where it is shown a disjunctive graph that corresponds to the feasible schedule represented in Fig. 2. Continuous arcs belong to set A whereas dotted arcs belong to set H. The critical path is marked with bold arcs, and it has two critical blocks: $B_1 = (\theta_{21}\theta_{11}\theta_{32})$ and $B_2 = (\theta_{33}\theta_{13})$.

3. Dominance-based multi-objective hybrid evolutionary algorithm

In this section we describe a dominance-based hybrid metaheuristic, adapting the successful ideas presented in González et al. (2017, 2019) to the problem considered in this paper. The proposal combines a genetic algorithm based on the NSGA-II framework (Deb et al., 2002) with a multi-objective local search method. The main differences between our proposal and those previous works are those derived from the modified energy model, which is now more complex, and so the energy optimization heuristic (see Section 3.2) should be completely redefined, and also the linear programming step used in those previous papers is substituted by a more exhaustive version of the proposed energy optimization heuristic. Additionally, in this paper we minimize the makespan instead of the weighted tardiness, and so several components, as for example the neighborhood structure used in the local search (see Section 3.5.1) is different because the concepts of critical path and critical block do change.

It is well-known that the minimization of non-regular objectives in the JSP is more difficult than that of regular objectives. As an example, in Brandimarte and Maiocco (1999) the authors propose the decomposition of the overall problem in the sequencing and timing subproblems. We are using a similar approach, and to solve the sequencing subproblem we represent solutions as permutations of jobs, whereas to tackle the timing subproblem we introduce heuristic energy optimization procedures.

3.1. Representation and evaluation of solutions

The most common representation for job shop scheduling problems is permutations with repetitions (Bierwirth, 1995), in which a solution is represented by a permutation of the set of operations, each denoted by its job number. Hence, each job number must appear as many times in the permutation as number of operations it has. For example, if we consider a problem instance with three jobs and three operations in each job, a chromosome (2 2 3 1 3 2 3 1 1) represents the operation ordering $(\theta_{21}\theta_{22}\theta_{31}\theta_{11}\theta_{32}\theta_{23}\theta_{33}\theta_{12}\theta_{13})$.

To create a schedule from a chromosome, an insertion algorithm is applied to it: we iterate the operations in the ordering given by the chromosome, assigning each operation the earliest possible starting time that fulfills all constraints with respect to all operations previously scheduled. Notice that we are using an insertion heuristic, and so the final ordering might be different than that of the chromosome (for example if we are able to insert an operation in a big enough "gap" that is located before an already scheduled operation. Hence, after building the schedule, we reconstruct the chromosome in order to represent the final ordering of the operations. As soon as the schedule is built, we can compute its makespan and energy consumption.

It is worth to remark that any permutation containing as many repetitions of a given number as number of operations of the corresponding job, represents a linear ordering compatible with precedence constraints, and so a feasible solution can always be built. As an

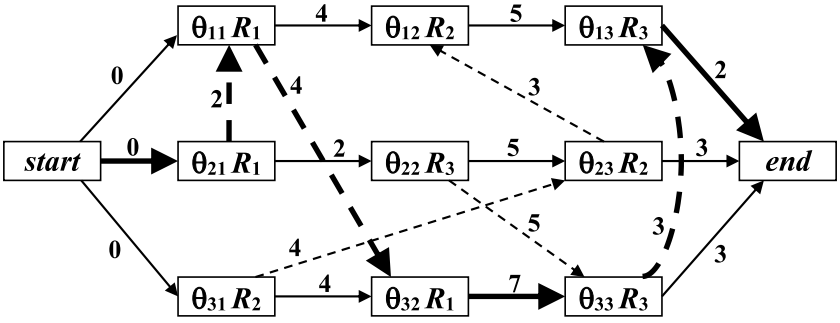


Fig. 4. Disjunctive graph of the feasible solution represented in Fig. 2. Bold arcs show a critical path whose length, i.e. the makespan, is 18.

example, the feasible schedule represented in Fig. 2 can be obtained from the chromosome (2 2 3 1 3 2 3 1 1). Notice that it can also be obtained from the chromosome (3 2 2 2 1 3 1 3 1) and many others, as several linear orderings are compatible with the represented solution.

3.2. Energy optimization heuristic

As we have already pointed out, the WEC objective function is non-regular, and therefore scheduling each operation as soon as possible is probably not the best option, as illustrated in Section 2.2.

In order to improve the schedules created by the insertion strategy described in Section 3.1 we propose a novel energy optimization heuristic. It is applied to the solution returned by the insertion strategy and it tries to improve the WEC of the schedule without modifying the linear ordering of the operations and also without increasing the makespan. In this way, the ordering of the chromosome tackles the sequencing subproblem while this energy optimization heuristic solves the timing subproblem. Algorithm 1 details the procedure with its two main steps.

The first step is based on the energy post-optimization procedure originally proposed in González et al. (2017), and it tries to delay as much as possible all operations of the schedule, with the exception of the last operation processed in each resource, which must remain in its same starting time. When delaying operations we always maintain their relative ordering in each resource and we of course take into account precedence constraints between operations of the same job, in order to obtain a resulting feasible schedule. Also notice that the makespan cannot be increased, as we are never delaying the last operation processed in each resource.

This step can improve the WEC, as processing several consecutive operations in a resource with no idle time between them consumes less energy than if those operations were separated by some time. The schedule resulting of delaying operations as described usually results in more "grouping" of operations, thus reducing the WEC. However, the resulting schedule can usually still be improved by applying another step.

The second step of the heuristic iterates all operations of the chromosome, and for each one it sets its best possible starting time such as the WEC is minimized, considering only its current space (i.e. we cannot move any other operation and we cannot modify the operation ordering). Fig. 5 shows an example where we have to choose the best starting time for operation i from the range [73,82]. This can be easily calculated, as we only have to find the starting time that minimizes $E_1 + E_2$ where E_1 is the WEC consumed between operations PM_i and i and E_2 is the WEC consumed between operations i and SM_i.

However, doing a single iteration of all operations of the chromosome might not be enough and further improvements in WEC can be obtained using a recursive approach. This recursive approach, when it actually moves an operation a of the schedule, instead of continuing iterating the next position of the chromosome, it goes back to the earliest position in the chromosome between the job predecessor of a and the resource predecessor of a. In this way we can tackle the cases where moving an operation might make desirable moving an earlier operation. Evidently, this results in a procedure of much higher complexity. We apply it by setting the parameter *Recursive* to *true* in Algorithm 1.

Algorithm 1: The energy optimization heuristic

input : A problem instance I and a feasible schedule S (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
  //First step: delaying operations
  $k \leftarrow |\Omega|$;
  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** $s'_a \leftarrow s'_{SM_a} - p_a$;
      else $s'_a \leftarrow \min\{s'_{SJ_a}, s'_{SM_a}\} - p_a$;
    end
    $k \leftarrow k - 1$;
  end
  //Second step: moving operations inside its space
  $k \leftarrow 1$;
  while $k \leq |\Omega|$ **do**
    $a \leftarrow O[k]$;
    //Calculate the earliest starting time for operation a
    $s_a^{min} \leftarrow 0$;
    if *exists* PJ_a **then** $s_a^{min} \leftarrow s'_{PJ_a} + p_{PJ_a}$;
    if *exists* PM_a **then** $s_a^{min} \leftarrow \max\{s_a^{min}, s'_{PM_a} + p_{PM_a}\}$;
    //Calculate the latest starting time for operation a
    $s_a^{max} \leftarrow$ makespan of the schedule $S - p_a$;
    if *exists* SJ_a **then** $s_a^{max} \leftarrow s'_{SJ_a} - p_a$;
    if *exists* SM_a **then** $s_a^{max} \leftarrow \min\{s_a^{max}, s'_{SM_a} - p_a\}$;
    //Select the best starting time for a that minimizes the energy
    $s_a^{opt} \leftarrow$ the time $\in [s_a^{min}, s_a^{max}]$ such that the WEC is minimized. In case of tie select the earliest possible starting time between those tied;
    //Determine the next operation to study
    if $Recursive = true \wedge s_a^{opt} \neq s'_a$ **then**
      $k \leftarrow \max\{$position of PJ_a in ordering O, position of PM_a in ordering $O\}$;
    else
      $k \leftarrow k + 1$;
    end
    $s'_a \leftarrow s_a^{opt}$;
  end
end

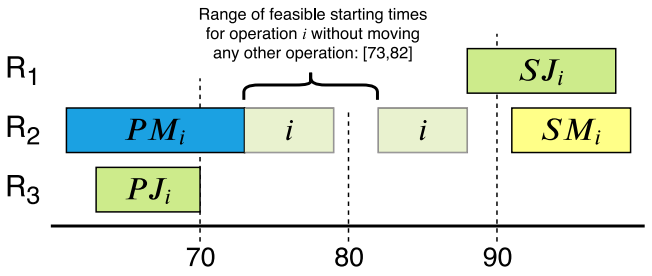


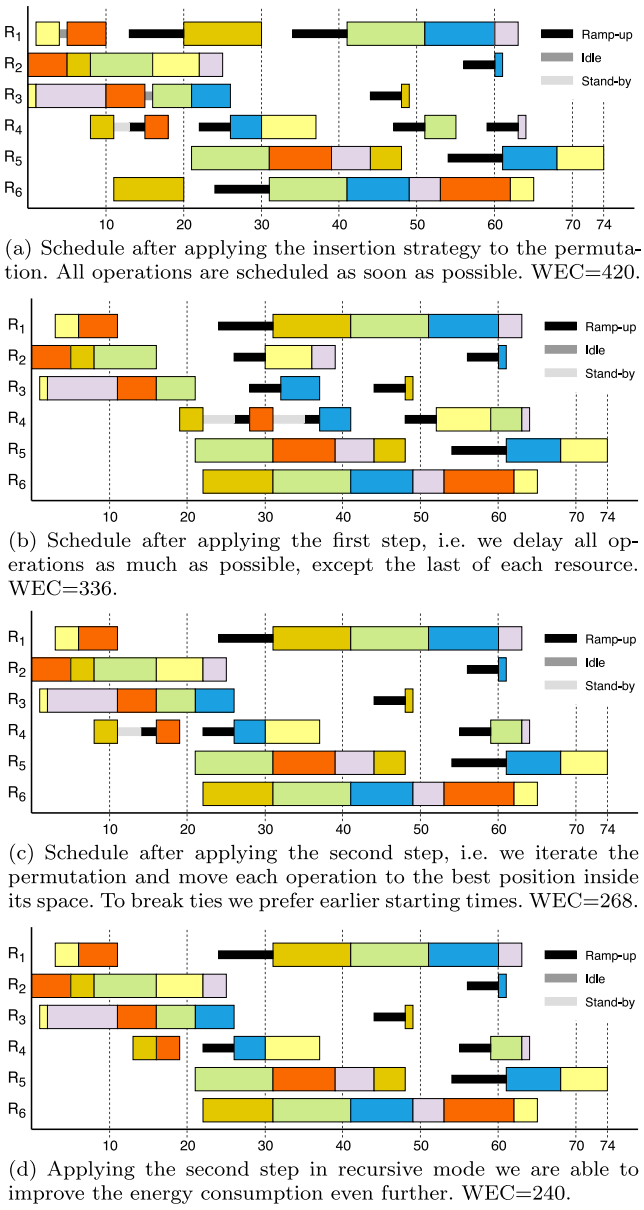
Fig. 5. Second step of the energy optimization heuristic: moving an operation i to the optimal position in its space such that WEC is minimized.

We propose to embed the described algorithm in the solution evaluation method and we apply it just after the insertion strategy described in Section 3.1, in order to try to improve the WEC of the resulting schedule. Therefore, this energy optimization heuristic is applied when evaluating every chromosome generated by the genetic algorithm and every neighbor considered in the local search described in Section 3.5.

Our approach is to always use the standard procedure for the second step (i.e. $Recursive = false$), and use the recursive procedure (i.e. $Recursive = true$) only to evaluate the final population of the last generation of the genetic algorithm. In this way, we can further improve the WEC of the final solutions, but we can maintain a reasonable computational burden. Notice that applying the recursive version all the time would require too much computational time.

Fig. 6 shows an example of the application of the heuristic in the well-known FT06 instance, with 6 jobs having 6 operations each, and 6 resources. The energy consumptions and ramp-up times are those described in Section 6.1. In these figures, each job is colored with different colors, in order to better distinguish the precedence constraints.

We consider the permutation (1 1 5 4 4 4 6 2 1 2 3 6 6 4 6 5 2 4 3 5 2 6 3 1 6 5 2 2 3 4 3 3 1 1 5 5). Fig. 6(a) shows the initial schedule after the application of the insertion heuristic, with a WEC of 420. Notice that all operations are scheduled as soon as possible. Then, after the first step of the heuristic the resulting schedule is that depicted in Fig. 6(b) with a WEC of 336, where we see that operations are delayed as much as possible, with the exception of the last operation executed in each resource. The schedule after the application of one iteration of the second step is shown in Fig. 6(c) with a WEC of 268. We notice that some operations were scheduled earlier in order to improve the energy consumption. Finally, if we applied the second step in recursive mode, the WEC can be even further improved to 240, as we can see in Fig. 6(d). We can notice that the recursive mode, in this case, only modifies one operation with respect to the standard mode of the second step, however we have experimentally seen that in larger instances it usually performs more moves.



(a) Schedule after applying the insertion strategy to the permutation. All operations are scheduled as soon as possible. WEC=420.

(b) Schedule after applying the first step, i.e. we delay all operations as much as possible, except the last of each resource. WEC=336.

(c) Schedule after applying the second step, i.e. we iterate the permutation and move each operation to the best position inside its space. To break ties we prefer earlier starting times. WEC=268.

(d) Applying the second step in recursive mode we are able to improve the energy consumption even further. WEC=240.

Fig. 6. Example of energy optimization heuristic (Algorithm 1).

3.3. Genetic operators

The first step of the genetic algorithm is the creation and evaluation of the initial population formed by $popSize$ random solutions. Then, in each generation, a set of $popSize$ offspring solutions is built by applying selection, crossover and mutation operators to the current population. The selection phase chooses which chromosomes will be parents in order to generate new offspring solutions. We adopt a tournament strategy in which, to select each parent, we choose $tSize$ chromosomes at random from the population and select the "best" of them, according to the criteria based on non-domination level and crowding distance described in Section 3.4. The parameter $tSize$ must be carefully set, as a low value may produce too much randomness in the selection, whereas a too large value may result in a low diversity in subsequent generations.

Then, the Job Order Crossover operator (JOX) (Bierwirth, 1995) is applied to each pair of parents with probability $crProb$ in order to generate two offspring solutions. The JOX is a crossover operator specifically designed for job shop problems when using permutations with repetitions. It selects a random subset of jobs and copies them from the first parent to the first offspring solution in the same positions as they are in the parent. The offspring is then completed taking the remaining jobs from the second parent in its same relative ordering. To create the second offspring solution the parents simply reverse their role. We clarify how JOX works by means of an example. Let us consider the following two parents:

$Parent1$	**2**	3	**2**	**2**	1	1	3	3
$= 1$								
$Parent2$	1	1	3	3	**2**	3	1	**2**
$= 2$								

If the selected subset of jobs is that marked in bold in the parents, i.e. only job 2, then the generated offspring is:

$Offspring1 = 1$	1	**2**	**2**	3	3	3	1
$Offspring2 = 2$	3	1	1	**2**	3	3	**2**

Each offspring solution is then mutated with $mutProb$ probability, and the mutation operator consists on swapping two randomly selected positions of the chromosome.

3.4. Replacement strategy

The replacement strategy is the main difference between a standard single-objective genetic algorithm and a multi-objective genetic algorithm, which in our case is based on the NSGA-II framework. We have to select the "best" $popSize$ solutions to build the next population, choosing between the $popSize$ solutions of the current population and the $popSize$ offspring solutions just created by means of the selection, crossover and mutation operators.

We adopt the strategy proposed in Deb et al. (2002) which is based in selecting solutions from lower non-domination levels, and using the crowding distance to break ties when not every solution from a given level can advance to the next population. We refer the interested reader to Deb et al. (2002) in order to see full details of the procedure.

In order to improve the diversity of the new population, we adopt an additional strategy taken from González et al. (2017) and González et al. (2019) that eliminates some individuals before applying the replacement strategy. It basically consists on removing the duplicated-fitness individuals from the pool of solutions, which is applicable when multiple solutions exist in the pool that share same values for all the objective functions. A similar procedure is also used in May et al. (2015).

3.5. Local search

It is frequent to hybridize genetic algorithms with local searchers in order to improve their results, because genetic algorithms have strong diversification capabilities whereas local searchers have strong intensification capabilities. Such combination is usually termed *memetic algorithm*. However, the design of multi-objective local searchers can be difficult, as the dominance relation $\prec$ only defines a partial order, and so selecting the "best" neighbor is a non-trivial issue. In this work we take the multi-objective hill climbing local search method proposed in González et al. (2017, 2019), which is fast and efficient and it is specifically designed to be combined with a multi-objective genetic algorithm.

The selection of the best neighbor is based on the dominance relation, but it also considers the current set of non-dominated solutions of the population of the genetic algorithm. The method is detailed in Algorithm 2. It starts from an initial solution S', then it generates one of its neighbors (using the neighborhood structure detailed in Section 3.5.1) and evaluates it (after applying the energy optimization heuristic described in Section 3.2). Then, we check if the neighbor S'' fulfills one of the following requirements:

1. $S'' \prec S'$.
2. $\nexists S1 \in P$ such that $S' \prec S1$ and $\exists S2 \in P$ such that $S'' \prec S2$, where P is the set of non-dominated solutions of the current population of the genetic algorithm.

Notice that the second requirement allows the local search to select a neighbor even if it does not dominate the current solution. The idea is to allow selecting neighbors able to improve the current set of non-dominated solutions of the genetic algorithm, in case that the current solution is not able to improve that set.

If at least one of those two requirements is met, we substitute the current solution S' for its neighbor S'' and we restart the procedure. In other case, we try with a different neighbor. The algorithm ends when no neighbor of the current solution met any of those requirements, i.e. we reached a local optimum. Then, the current solution S' is returned and the chromosome is rebuilt from this improved schedule, so its characteristics can be inherited in subsequent generations of the genetic algorithm, an effect known as Lamarckian evolution. As this hill-climbing based local search is not computationally costly, we are able to apply it to every chromosome of the initial population of the genetic algorithm, and also to all new offspring chromosomes created just after performing crossover and mutation.

Algorithm 2: Multi-objective local search based on hill climbing

input : A feasible schedule S for a problem instance I
output: A hopefully improved (with respect to the current set of non-dominated solutions of the genetic algorithm) solution S' for instance 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'' ≺ S', or S'' would improve 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
```

3.5.1. Neighborhood structure

The neighborhood adopted in this paper was initially proposed in Van Laarhoven et al. (1992) and is one of the most used in the job shop literature. It relies on the concepts of critical block and critical path, described in Section 2.3. In particular, a neighbor of a solution is created by reversing a single critical arc from a critical block. It can be proven that reversing a single critical arc cannot form a cycle in the resulting graph and so it always produces a feasible schedule, hence repairing procedures are not needed.

Consider, for example, the schedule depicted in Fig. 7 for the toy instance described in Section 2.2. Its critical path is marked in bold arcs: (θ_{21}, θ_{22}, θ_{23}, θ_{12}, θ_{13}, θ_{33}), and its length (i.e. the makespan of the schedule) is 20. It has two critical blocks: (θ_{23}, θ_{12}) and (θ_{13}, θ_{33}). Then, we consider two possible neighbors: (1) reverse critical arc ($\theta_{23}\theta_{12}$) and (2) reverse critical arc ($\theta_{13}\theta_{33}$). Fig. 4 shows the disjunctive graph of the second neighbor, with a makespan of 18. In principle, this neighborhood structure is designed for makespan minimization, but we have experimentally seen that it often improves the WEC as well.

When minimizing makespan, there exist in the literature some non-improving conditions and algorithms for fast estimation of the neighbors' makespan. However it is much more difficult to determine if the WEC will improve or not, and so we are not using these type of conditions in our work.

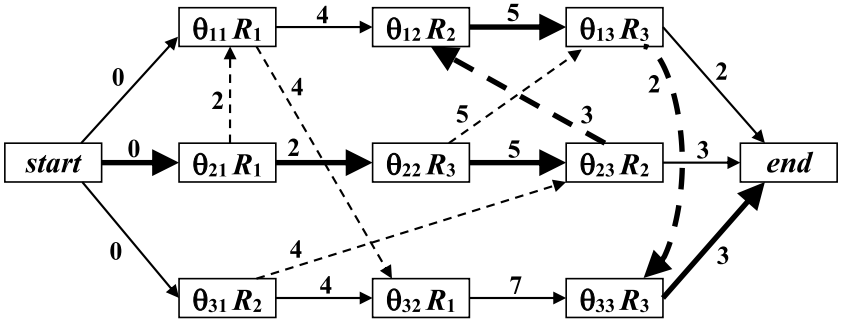


Fig. 7. Disjunctive graph of a feasible solution with makespan 20. Its critical path is marked in bold arcs, and it has two critical blocks: (θ_{23}, θ_{12}) and (θ_{13}, θ_{33}).

4. A constraint programming approach

In this section we describe a procedure that takes into account the non-regularity of the WEC objective such that an approximation of the Pareto front is generated by a Constraint Programming (CP) procedure. It is worth noting that the proposed CP approach is in principle able to find an optimal WEC value if given sufficient computational time (we do not provide any formal proof about this property).

4.1. A CP-based energy optimization model

Constraint Programming (CP) is a declarative programming paradigm (Apt, 2003) that allows to define programs as a set of *decision variables*, each ranging on a discrete domain of values, and a set of *constraints* that limit the possible combination of variable-value assignments. Once a *model* of the problem is created, the solver interleaves two main steps: *search*, such that values are assigned to decision variables, and *constraint propagation*, where inconsistent values are removed from variable domains. CP is particularly suited for solving scheduling problems where the decision variables are associated to the problem operations. In particular, each operation variable a has at least two features: s_a representing its start time, and p_a representing its duration. For scheduling problems, a number of different *global constraints* have been proposed in the literature, the most important being: (i) the `unary-resource` constraint (Vilím et al., 2004) for modeling simple machines, such that the constraint holds if and only if all the assigned operations never overlap at any time point; (ii) the `cumulative` resource constraint (Le Pape et al., 2001) for modeling cumulative resources (e.g., a pool of workers); (iii) the `reservoir` (Laborie, 2003) for modeling consumable resources (e.g., a fuel tank).

We describe a CP model based on the problem defined in Section 2, where the main *decision variables* are the start times s_a of the operations $a \in \Omega$ characterized by a processing time p_a. Each start time s_a ranges in the interval $[0, H - p_a]$, where H is the problem's horizon. The set of decision variables is then extended with the start times s_{OnOff_k} of the $OnOff_k$ intervals, where each $OnOff_k$ interval is defined as spanning over all the operations executed on machine k. Hence, the s_{OnOff_k} variable represents the first instant when machine k is turned on. The model is built on top of the IBM-ILOG CPLEX Optimization Studio CP Optimizer 12.10 and will be tested in Section 6. Its details are as follows.

Let O_k be the set of problem operations assigned to machine $k = 1, \dots, M$ and U_k be a set of auxiliary *unit-duration operations*, assigned to a dummy unary machine mirroring k (it is worth noting that the two sets O_k and U_k represent separate processing orders of activities). The introduction of the auxiliary set of operations U_k.[1] is necessary to represent the position of each activity $a \in O_k$ in the processing orders imposed among the operations assigned to each machine $k \in R$. More concretely, the auxiliary unit-duration operations indirectly implement the definition of a *successor function* SM_a (returning the successor of each operation a for each total order imposed on the set of operations O_k assigned to a machine k). To the best of our knowledge, this workaround is necessary because we want to use the native OPL construct to implement the global constraints `unary` $-$ `resource`(O_k) for efficiency reasons, and the successor function is not natively present in the OPL language (see IBM ILOG CPLEX Optimization Studio OPL Language Reference Manual, Version 12 Release 10).

Operationally, the set of *unit-duration* operations $u \in U_k$ can be assigned to the dummy machine k (in the same fashion of the operations a) so that, for each processing order[2] imposed on a machine k, $a_0 \prec a_1 \prec \dots \prec a_i \prec \dots \prec a_M$, an identical order is imposed on the unit-duration operations $u_0 \prec u_1 \prec \dots \prec u_i \prec \dots \prec u_M$. In this manner, the position i of the operation a_i coincides with the start-time value of the unit-duration operation u_i. For the reasons above, the starting times s_u of the operations $u \in U_k$ must be added to the model as additional set of *decision variables*. In addition, a specific global constraint is added in the CP model given below to impose the same order among the activities in the sets O_k and U_k, see constraints (4i).

$$SM_p = \begin{cases} q & \exists u^{(q)} \in U_k : s_{u^{(q)}} = s_{u^{(p)}} + 1 \\ nil & \text{otherwise} \end{cases} \tag{3a}$$

$$\begin{aligned} E^k_{pq} = min\{ & P^{idle}_k \, d_{pq}, \\ & P^{stand\text{-}by}_k \, (d_{pq} - T^{ramp\text{-}up\text{-}standby}_k) + P^{ramp\text{-}up}_k \, T^{ramp\text{-}up\text{-}standby}_k, \\ & P^{ramp\text{-}up}_k \, T^{ramp\text{-}up\text{-}off}_k \} \end{aligned} \tag{3b}$$

$$WEC = \sum_{k=1,\dots,M} \sum_{\substack{p \in O_k, \\ q = SM_p, q \neq nil}} E^k_{pq} \tag{3c}$$

$$C_{max} = \max_{a \in \Omega} \{ s_a + p_a \} \tag{3d}$$

The definition (3a) represents the successor function SM_p, such that the position of the operation $p \in O_k$ coincides with the start-time value $s_{u^{(p)}}$ of its corresponding unit-duration operation $u^{(p)} \in U_k$, and the successor q (if exists) corresponds to the unary activity $u^{(q)} \in U_k$, such that $s_{u^{(q)}} = s_{u^{(p)}} + 1$. Whereas, according to Section 2, the energy objective *WEC* (3c) is the total amount of the worthless energy consumption E^k_{pq} (i.e., when a machine is *Idle*, switched *Off*, or switched to a *Stand-by* state) of each pair of contiguous operations (p, q) assigned on the same machine k (3b), where $d_{pq} = s_q - e_p$ is the difference between q's start time and p's end time. The makespan objective C_{max} is described at line (3d).

Once all the necessary definitions have been provided and all the variables have been introduced, we present the CP model (optimization

[1] We were inspired to adopt this solution by a post on a discussion board on the website www.or-exchange.com about the explicit representation of an interval position in a OPL `sequence` This discussion board seems no longer available.

[2] Here, we use for the precedence relation the same formalism used earlier for the dominance relation. The difference is clear by the context.

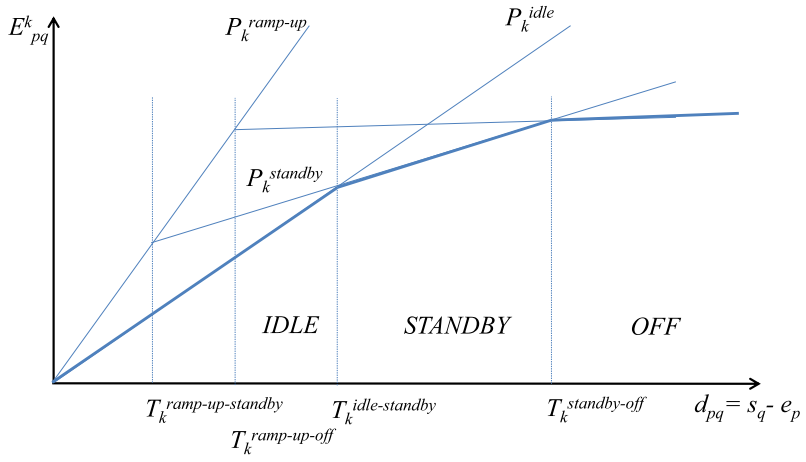


Fig. 8. Minimal energy consumption E^k_{pq} between two consecutive operations (p, q).

criteria and constraints). Line (4a) represents the lexicographic minimization of the objective pair *WEC* and C_{max} with the *WEC* as primary objective. According to the implemented ϵ-constraint method (Miettinen, 2012) for calculating the Pareto set, we optimize the energy *WEC*, while we impose an upper bound to the other objective C_{max} in the form $C_{max} \leq C_\epsilon$ (see (4b)). The constraints in (4c) represent the linear orderings imposed on the set of operations Ω by the set of jobs J. Constraints (4d) impose to the set O_k of operations requiring machine k to be contained in the *spanning* operations $OnOff_k$, $k = 1, \ldots, M$. More specifically, for each operation $v \in O_k$, the following constraints $s_{OnOff_k} \leq s_v$ and $s_v + p_v \leq s_{OnOff_k} + p_{OnOff_k}$ hold, such that operation $OnOff_k$ starts together with the *first* present operation in O_k according to the order imposed on the kth machine, and ends together with the *last* present operation.

Constraints (4f), (4g), and (4h) impose that the minimal energy is consumed between the end of the first and the beginning of the second task, for each pair of contiguous activities (p, q) on a resource k. These constraints rely on the assumption that $P_k^{stand\text{-}by} \leq P_k^{idle} \leq P_k^{ramp\text{-}up}$ and $T_k^{ramp\text{-}up\text{-}standby} \leq T_k^{ramp\text{-}up\text{-}off}$; under such assumptions, there are two cutoff values, $T_k^{idle\text{-}standby}$ and $T_k^{standby\text{-}off}$ (depicted in Fig. 8), such that if $s_v - e_u \in [0, T_k^{idle\text{-}standby}]$ the minimal energy state is *Idle*, when $s_v - e_u \in (T_k^{idle\text{-}standby}, T_k^{standby\text{-}off}]$ the minimal energy state is *Stand-by*, otherwise the minimal energy state is *Off*.

$$\text{lex min}\,(WEC, C_{max}) \tag{4a}$$

s.t. :

$$C_{max} \leq C_\epsilon \tag{4b}$$

$$s_v + p_v \leq s_{SJ_v} \quad v \in \Omega \setminus \{\theta_{1n_1}, .., \theta_{Nn_N}\} \tag{4c}$$

$$\texttt{span}(OnOff_k, O_k) \quad k = 1, .., M \tag{4d}$$

$$ed_p \in \{0, 1, 2\} \; p \in \Omega \tag{4e}$$

$$SM_p = q \wedge (ed_p = 0) \Rightarrow s_q - e_p \leq T_k^{idle\text{-}standby} \tag{4f}$$

$$SM_p = q \wedge (ed_p = 1) \Rightarrow s_q - e_p > T_k^{idle\text{-}standby} \wedge s_q - e_p \leq T_k^{standby\text{-}off} \tag{4g}$$

$$SM_p = q \wedge (ed_p = 2) \Rightarrow s_q - e_p > T_k^{standby\text{-}off} \tag{4h}$$

$$\texttt{same-sequence}(O_k, U_k) \; k = 1, \ldots, M \tag{4i}$$

$$s_u \leq (|O_k| - 1) \; u \in U_k; \; k = 1, \ldots, M \tag{4j}$$

$$\texttt{unary-resource}(O_k) \; k = 1, \ldots, M \tag{4k}$$

$$\texttt{unary}-\texttt{resource}(U_k) \; k = 1, \ldots, M \tag{4l}$$

$$\Delta t_k^{proc} + \Delta t_k^{standby} + \Delta t_k^{off} \leq C_{max} \quad k = 1, \ldots, M \tag{4m}$$

We introduce a set of decision variables $ed_p \in \{0, 1, 2\}$, $p \in \Omega$ (constraint (4e)) representing the unload state (i.e., 0 when machine is *Idle*, 1 when it is switched to a *Stand-by* state, and 2 when switched *Off*) imposed on every pair of contiguous activities (p, q) on the same machine. The constraints in (4i) impose the same order between the activities in the two sets O_k and U_k by means of the global constraints $\texttt{same-sequence}(O_k, U_k)$. The constraints in (4j) bound the start-time value of each unit-duration operation u to $|O_k| - 1$ operations assigned to the machine k. (4k) and (4l) represents the non-overlapping constraints imposed by the machines M to the operations in O_k and U_k, through the global constraints $\texttt{unary} - \texttt{resource}(O_k)$ and $\texttt{unary} - \texttt{resource}(U_k)$, respectively.

Finally, (4m) represent the so-called *energy aware constraints* imposed on the subset of (energy) decision variables ed_p associated to each subset of operations O_k, $k = 1, \ldots, M$. The rationale behind these constraints is the following: for each machine k, the set of operations O_k requiring that machine must be totally ordered. In addition, according to the values of the decision variables ed_u, with $u \in O_k$, a minimum (non zero) delay equal to $T_k^{idle\text{-}standby}$ (when $ed_u = 1$) or $T_k^{standby\text{-}off}$ (when $ed_u = 2$), must be inserted between the operation u and its successor (if it exists). Hence, each machine's total order has a lower-bound of the total execution time (from the start-time of the first operation to the end-time of the last one) which can be calculated as the sum of the three terms $\Delta t_k^{proc} + \Delta t_k^{stand\text{-}by} + \Delta t_k^{off}$, such that: $\Delta t_k^{proc} = \sum_{u \in O_k} p_u$ is the sum of the operation processing times in machine k; $\Delta t_k^{standby} = \sum_{u \in O_k, ed_u = 1} T_k^{idle\text{-}standby}$ is the minimum total delay due to *Stand-by* states; $\Delta t_k^{off} = \sum_{u \in O_k, ed_u = 2} T_k^{standby\text{-}off}$ is the minimum total delay due to *Off* states. Such lower-bound cannot be greater than the solution makespan C_{max}, hence decisions on the variables ed_u can be pruned according to the constraints (4m).

The previous proposed model basically coincides with the one proposed in Oddi et al. (2018), in Section 6 we evaluate it against the hybrid evolutionary algorithm described in Section 3. In particular, we will consider a larger benchmark set (size of instances ranging from 36 to 300 activities) than the one used in Oddi et al. (2018), where the sizes of the instances were from 36 to 100 activities. Under these new conditions, as we will see in Section 6, the model proposed in Oddi et al. (2018) degrades its performance as the size of the instances is greater than 100 activities. In particular, the above given model within the imposed time limits is able to find a very small number of solutions for each run. Hence, we have tested two different modifications of the constraints contained in the original model, for a total of four different CP models.

– The first proposed modification reverses the lexicographic minimization objective (4a) $lex\ min(WEC, C_{max})$ into $lex\ min(C_{max}, WEC)$.
– In the second one we test the idea of reversing the constraint (4b) $C_{max} \leq C_\epsilon$ into $C_{max} \geq C_\epsilon$. As this choice limits the utility of the propagation rules (4m) for obvious reasons, we will not consider them in the corresponding generated model.

Hence, the idea is to use the four combinations of the proposed modifications in conjunction with the bi-criterion ϵ-constraint method, described in the following Section 4.2, the goal is to increase the number and the quality of the solutions in the output Pareto set. In fact, the use of the objective $lex\ min(C_{max}, WEC)$, with C_{max} as primary key, might leverage the use of the default propagation rules used within the CP Optimizer 12.10. In addition, we note that without the upper bound constraint (4b) the randomized solving process used within the CP Optimizer 12.10 is able to generate a higher number of different solutions, even if the quality of such solutions might be lower than the ones obtained by the contribution of the constraints (4b) and (4m). In fact, as the size of the instances increases, a lighter model (without the constraints (4b) and (4m)) is clearly more scalable.

4.2. The bi-criterion ϵ-constraint method

A well-known multi-objective optimization method to generate the Pareto front is the ϵ-constraint method (Miettinen, 2012). It works by choosing one objective function as the only objective and properly constraining the remaining objective functions during the optimization process. Through a systematic variation of the constraint bounds, different elements of the Pareto front can be obtained. Algorithm 3 presents the ϵ-constraint method for the case of a bi-criterion objective function $\mathbf{f} = (f^{(1)}, f^{(2)})$. The algorithm is used in the experimental section of the work in two different variants.

The first variant takes the following 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, i.e., the CP() procedure corresponding to the constraint programming model previously described. Note that we consider a slightly different ϵ-constraint method, such that the given CP procedure considers a lexicographic minimization instead of single-objective minimization problem, with $f^{(1)}$ as primary and $f^{(2)}$ as secondary key. The algorithm 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 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.

The second variant is similar to the previous one, the only difference is the use of the first component of the objective function $f^{(1)}$ in place of $f^{(2)}$. Hence, the algorithm accepts as an input the bounds $f^{(1)}_{min}$ and $f^{(1)}_{max}$ and proceed analogously as previously described.

Algorithm 3: Bi-criterion ϵ-constraint method

input : The objective $\mathbf{f}$, the bounds $f^{(2)}_{min}$ and $f^{(2)}_{max}$, and the decrement value δ
output: P
begin
  $P \leftarrow \emptyset$;
  $\epsilon \leftarrow f^{(2)}_{max}$;
  while $\epsilon \geq f^{(2)}_{min}$ **do**
    $S \leftarrow$ CP($\mathbf{f}$, 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
  return (P);
end

5. A mixed-integer linear programming approach

In this section we propose a Mixed-Integer Linear Programming (MILP) model (Fourer et al., 2003) for finding a solution to the problem defined in Section 2. The proposed model considers five different sets of *decision variables*.

– The start times s_u of the operations $u \in \Omega$ characterized by a processing time p_u.
– The set of ordering decision variables $S_{uv} \in \{0,1\}$, $u, v \in O_k$, $u \neq v$, $k = 1, \ldots, M$, such that $S_{uv} = 1$ *iff* the simple precedence constraint $u \prec v$ is imposed between the two operations u and v.
• The set of ordering decision variables $X_{uv} \in \{0,1\}$, $u, v \in O_k$, $u \neq v$, $k = 1, \ldots, M$, representing the successor function SM_u (see Section 2) on the imposed machine orderings, such that $X_{uv} = 1$ *iff* $SM_u = v$ on the corresponding machine.
– The set of E_u decision variables, $u \in \Omega$, representing the *worthless energy consumption* (WEC) after the execution of operation u and before the execution of an eventual successor v on the same machine. The previous set of variables is strictly connected to the following set of decision variables, $idle_u, stby_u, off_u \in \{0,1\}$, such that for each operation u, at most only one of the previous variables has the value 1, and such assignment corresponds to turn the machine into one of the corresponding states *Idle*, *Stand-by* or *Off*, just after the execution of the operation u.
– The two objectives: *WEC* and C_{max}.

The MILP model minimizes the total worthless energy consumption *WEC*, see (6a) and (6b), under the two upper-bound constraints on C_{max} (6c) and *WEC* (6b).

Constraints (6e) represent the linear orderings imposed on the set of operations $v \in \Omega$ by the jobs $J = \{J_1, J_2, \ldots, J_N\}$, note that they hold for each operation $v \in \Omega$ except when v is the last operation of a job J_i. Inequalities (6f) impose to the first operation θ_{i1} of each job J_i to start after the reference value 0, whereas constraints (6g) impose to the last operations θ_{in_i} of each job J_i to end before the makespan value C_{max}.

Constraints on the ordering decision variables are the ones from (6h) to (6m). (6h) guarantees that for each pair of operations $u, v \in O_k$, $u \neq v$, either the simple precedence constraint $u \prec v$ or $v \prec u$ is imposed. Whereas the rest of the constraints guarantee that the values assumed by the decision variables X_{uv} represent a successor function SM_u on the machine orderings. In particular, (6i) guarantees that either v is the successor of u or u is the successor of v. Note that the $\leq$ constraint in (6i) includes the particular case $X_{uv}+X_{uv} = 0$, which holds when u is the last activity in a machine ordering. This is also the reason of imposing the set of constraints (6k), which limits the number of successors for each machine ordering to the value $M-1$. (6j) guarantees that X_{uv} orderings do not contradict the S_{uv} orderings. Finally, the two sets of constraints (6l) (6m) impose a set of *flow* constraints, such that, for each activity $u \in O_k$, at most one activity $v \in O_k$, $v \neq u$, is the successor (predecessor) of $u \in O_k$.

The energy decisions $idle_u, stby_u, off_u \in \{0,1\}$ are constrained by set of Eqs. (6n), hence, for each operation u the three variables can exclusively take the value 1, except for the last operation u of each machine k ordering, where $\sum_{v \in O_k} X_{uv} = 0$. The three sets of inequalities (6o), (6p) and (6q) set the worthless energy values according to the

values of the corresponding decision variables $idle_u$, $stby_u$, off_u and X_{uv} as explained in Fig. 8, B is a large positive penalty constant. The assigned values of the energy, according to the possible state values *idle*, *stand-by*, and *off* are defined as follows:

$$E_k^{idle}(e_u, s_v) = P_k^{idle}(s_v - e_u) \tag{5a}$$

$$E_k^{stby}(e_u, s_v) = P_k^{stand\text{-}by}(s_v - e_u - T_k^{ramp\text{-}up\text{-}standby}) + P_k^{ramp\text{-}up} T_k^{ramp\text{-}up\text{-}standby} \tag{5b}$$

$$E_k^{off} = P_k^{ramp\text{-}up} T_k^{ramp\text{-}up\text{-}off} \tag{5c}$$

Lastly, Eq. (6r) guarantees the value $E_u = 0$ for each operation u which is the last operation of a machine ordering.

About the imposed temporal constraints, the set of inequalities (6s) and (6t) represent the set of temporal constraints implied by the set of decision variables S_{uv} and X_{uv}. Whereas, the set of inequalities from (6u) to (6x) represent the imposed temporal interval constraints for each pair of successive operations u and v (see Fig. 8) according to the value of the decision variables $idle_u$, $stby_u$, off_u.

Finally, inequalities (6y) represent a set of constraints on the decision variables $stby_u$, off_u, analogously to the *energy aware* constraints described for the CP model (see Section 4). In this case we introduce two set of bounds LBm_k (UBm_k) $k = 1, \ldots, M$ on the start-times s_u (end-times $s_u + p_u$) of the set of activities $u \in O_k$ requiring machine k. In particular, for the calculus of the bounds LBm_k (UBm_k), we have to consider that each activity $u \in O_k$ is part of a job J_u. Let P^u_{before} (P^u_{after}) the sum of the processing times of the operations in the job ordering J_u before (after) u, $LBm_k = min_{u \in O_k}\{P^u_{before}\}$ ($UBm_k = min_{u \in O_k}\{P^u_{after}\}$). Hence, for each machine k, given the value of the decision variables $stby_u$, off_u, with $u \in O_k$, it is easy to verify that the sum of the terms on the left of the inequalities (6y) is a lower-bound of the makespan and cannot be greater than the value C_{max}.

$$\min WEC \tag{6a}$$

$$s.t.:$$

$$\sum_{u \in \Omega} E_u \leq WEC \tag{6b}$$

$$C_{max} \leq C_0 \tag{6c}$$

$$WEC \leq WEC_0 \tag{6d}$$

$$s_v + p_v \leq s_{SJ_v} \quad v \in \Omega \setminus \{\theta_{1n_1}, \ldots, \theta_{Nn_N}\} \tag{6e}$$

$$0 \leq s_{\theta_{i1}} \quad i = 1, \ldots, N \tag{6f}$$

$$s_{\theta_{in_i}} + p_{\theta_{in_i}} \leq C_{max} \quad i = 1, \ldots, N \tag{6g}$$

$$S_{uv} + S_{vu} = 1 \quad u, v \in O_k, u < v, k = 1, \ldots, M \tag{6h}$$

$$X_{uv} + X_{vu} \leq 1 \quad u, v \in O_k, u < v, k = 1, \ldots, M \tag{6i}$$

$$X_{uv} \leq S_{uv} \quad u, v \in O_k, u \neq v, k = 1, \ldots, M \tag{6j}$$

$$\sum_{u,v \in O_k} X_{vu} = N - 1 \quad k = 1, \ldots, M \tag{6k}$$

$$\sum_{v \in O_k} X_{uv} \leq 1 \quad u \in O_k, k = 1, \ldots, M \tag{6l}$$

$$\sum_{v \in O_k} X_{vu} \leq 1 \quad u \in O_k, k = 1, \ldots, M \tag{6m}$$

$$idle_u + stby_u + off_u = \sum_{v \in O_k} X_{uv} \quad k = 1, \ldots, M \tag{6n}$$

$$B(idle_u + X_{uv} - 2) \leq E_u - E_k^{idle}(e_u, s_v) \leq B(2 - idle_u - X_{uv}) \quad u, v \in O_k, u \neq v, k = 1, \ldots, M \tag{6o}$$

$$B(stby_u + X_{uv} - 2) \leq E_u - E_k^{stby}(e_u, s_v) \leq B(2 - stby_u - X_{uv}) \quad u, v \in O_k, u \neq v, k = 1, \ldots, M \tag{6p}$$

$$B(off_u + X_{uv} - 2) \leq E_u - E_k^{off} \leq B(2 - off_u - X_{uv}) \quad u, v \in O_k, u \neq v, k = 1, \ldots, M \tag{6q}$$

$$E_u \leq E_k^{off}(idle_u + stby_u + off_u) \quad k = 1, \ldots, M \tag{6r}$$

$$s_u + p_u - s_v \leq B(1 - S_{uv}) \quad u, v \in O_k, u \neq v, k = 1, \ldots, M \tag{6s}$$

$$s_u + p_u - s_v \leq B(1 - X_{uv}) \quad u, v \in O_k, u \neq v, k = 1, \ldots, M \tag{6t}$$

$$s_v - (s_u + p_u) - T_k^{idle\text{-}standby} \leq B(2 - idle_u - X_{uv}) \quad u, v \in O_k, u \neq v, k = 1, \ldots, M \tag{6u}$$

$$s_v - (s_u + p_u) - T_k^{idle\text{-}standby} \geq B(2 - stby_u - X_{uv}) \quad u, v \in O_k, u \neq v, k = 1, \ldots, M \tag{6v}$$

$$s_v - (s_u + p_u) - T_k^{standby\text{-}off} \leq B(2 - stby_u - X_{uv}) \quad u, v \in O_k, u \neq v, k = 1, \ldots, M \tag{6w}$$

$$s_v - (s_u + p_u) - T_k^{standby\text{-}off} \geq B(2 - off_u - X_{uv}) \quad u, v \in O_k, u \neq v, k = 1, \ldots, M \tag{6x}$$

$$LBm_k + \sum_{u \in O_k} p_u + T_k^{idle\text{-}standby} \sum_{u \in O_k} stby_u + T_k^{standby\text{-}off} \sum_{u \in O_k} off_u + UBm_k \leq C_{max} \quad k = 1, \ldots, M \tag{6y}$$

6. Experimental results

In this section we will report the results obtained with our methods. The evolutionary metaheuristic is implemented in C++ using a single thread and runs in a Intel Core i5-2450M CPU at 2.5 GHz with 4 GB of RAM, using Windows 10 Pro. The Constraint Programming (CP) method is implemented using IBM-ILOG CPLEX Optimization Studio v12.10, running on a Intel Core i7-7700 CPU @3.60 GHz with 16 GB of RAM, using Ubuntu Linux 16.04. As we are dealing with stochastic algorithms, we perform 10 runs for each method and each instance in order to obtain statistically significant results.

Firstly in Sections 6.1 and 6.2 we will define the benchmark instances and the performance measures used, respectively. Then, in Section 6.3 we will study the best values for the configuration of the parameters used in our evolutionary algorithm. Section 6.4 presents some preliminary experiments to compare the performance of a set of different CP solver configurations. Finally, Section 6.5 presents the results and the comparison with state-of-the-art algorithms.

6.1. Benchmark instances

In the papers related to the state of the art (which, as far as we know, are May et al. (2015), Oddi et al. (2017, 2018)) the authors only considered 3 problem instances in their experimental study, in particular those denoted $FT06$, $FT10$ and $FT20$, and firstly introduced in Fisher and Thomson (1963). The $FT06$ instance has 6 jobs and 6 machines, whereas $FT10$ has 10 jobs and 10 machines and $FT20$ has 20 jobs and 5 machines. The optimal makespan of these instances can be found in the literature and is, respectively, 55, 930 and 1165.

We argue that 3 instances are not enough for extracting meaningful conclusions, and so we are considering 41 extra benchmark instances: the $LA01$ to $LA40$, introduced in Lawrence (1984) for makespan minimization, plus the well-known challenging $ABZ7$ instance. The size of the $LA01 - LA40$ instances vary from 10×5 (the smaller ones) to 30×10 and 15×15 (the largest ones), while the size of the $ABZ7$ instance is 20×15. We have to define the exact power consumptions and transition times, and we propose using for all instances the same parameters described in May et al. (2015) for every resource R_k:

- $P_k^{processing} = 10kW$
- $P_k^{idle} = 6kW$
- $P_k^{stand\text{-}by} = 4kW$
- $P_k^{ramp\text{-}up} = 8kW$
- $T_k^{ramp\text{-}up\text{-}off} = AvgProc$, where $AvgProc$ is the average of the processing times of all operations requiring R_k
- $T_k^{ramp\text{-}up\text{-}stand\text{-}by} = 0.5 \times T_k^{ramp\text{-}up\text{-}off}$

Table 2
Results of the parametric analysis for the evolutionary metaheuristic, showing the values tested and, in bold, 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

For the parameter tuning and preliminary analysis performed in Sections 6.3 and 6.4 we have selected a subset of 7 instances, in particular those indicated in Applegate and Cook (1991) to be the most difficult and challenging job shop instances (at least for makespan minimization) from the benchmark set introduced in Lawrence (1984): $LA21$, $LA24$, $LA25$, $LA27$, $LA29$, $LA38$ and $LA40$. We also consider instance $ABZ7$, known to be very challenging.

6.2. Performance measures

The comparison of multiobjective algorithms is not trivial and there is a lot of research and proposals about how to compare them. In our work, we can consider solutions as points in a 2-dimension space, as we are considering two objective functions. As it is well-known that a single indicator is not enough to compare several Pareto sets, we propose to use the hypervolume indicator (Zitzler and Thiele, 1998) (to be maximized) and the binary ϵ-indicator (Zitzler et al., 2003) (to be minimized). The hypervolume can be defined as the area of the set of points relative to a reference point, and in this work we take as reference point, for each instance, the worst values observed for each objective in any run multiplied by 1.05. On the other hand, the binary ϵ-indicator can be briefly described as the factor by which a set of points is worse than another with respect to all objectives. As the optimal Pareto set for each instance is not known, for calculating this indicator we use an approximation taking, for each instance, the non-dominated points considering all solutions from all runs of all methods. Both indicators are normalized and so they range between 0 and 1.

6.3. Parameter tuning and analysis of the evolutionary metaheuristic

In a preliminary parametric analysis we tested some values for the parameters of the evolutionary metaheuristic, in order to find a satisfactory configuration. Table 2 summarizes the tested values, indicating in bold the configuration that reached the best average results (regarding hypervolume and binary ϵ-indicator) in the selected subset of 8 instances (see Section 6.1).

We have experimentally seen that the proposed configuration achieves proper convergence patterns in reasonable running times. We will see along this study that in 10 min it reaches high quality solutions, and in hard instances these solutions can be further improved by using long runs of 3 h. As an example, we show in Fig. 9 two convergence patterns of single runs with 3 h time limit in $LA21$ (size 15×10) and $ABZ7$ (size 20×15) instances, where we see how the convergence in the largest instance is slower, as expected, as it is more difficult to reach high quality solutions due to the much larger search space.

Table 3 shows a comparison between the proposed configuration and alternative configurations built by taking the base configuration and modifying a single parameter. We can see that most of the time the proposed configuration achieves the best values for hypervolume and ϵ-indicator. Regarding the parameter *popSize*, it obviously has a large influence on the number of generations performed in the 10 min time limit, and the worst value for it seems to be 2000, which results in a smaller number of generations that are not enough for the algorithm to converge. Regarding the parameter *tSize*, it is interesting that in the largest instances (see $ABZ7$) it seems to be beneficial to choose high values in order to accelerate the convergence and reach good results in the considered time limit. It is well-known that in short executions it can be good to apply more selective pressure, however in long executions this might cause premature convergence. About parameter *crProb*, it is interesting that using value 0.6 also produces competitive results, maybe because of the ability to perform some more generations in the given time limit, although overall it seems to perform slightly worse than the value 1.0. Finally, about *mutProb*, it seems that too high mutation probabilities are not recommended, as the 0.3 configuration performs the worst between the four different values.

Then, we have also performed some experiments in order to assess the efficiency of the proposed energy-optimization heuristic and to prove that the local search alone and the plain genetic algorithm perform worse than the combination of both methods.

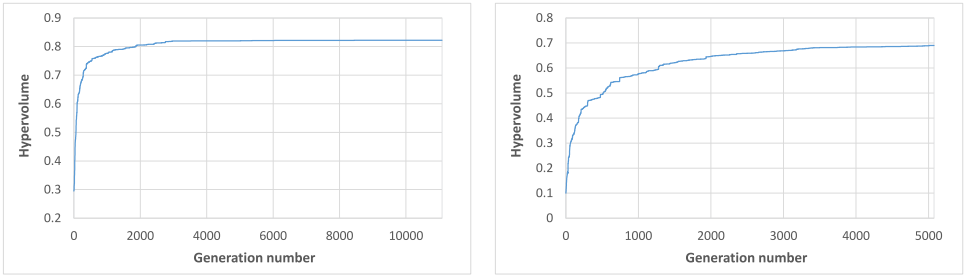
Table 4 shows the results of several variants of our memetic algorithm, all of them with a time limit of 10 min. For each method we show the average values of the hypervolume and ϵ-indicator for each instance, and also the average number of generations made in the 10 min time limit, in order to assess the differences in computational cost required by each configuration. We marked in bold the best values of each indicator obtained in each instance. The configurations tested were the following, where configuration (8) is our base configuration described in the rest of the paper.

- (1) No energy heuristic at all.
- (2) No energy heuristic during the execution, but first step and second step in recursive mode in the last generation.
- (3) Only first step of the energy heuristic.
- (4) Only first step of the energy heuristic, and also second step in recursive mode in the last generation.
- (5)[–] Only second step of the energy heuristic.
- (6) Only second step of the energy heuristic, and also applying it in recursive mode in the last generation.
- (7) Both first and second steps of the energy heuristic.
- (8) Both first and second steps of the energy heuristic, and also in recursive mode in the last generation.
- (9) Both first and second steps, and also in recursive mode during all the execution.
- (10) The same as (8), but without using local search
- (11) The same as (8), but without using the genetic algorithm (i.e. performing a number of local searches starting from random solutions)

Looking at Table 4, the best performing methods overall seem to be (4) and (8), as they always obtain the best values for both performance indicators (also (7), however it is always worse than or equal to (8)). Method (8) is our base configuration described in Section 3.2, i.e. perform both first and second steps, and perform the second step in recursive mode only in the last generation of the memetic algorithm, whereas method (4) omits the second step until the last generation, and (7) only omits the recursive mode in the last generation.

We have done some statistical tests to analyze differences between the different variants, in order to confirm the superiority of variants (4) and (8). As we have multiple-problem analysis, we used non-parametric statistical tests. First, we run a Shapiro–Wilk test to confirm the non-normality of the data. Then we used paired Wilcoxon signed rank tests to compare the hypervolume and the ϵ-indicator values between methods. In these tests, the level of confidence used was 95% and the alternative hypothesis were "the difference between hypervolume values is smaller than 0" or "the difference between ϵ-indicator values is larger than 0", i.e. the corresponding indicator is better.

The p- values obtained with these tests prove that (4) is the best performing method regarding hypervolume (the p- value against method (8) is 0.0002601, and against all other methods is even lower) and also regarding the binary ϵ-indicator (the p- value against method (8) is 0.006799, and against all other methods is much lower).



(a) LA21 (b) ABZ7

Fig. 9. Examples of convergence graphs of one run with 3 h time limit in $LA21$ and $ABZ7$ instances. The graphs show the hypervolume indicator with respect to generation number.

Table 3
Comparison of different parameter configurations for the proposed evolutionary metaheuristic with respect to the base configuration, using 10 min time limit. In bold, the best value for each indicator found in each instance.

Inst.	Base	*popSize*		*tSize*			*crProb*		*mutProb*		
		500	2000	4	8	16	0.6	0.8	0	0.1	0.3
Average hypervolume											
ABZ7	.741	.738	.606	.748	.764	**.780**	.757	.729	.706	.694	.682
LA21	**.811**	.752	.739	.751	.763	.751	.801	.762	.775	.761	.768
LA24	**.802**	.780	.776	.755	.772	.737	.798	.784	.740	.783	.779
LA25	**.911**	.881	.887	.902	.864	.846	.890	.890	.899	.896	.898
LA27	**.811**	.797	.691	.805	.794	.784	.804	.772	.777	.782	.760
LA29	.788	.748	.684	.767	.758	**.799**	.766	.772	.757	.733	.747
LA38	.780	.749	.743	.758	.750	.745	.763	.789	**.805**	.796	.748
LA40	**.848**	.828	.811	.837	.842	.815	.844	.845	.847	.837	.827
Average binary ϵ-indicator											
ABZ7	.153	.175	.274	.156	.149	.147	**.145**	.174	.197	.198	.208
LA21	**.112**	.172	.168	.157	.156	.171	.115	.163	.144	.147	.139
LA24	.151	.166	.145	.160	.171	.183	**.144**	.153	.173	.148	.154
LA25	**.064**	.081	.080	.068	.094	.092	.071	.073	.070	.069	.067
LA27	**.105**	.130	.227	.112	.123	.142	.120	.141	.139	.132	.157
LA29	**.147**	.173	.232	.181	.177	.149	.182	.164	.189	.188	.189
LA38	.127	.149	.163	.161	.148	.157	.140	.135	.120	**.118**	.147
LA40	.092	.105	.111	.096	.089	.108	.091	.096	**.085**	.093	.101
Average number of generations											
	524	1069	249	532	534	539	684	597	529	515	511

Table 4
Comparison of different variants of the proposed evolutionary metaheuristic, with 10 min time limit. In bold, the best value for each indicator found in each instance.

Inst.	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)
Average hypervolume											
ABZ7	.615	.789	.786	**.857**	.657	.661	.814	.814	.605	.783	.364
LA21	.689	.747	.844	**.866**	.756	.758	.854	.854	.792	.818	.430
LA24	.625	.789	.845	**.860**	.663	.668	.845	.845	.790	.743	.315
LA25	.697	.775	.879	.898	.769	.770	**.919**	**.919**	.870	.883	.446
LA27	.795	.856	.903	**.908**	.794	.794	.884	.884	.764	.840	.370
LA29	.618	.749	.887	**.898**	.640	.640	.817	.817	.614	.757	.288
LA38	.696	.845	.831	**.889**	.806	.809	.873	.873	.787	.866	.461
LA40	.738	.860	.839	**.914**	.817	.819	**.914**	**.914**	.829	.865	.414
Average binary ϵ-indicator											
ABZ7	.336	.171	.135	**.090**	.270	.264	.115	.115	.253	.140	.432
LA21	.276	.225	.114	.094	.183	.179	**.090**	**.090**	.139	.118	.392
LA24	.352	.182	.119	**.108**	.311	.305	.120	.120	.144	.162	.557
LA25	.276	.202	.083	.070	.192	.192	**.051**	**.051**	.079	.086	.395
LA27	.180	.124	.056	**.055**	.175	.175	.074	.074	.165	.100	.493
LA29	.362	.232	.103	**.096**	.328	.327	.155	.155	.293	.168	.543
LA38	.267	.118	.135	**.081**	.145	.141	.083	.083	.130	.091	.355
LA40	.219	.096	.121	.052	.130	.129	**.048**	**.048**	.103	.101	.420
Average number of generations											
	1849	1843	1712	1703	607	606	525	524	125	6434	380550

It can also be proven that (8) is the second best method regarding hypervolume (the p- value against method (3) is 0.006357, against (7) is 0.0001575 and against all other methods, except (4), is even lower) and also regarding the binary ϵ-indicator, tied with method (7) (the p- value against method (7) is 0.1855, against (3) is 0.0006342 and against all other methods, except (4), is even lower).

When we compare columns (8), (10) and (11) we can conclude that the hybridization of genetic algorithm and local search produces much better results than each method alone. This is true even when (10) is able to perform many more generations than (8) in the same time limit. It is remarkable that the number of local searches performed by (11), which is in average 380550, is lower than the number performed by (8), which is in average 524000 (as it is run an average of 524 generations and the population size is 1000). This is due to the fact that local searches starting from random solutions take many more steps to reach a local optimum than when starting from good solutions.

Approach (1), i.e. never using the energy-optimization heuristic, performed poorly, even if the algorithm was able to perform many more generations, and approach (9), i.e. using the recursive mode through all the execution, also produced bad results, as in this last case the memetic algorithm was able to perform very few total generations because the computational burden was too high, and so it had not enough time to converge.

The results also indicate that the first step is more important than the second step, and also less computationally expensive. Regarding the application of the second step in recursive mode only in the last generation, we can see that configuration (2) is better than (1), configuration (4) is better than (3), configuration (6) is better than (5), and configuration (8) is better than (7) although in these last two cases the improvement is certainly minimal (we could see there are in fact improvements if we show more significant digits in Table 4). This proves that this is an efficient compromise, as the increase in running time is negligible but the final results do sometimes improve, even if the improvement is minor when using the second step during all the execution. As an example, when comparing (7) to (8) in one of the runs of the instance ABZ7, by applying the recursive mode we were able to slightly improve the energy consumption in 3 of the 17 solutions of the obtained Pareto set.

However, we argue that configuration (8) might have better potential than (4) for reaching better results in long executions. Additionally, we have seen that always applying the recursive mode (configuration (9)) had not enough time to converge, so it might obtain great results in very long executions. Hence, we performed another round of experiments with a 3-hour time limit, comparing configurations (4), (8) and (9). The results are reported in Table 5, where we show for each method the average and best values of the hypervolume and ϵ-indicator for each instance, and the average number of generations made in the 3 h time limit.

We see that in long executions, configuration (8) is clearly better than the others, and so comparing it to (4) and (9) we conclude that the proposed way of applying the energy-optimization heuristic is better than the other possibilities. This can be confirmed with statistical tests, as in this case, regarding hypervolume, the p- value against method (4) is 0.0003702 and against (9) is 0.0000002346, and also regarding the binary ϵ-indicator, where the p- value against method (4) is 0.000002417 and against (9) is 0.00002364.

In conclusion, in very short runs the best method is (4) (i.e. not using the second step of the energy heuristic until the last generation) whereas in long runs the best method is (8) (using that second step during all the execution). We argue that in most real applications of this scheduling problem the time limit will not be an issue, and so we choose method (8) as the winner and we will use it to compare with other algorithms and with the state of the art. However we suggest that variant (4) should be used in any real application that imposes a very short time limit.

6.4. Preliminary analysis on the CP approach

In Section 4.1 we have presented a CP model that implements a lexicographic approach for the minimization of the objective pair *WEC* and C_{max} with the energy *WEC* as primary objective, and that exploited the energy constraint propagation rule (constraint (4m), in the model).

Table 5
Comparison of different variants of the utilization of the energy-optimization heuristic, with 3 h time limit. In bold, the best value of each indicator found in each instance.

Inst.	(4)	(8)	(9)
Hypervolume: Avg. (Best)			
ABZ7	.624 (.649)	**.691 (.764)**	.552 (.580)
LA21	.798 (.869)	**.821 (.871)**	.800 (.867)
LA24	.686 (**.888**)	**.705** (.746)	.678 (.723)
LA25	.799 (.843)	**.839 (.879)**	.831 (.872)
LA27	.776 (.848)	**.794 (.870)**	.756 (.809)
LA29	**.811** (.898)	.738 (**.858**)	.667 (.785)
LA38	.694 (.745)	**.732 (.799)**	.701 (.796)
LA40	.823 (.859)	**.860 (.889)**	.835 (.853)
Binary ϵ-indicator: Avg. (Best)			
ABZ7	.246 (.182)	**.170 (.120)**	.297 (.246)
LA21	.141 (.073)	**.108 (.050)**	.128 (.073)
LA24	.263 (**.029**)	**.227** (.166)	.247 (.166)
LA25	.108 (.079)	.079 (**.026**)	**.074** (.034)
LA27	**.137 (.112)**	**.137** (.113)	.146 (.122)
LA29	**.167 (.080)**	.209 (.087)	.259 (.198)
LA38	.193 (.131)	**.136 (.087)**	.169 (**.087**)
LA40	.115 (.101)	**.070 (.055)**	.078 (.060)
Average number of generations			
	31095	9227	2340

Table 6
Comparison among different variants of the CP approach with 3 h time limit, used in the preliminary analysis. In bold, the best value of each indicator found in each instance.

Inst.	CP MK-WEC	CP-Prop MK-WEC	CP WEC-MK	CP-Prop WEC-MK
Hypervolume: Avg. (Best)				
ABZ7	**.170 (.198)**	.000 (.000)	.114 (.119)	.000 (.000)
LA21	**.394 (.523)**	.224 (.354)	.237 (.246)	.329 (.509)
LA24	**.512 (.546)**	.280 (.456)	.096 (.143)	.365 (.388)
LA25	**.523 (.571)**	.251 (.444)	.111 (.179)	.433 (.548)
LA27	**.263 (.342)**	.000 (.000)	.054 (.054)	.000 (.000)
LA29	**.371 (.453)**	.000 (.000)	.087 (.134)	.000 (.000)
LA38	**.174 (.224)**	.000 (.000)	.054 (.058)	.000 (.000)
LA40	**.200 (.259)**	.000 (.000)	.085 (.088)	.000 (.000)
Binary ϵ-indicator: Avg. (Best)				
ABZ7	**.830 (.802)**	– (–)	.886 (.881)	– (–)
LA21	**.605 (.476)**	.776 (.645)	.757 (.748)	.670 (.489)
LA24	**.488 (.453)**	.719 (.544)	.902 (.854)	.635 (.612)
LA25	**.476 (.428)**	.748 (.554)	.887 (.817)	.567 (.452)
LA27	**.736 (.658)**	– (–)	.946 (.946)	– (–)
LA29	**.628 (.547)**	– (–)	.912 (.866)	– (–)
LA38	**.826 (.776)**	– (–)	.946 (.942)	– (–)
LA40	**.800 (.741)**	– (–)	.915 (.912)	– (–)

In the current analysis however, given the extended set of benchmark instances to be solved compared to our previous work, as well as the significant size of many of those instances, we found advisable to test slight variations of the model. For instance, we tested the inversion between the primary (*WEC*) and the secondary (C_{max}) key, as well as the CP model deprived of the (4m) constraint.

This preliminary experimental analysis therefore leads to four possible cases, whose results in terms of hypervolume and ϵ-indicator average measurements are shown in

Table 6. In particular, the *CP-PROP WEC-MK* results correspond to the application of the full model presented in Section 4.1, with WEC as primary key and C_{max} as secondary key; the *CP WEC-MK* results correspond to the application of the model presented in Section 4.1 without the (4m) constraint, with WEC as primary key and C_{max} as secondary key; the *CP-PROP MK-WEC* results correspond to the application of the full model presented in Section 4.1, with C_{max} as primary key and WEC as secondary key; finally, the *CP MK-WEC* results correspond to the application of the model presented in Section 4.1 without the (4m) constraint, with C_{max} as primary key and WEC as secondary key.

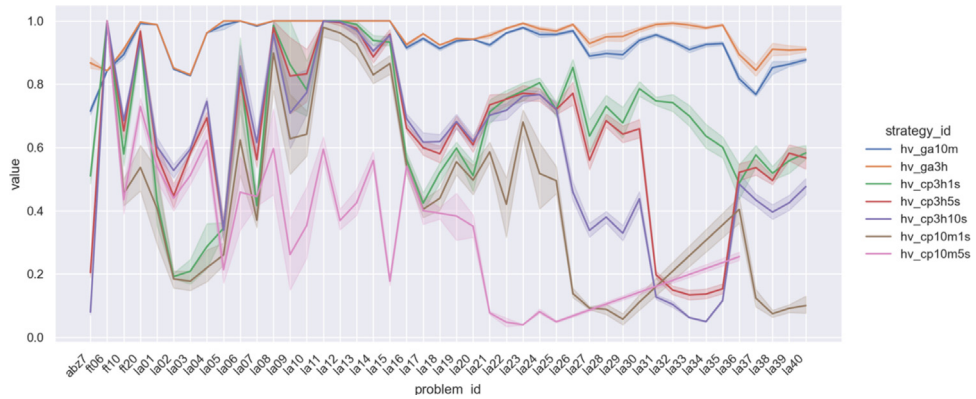


Fig. 10. Hypervolumes: CP Vs. GA algorithms.

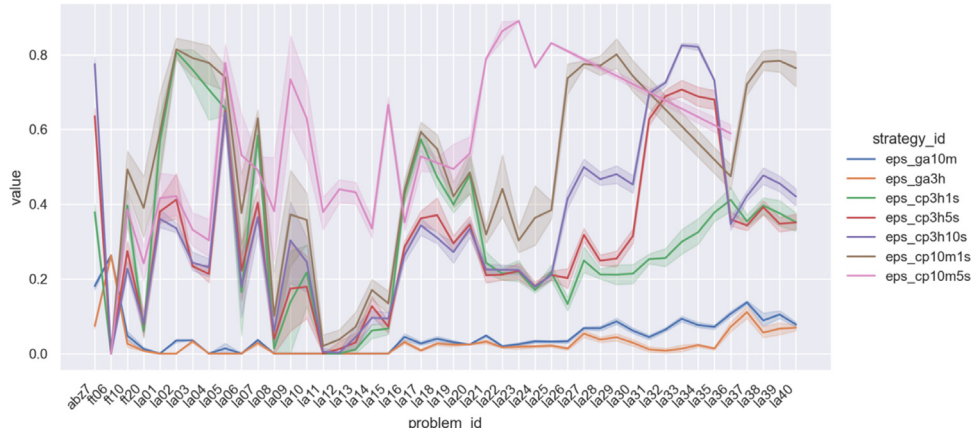


Fig. 11. ϵ-Indicator: CP Vs. GA algorithms.

All tests are performed within maximum CPU times of 3 hours divided in 10 solving runs. As known, the lexicographic multi-objective optimization procedure of CPLEX returns the best solution found at the end of each run (candidate solutions), out of which the Pareto set is built, by considering the non-dominated solutions. Moreover, each solution is computed by optimizing the primary objective first, and trying to optimize the secondary objective during the time between one improvement of the primary objective and the next. However, this approach lends itself to disregarding some good solutions possibly discovered during the optimization process, because as soon as the primary objective is improved, the value of secondary objective of the previous solution is lost, and its optimization must start anew. In this paper however, we use a slightly different method to save the candidate solutions, by considering *all* the solutions computed during the optimization process,[3] and not only the single solution returned at the end of each individual run. Finally, the Pareto set is computed filtering the non-dominated solutions out of this set.

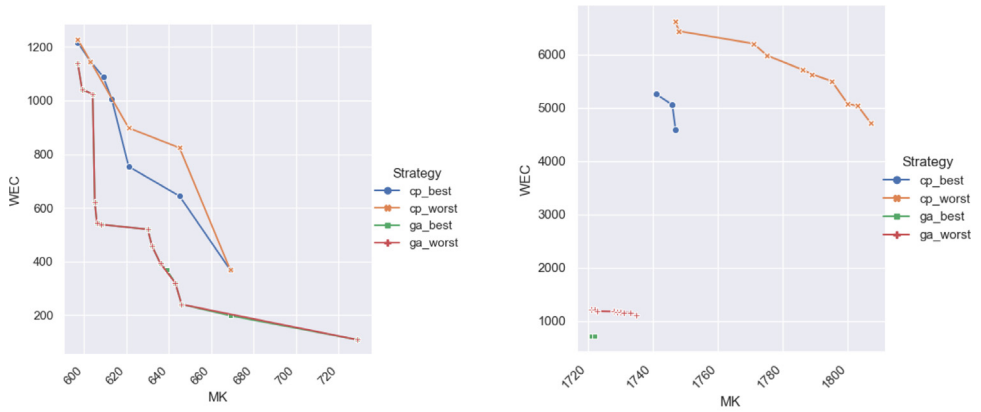
Table 6 clearly shows that the most performing configuration across the range of the selected benchmark set is the *CP MK-WEC*, followed by the *CP-PROP MK-WEC* configuration, which however seems to be usable only over the benchmark's smallest instances (i.e., from $LA21$, $LA24$ and $LA25$). We conjecture that this behavior is mainly due to computational burden imposed by the propagation rule, which is heavy on the largest instances and precludes the synthesis of good solutions in the available time. The *CP WEC-MEC* configuration solves instead all the instances though with poor performance. Statistical tests also prove that the differences between *CP MK-WEC* and the other three methods are statistically significant in both hypervolume and ϵ-indicator. Given the results obtained from this preliminary analysis, in the comparison that follows we will select the *CP MK-WEC* configuration.

6.5. Results and comparison with the state of the art

In our experiments we consider two types of executions: short runs with a time limit of about 10 min, and long runs with a time limit of about 3 h. In this way, we can better see the convergence of our methods, and conclude if there is some better method depending on the available computational time.

Figs. 10 and 11 show a graphical comparison among the hypervolume and ϵ-indicator values produced by the genetic algorithm (GA) and the constraint programming algorithm (CP), with total CPU times of either 10 min or 3 h. In the CP case, we consider a further subdivision based on the number of random restarts (steps) performed within the CPU time allotted for each run; in particular, we uniformly distribute the total CPU time over 1, 5 or 10 steps when the total CPU time is 3 h, and over 1 or 5 steps when the total CPU time is 10 min (we have experienced that 1-minute runs do not return any solution). As a first result, we observe that the proposed genetic approach shows the best performance over the full benchmark set, with the only exception of the smallest $FT06$ instance. In particular, we note that 10 min of computation are enough for the GA to outperform over the set of CP results; quite naturally, the 3 h GA results produce a further improvement over the respective 10 min results, though this improvement is particularly evident only for the bigger instances ranging from $LA26$ to $LA40$. The rather similar performance exhibited by both the 10 min

[3] In fact, IBM-ILOG CPLEX maintains an *engine log* of the whole optimization process, containing the history of all the produced solution values.



(a) LA03 (b) LA34

Fig. 12. GA Vs. CP: LA03 and LA34 results.

and 3 h version of the GA approach proves that the GA has very good convergence speed towards the optimum.

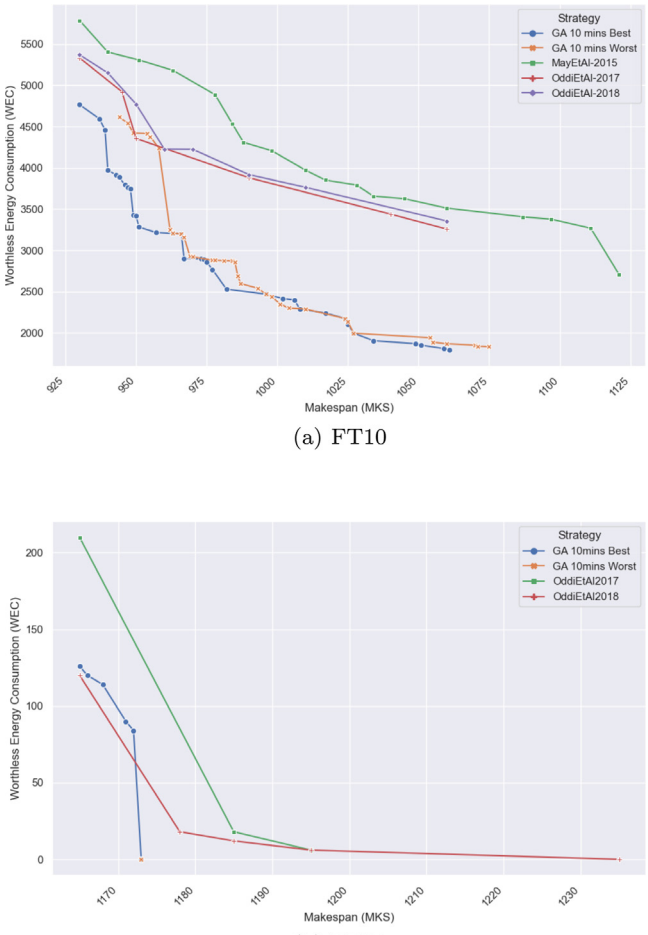
Similarly as in Section 6.3, we have confirmed these results with statistical tests. Regarding hypervolume, the GA using 3 h run time is the best method (p- value against GA using 10 min is 4.171e−07, whereas against all CP methods is even lower), and also regarding the binary ϵ-indicator (p- value against GA using 10 min is 6.169e−07 and against all CP methods is even lower). It can also be confirmed that GA using 10 min is significantly better than all CP methods regardless of its running time (all p- values are below 5.004e−08).

As the figures show, the CP approach is also able to obtain high quality results but, as opposite to the GA case, there is substantial difference between the 10 min and the 3 h results. These differences are statistically significant. We conjecture that this is due to the fact that the time necessary for the CP approach to find a first solution is considerably higher than the time necessary for the GA approach (which is immediate), and hence the CP's optimization phase starts much later, ultimately exhibiting a much lower convergence speed than GA. This is also indirectly confirmed by the observation that when the total imposed CPU time on the CP approach is 10 min, the best performance on the majority of instances (i.e., the larger ones) is obtained by the 1 step version of the procedure (i.e., 10 min per run) rather than by the 5 step version (i.e., 2 min per run). Indeed, it is evident that 2 min is often not enough to both find an initial solution and perform an effective optimization phase.

Statistical tests confirm that when using CP with 10 min time limit, using 1 step is better than using 5 steps (p- value of 0.0002754 regarding hypervolume and p- value 0.0009918 regarding the binary ϵ-indicator).

Fig. 12 shows one example of the Pareto sets for the $LA03$ (a) and the $LA34$ (b) instances (whose hypervolume and ϵ-Indicator values are respectively charted in Fig. 10 and Fig. 11), obtained with the GA and the CP approach. As readily observable, the GA approach outperforms the CP in both cases, though the difference in performance is not as large in case of smaller instances ($LA03$).

Similarly to what we did with Fig. 12, in Fig. 13 we present an example of the differences between the Pareto sets obtained with our new methods and the Pareto sets in previous state of the art, limited to the two instances $FT10$ and $FT20$, given that the $FT06$ instance has in fact a Pareto set composed of a single point $(55, 124)$ that represents its optimal solution. In particular, we compare the results obtained by the genetic algorithm proposed in May et al. (2015) and the constraint programming approaches of Oddi et al. (2017, 2018) (see Section 1) against those obtained with the 10 minute version of our current GA procedure, to keep the comparison fair. Note that both the *Best* and the *Worst* Pareto sets obtained by the GA are shown in the plots. As the figure shows, the GA approach outperforms all the other methods



(a) FT10

(b) FT20

Fig. 13. Comparison with previous state-of-the-art.

even in the worst of the ten runs performed; the GA's superiority is particularly clear in the $FT10$ instance.

Regarding the FT20 instance, we remark that in May et al. (2015) it is only tackled using single-objective optimization, either makespan or WEC: when minimizing w.r.t. the makespan they report a solution with makespan 1242 and WEC 1584, whereas when minimizing w.r.t. the WEC they report a solution with makespan 1630 and zero WEC. The authors do not perform multi-objective optimization in this instance,

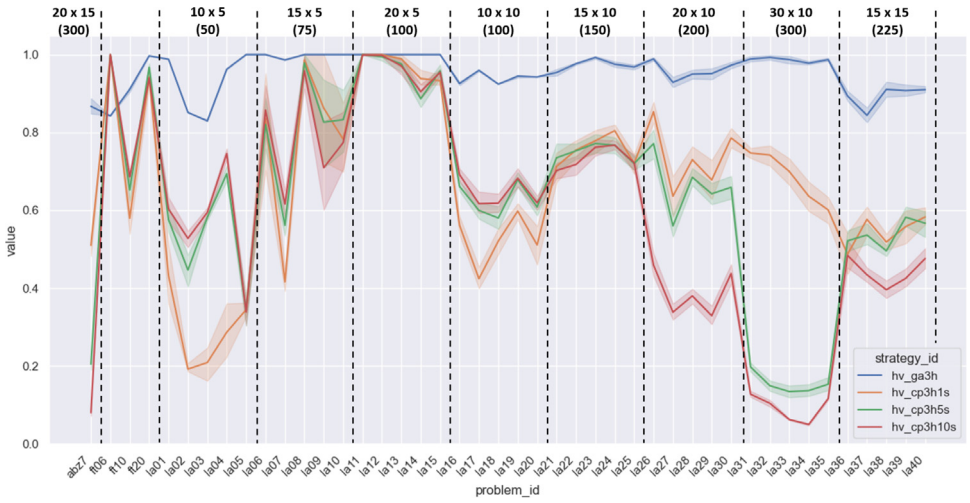


Fig. 14. Hypervolumes: CP Vs. GA algorithms, limited to the 3 h CPU time case, with different number of random restarts. On top of the figure the size of each instance batch is shown in terms of ($Jobs \times machines$) with the total number of involved activities between brackets.

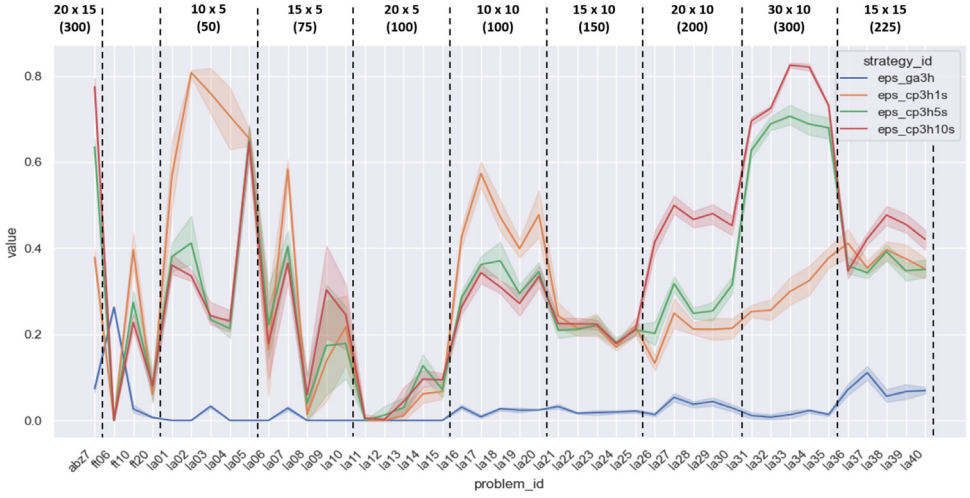


Fig. 15. ϵ-Indicator: CP Vs. GA algorithms, limited to the 3 h CPU time case, with different number of random restarts. On top of the figure the size of each instance batch is shown in terms of ($Jobs \times machines$) with the total number of involved activities between brackets.

arguing that preliminary tests show that room for improvement is negligible. However, our hybrid evolutionary metaheuristic was able to obtain a solution with $makespan = 1173$ and $WEC = 0$ in all 10 runs. Moreover, further interesting solutions were found; for example, our evolutionary metaheuristic also reached a solution with $makespan = 1165$ (i.e. the optimal makespan) and $WEC = 126$, in 8 of the 10 runs. Hence, our results in the FT20 instance look vastly superior than those reported in May et al. (2015), and demonstrate our capability to minimize over the WEC objective, capability which is confirmed on all the other instances of this study.

We will now compare the results obtained from the GA and CP approaches (see Figs. 14 and 15), focusing on the most CPU-time intensive versions that we have used (3 hours). On the top of both figures, the size of each instance batch is shown in terms of the ($Jobs \times machines$) indicator, while the total number of involved activities is shown between brackets. In these figures, the mean hypervolume and ϵ-indicator values are respectively plotted, together with the 95% confidence interval, around the mean.

As previously described, the CP approach was run in three different flavors: in the first ($hv\_cp3h10s$), the 3 hours CPU time is equally divided in 10 different steps (thus favoring exploration over exploitation), while in the third (hv_cp3h1s), all the CPU time is used for one step only (thus relying on a purely exploitation approach). The second method (hv_cp3h5s) places itself halfway between the previous two, in that the CPU time is divided in 5 different steps. Beyond the fact that the GA approach is the best performing method, which was already clear in Fig. 10, the following further observations can be made.

The results are interesting as they show how the different ratio between exploration and exploration affects the algorithm's performance depending on the given instance's complexity (i.e., its size). To begin with, we observe that the hv_cp3h1s approach performs particularly well on the bigger instances, that is, from $LA26$ to $LA40$, including $ABZ7$, while the hv_cp3h5s and $hv\_cp3h10s$ approaches are generally less effective. We also observe that the $hv\_cp3h10s$ method exhibits particularly poor performance against the bigger instances, the reason being that the exploration granted by the approach is not sufficient if it is not coupled with an intensive exploitation phase. In this respect, the results show that the hv_cp3h5s approach do not provide sufficient exploitation power to effectively solve the biggest benchmark instances, i.e., those ranging from $LA31$ to $LA35$ (300 activities). Conversely, the $hv\_cp3h10s$ and the hv_cp3h5s approaches do perform better than hv_cp3h1s on the smaller instances, providing an example of how a wise mixture of exploitation and exploration can be very effective when the instances' complexity allows. Regarding statistical tests to confirm these results about the CP approach, it can be proven that in the overall benchmark there are no statistical differences between using 1 step or

Table 7
MILP and GA: Pareto sets comparison.

Inst.	MILP	HV	ϵ-I	GA	HV	ϵ-I
FT06	(**55***,**124***)	1.00	.000	(56,124), (55*,134)	.842	.263
LA01	(677,390), (666*,408)	.253	.714	(677,294), (666*,354)	.891	.000
LA02	(671,544), (669,760), (655*,898)	.379	.577	(747,0), (734,102), (731,150), (715,156), (707,204), (703,228), (678,350), (677,398), (674,478), (672,484), (671,520), (669,556), (665,618), (663,850), (660,868), (657,886), (655*,898)	.744	.000
LA03	(646,368), (606,544), (605,622), (604,1024), (599,1042), (**597***,**1108***)	.691	.239	(729,108), (646,240), (643,320), (636,392), (632,458), (630,520), (608,538), (606,544), (605,622), (604,1024), (599,1042), (597*,1138)	.813	.039
LA04	(607,368), (600,632), (596,776), (595,800), (590*,818)	.771	.175	(611,368), (607,386), (603,452), (600,464), (598,506), (596,782), (595,818), (590*,848)	.827	.057
LA05	(593*,240)	1.00	.000	(593*,240)	1.00	.000

5 steps, but the differences between these two versions and the version using 10 steps is in fact statistically significant, even if the version using 10 steps can perform better in some groups of instances.

Finally, we provide in Table 7 some results obtained with the MILP procedure described in Section 5, and compare them with the best results obtained with the GA. For each instance, the Table's columns report the best Pareto set found by both solving methods (for each solution we show its makespan and its WEC), together with the corresponding hypervolume value (HV column) and ϵ-indicator value (ϵ-I column) of the Pareto set.

The number of instances used for this comparison is limited because the complexity of the MILP model did not allow to solve any instance of size greater than the $LA05$, within the given CPU time limit. Given the small size of the instances, the Pareto fronts computed during the GA's 10 runs allowed for each instance are the same or extremely similar; therefore, it is sufficient to report only one of them in the table.

Despite the limitation on the number of instances, some interesting conclusions can however be drawn. First of all, the MILP model allowed to guarantee the optimality of some solutions found (represented in bold with asterisks, in the table), see for instance the instance $FT06$ and the instance $LA03$. These are solutions that the GA procedure (the best performing approach in our work) could not reach. Secondly, we would like to highlight that the overall quality of the solutions found by the MILP is rather high, see in particular the values for the $LA04$ instance, where the underlined pairs represent solutions that remained unparalleled by the GA. Accidentally however, the reader may notice that the GA's hypervolume and ϵ-indicator values are better than the MILP's (0.771 Vs. 0.827 and 0.175 Vs. 0.057); this is partly due to the fact that the GA's Pareto set contains a higher number of solutions.

The overall lesson we can learn by observing the MILP performance is that despite the solutions returned by the GA are very good, there may be still significant room for further improvement on the overall benchmark. The detailed schedules of the solutions found by our hybrid evolutionary algorithm for all instances considered in this experimental study are openly available on the web.[4]

[4] Repository section in http://di002.edv.uniovi.es/iscop

7. Conclusions

In this paper we have considered the job shop scheduling problem, minimizing both the makespan and the energy consumption. The considered energy model allows each machine to be in several states: *Off*, *Stand-by*, *Idle* or *Processing*. Each state consumes a different amount of energy, and some transitions between states consume some time and energy. This is a very relevant problem in real scenarios, but it is very complex. To solve it we propose different techniques: a hybrid evolutionary algorithm that combines a NSGA-II based genetic algorithm with local search and energy optimization heuristics, a constraint-programming (CP) approach and a mixed-integer linear programming (MILP) approach. Our methods are analyzed and compared with the state-of-the-art algorithms, obtaining competitive results.

The results of the experimental study show that the genetic algorithm is the best performing approach, compared to both the CP and the MILP, mainly because of its ability to find the best balance between the diversification and the exploration aspects of the optimization process, and because of the efficacy of the energy optimization heuristic. Another important aspect relative to the methods compared in this work are the conclusions we can draw about their convergence speed. In fact, the analysis of the results obtained in the 10 minutes and the 3 hours versions of the GA and CP approaches clearly shows that the GA generally converges to very good solutions much earlier than the CP. Given sufficient time however, the latter succeeds in achieving comparable results, even though the CP's solution quality generally remains behind. Regarding the energy optimization heuristic used in the GA, we have seen that when the time limit is very reduced it is better to omit its second step until the last generation.

Relatively to the future work, the most interesting research direction is to consider energy models with more characteristics present in real environments. One possibility is to explicitly consider an additional "Setup" state for machines, representing the energy and time needed to readjust it between the execution of two consecutive operations. The energy model would be more realistic than that considered in this paper in the cases where setup times are sequence-dependent, which happens in some real environments, as painting and printing industries or some manufacturing environments (Wilbrecht and Prescott, 1969). It would also be interesting to allow different processing modes in machines, i.e. having the possibility to consume extra energy in order to reduce processing time (Zhang and Chiong, 2016), or even shifting energy costs (Grimes et al., 2014).

Another possibility is to tackle variants of the job shop, as for example flexibility in machine selection, so each operation can be executed by different machines, possibly with different duration and energy consumption. In fact, the flexible job shop scheduling problem is possibly the most studied job shop variant in the literature. Parallel machines are also common in scheduling literature. In these variants, the utilization of Grouping Genetic Algorithms (Ramos-Figueroa et al., 2021) could be interesting.

The use of automated planning in scheduling problems is a recent and promising research direction. For example, in Parkinson et al. (2017) the authors minimize energy consumption and machine error considering an energy model where machines can be in different states, each consuming different energy. They use automatic planning to vary the energy consumed in between the processing intervals by machine tools. The machine error measure minimized in the paper is also an objective function worthy of further study, as it is relevant in some real environments.

The evolutionary algorithm proposed in this paper can also be improved, for example by designing a local search neighborhood focused on energy reduction. Moreover, the removal of duplicated-fitness individuals described in Section 3.4 can possibly be improved, by keeping a larger number of them and/or by choosing them using some similarity metric instead of randomly. More research is needed in this topic. Finally, MOEA/D (Zhang and Li, 2007) is a multi-objective evolutionary algorithm which is based on decomposing a multi-objective problem into several single-objective subproblems. It could be interesting to design it and perform a comparison with the proposal of this paper, to assess whether a dominance-based metaheuristic or a decomposition-based metaheuristic is better suited to our particular problem.

CRediT authorship contribution statement

Miguel A. González: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Writing – original draft, Writing – review & editing. **Riccardo Rasconi:** Conceptualization, Methodology, Software, Formal analysis, Investigation, Writing – original draft. **Angelo Oddi:** Conceptualization, Methodology, Writing – original draft, Supervision.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research has been supported by the Spanish Government under project PID2019-106263RB-I00. ISTC-CNR authors were supported by the European Space Agency Contract No. 4000112300/14/D/MRP "Mars Express Data Planning Tool MEXAR2 Maintenance".

References

Adams, J., Balas, E., Zawack, D., 1988. The shifting bottleneck procedure for job shop scheduling. Manag. Sci. 34, 391–401.
Applegate, D., Cook, W., 1991. A computational study of the job-shop scheduling problem. ORSA J. Comput. 3, 149–156.
Apt, K., 2003. Principles of Constraint Programming. Cambridge University Press, New York, NY, USA.
Baker, K., 1974. Introduction to Sequencing and Scheduling. Wiley.
Beck, J.C., Feng, T., Watson, J.-P., 2011. Combining constraint programming and local search for job-shop scheduling. Inf. J. Comput. 23, 1–14.
Bierwirth, C., 1995. A generalized permutation approach to jobshop scheduling with genetic algorithms. OR Spectrum 17, 87–92.
Brandimarte, P., Maiocco, M., 1999. Job shop scheduling with a non-regular objective: a comparison of neighbourhood structures based on a sequencing/timing decomposition. Int. J. Prod. Res. 37 (8), 1697–1715.
Dai, M., Tang, D., Giret, A., Salido, M.A., Li, W., 2013. Energy-efficient scheduling for a flexible flow shop using an improved genetic-simulated annealing algorithm. Robot. Comput.-Integr. Manuf. 29, 418–429.
Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., 2002. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. 6 (2), 182–197.
Escamilla, J., Salido, M.A., 2018. A dual scheduling model for optimizing robustness and energy consumption in manufacturing systems. Proc. Inst. Mech. Eng. B 232 (1), 5–16.
Faria, G., Vieira, S., Branco, P.J.C., 2019. Evolutionary process scheduling approach for energy cost minimization in a yeast production factory: design, simulation, and factory implementation. Energy Syst. 10 (1), 113–139.
Fisher, H., Thomson, G.L., 1963. Probabilistic learning combinations of local job-shop scheduling rules. In: Muth, J.F., Thomson, G.L. (Eds.), Industrial Scheduling. Prentice Hall, pp. 225–251.
Fourer, R., Gay, D.M., Kernighan, B.W., 2003. AMPL: A Modeling Language for Mathematical Programming. Duxbury-Thomson.
Gao, K., Huang, Y., Sadollah, A., Wang, L., 2019. A review of energy-efficient scheduling in intelligent production systems. Complex Intell. Syst. 6, 237–249.
González, M.A., Oddi, A., Rasconi, R., 2017. 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, pp. 140–148.
González, M.A., Oddi, A., Rasconi, R., 2019. Efficient approaches for solving a multiobjective energy-aware job shop scheduling problem. Fund. Inform. 167 (1–2), 93–132.
Grimes, D., Ifrim, G., O'Sullivan, B., Simonis, H., 2014. Analyzing the impact of electricity price forecasting on energy cost-aware scheduling. Sustain. Comput. Inf. Syst. 4 (4), 276–291.
He, L., Chiong, R., Li, W., Dhakal, S., Cao, Y., Zhang, Y., 2022. Multiobjective optimization of energy-efficient JOB-shop scheduling with dynamic reference point-based fuzzy relative entropy. IEEE Trans. Ind. Inform. 18 (1), 600–610.
He, L., Li, W., Zhang, Y., Cao, Y., 2019. A discrete multi-objective fireworks algorithm for flowshop scheduling with sequence-dependent setup times. Swarm Evol. Comput. 51, 100575.
Jiang, X., Tian, Z., Liu, W., Suo, Y., Chen, K., Xu, X., Li, Z., 2022. Energy-efficient scheduling of flexible job shops with complex processes: A case study for the aerospace industry complex components in China. J. Ind. Inf. Integr. 27:100293.
Jiang, T., Zhang, C., Zhu, H., Gu, J., Deng, G., 2018. Energy-efficient scheduling for a job shop using an improved whale optimization algorithm. Mathematics 6 (11), 220.
Laborie, P., 2003. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. Artificial Intelligence 143 (2), 151–188.
Lawrence, S., 1984. Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement). Tech. rep., Graduate School of Industrial Administration, Carnegie Mellon University.
Le Pape, C., Baptiste, P., Nuijten, W., 2001. Constraint-Based Scheduling: Applying Constraint Programming To Scheduling Problems. Springer Science+Business Media, New York, NY, USA.
Li, M., Lei, D., 2021. An imperialist competitive algorithm with feedback for energy-efficient flexible job shop scheduling with transportation and sequence-dependent setup times. Eng. Appl. Artif. Intell. 103:104307.
Li, M., Wang, G.-G., 2022. A review of green shop scheduling problem. Inform. Sci. 589, 478–496.
Liu, Y., Dong, H., Lohse, N., Petrovic, S., Gindy, N., 2014. An investigation into minimising total energy consumption and total weighted tardiness in job shops. J. Clean. Prod. 65, 87–96.
May, G., Stahl, B., Taisch, M., Prabhu, V., 2015. Multi-objective genetic algorithm for energy-efficient job shop scheduling. Int. J. Prod. Res. 53 (23), 7071–7089.
Miettinen, K., 2012. Nonlinear Multiobjective Optimization. In: International Series in Operations Research & Management Science, Springer US.
Mouzon, G., Yildirim, M.B., Twomey, J., 2007. Operational methods for minimization of energy consumption of manufacturing equipment. Int. J. Prod. Res. 45 (18–19), 4247–4271.
Nowicki, E., Smutnicki, C., 2005. An advanced tabu search algorithm for the job shop problem. J. Sched. 8 (2), 145–159.
Oddi, A., Rasconi, R., González, M.A., 2017. A constraint programming approach for the energy-efficient job shop scheduling problem. In: Proceedings of the 8th Multidisciplinary International Scheduling Conference: Theory & Applications (MISTA-2017). pp. 158–172.
Oddi, A., Rasconi, R., González, M.A., 2018. Energy-aware multiple state machine scheduling for multiobjective optimization. In: Ghidini, C., Magnini, B., Passerini, A., Traverso, P. (Eds.), AI*IA 2018 – Advances in Artificial Intelligence. Springer International Publishing, Cham, pp. 474–486.
Parkinson, S., Longstaff, A., Fletcher, S., Vallati, M., 2017. 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, pp. 400–408.

Ramos-Figueroa, O., Quiroz-Castellanos, M., Mezura-Montes, E., Kharel, R., 2021. Variation operators for grouping genetic algorithms: A review. Swarm Evol. Comput. 60:100796.

Van Laarhoven, P., Aarts, E., Lenstra, K., 1992. Job shop scheduling by simulated annealing. Oper. Res. 40, 113–125.

Vilím, P., Barták, R., Cepek, O., 2004. 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. pp. 62–76.

Wilbrecht, J., Prescott, W., 1969. The influence of setup time on job shop performance. Manage. Sci. 16 (4), 391–401.

Zhang, R., Chiong, R., 2016. Solving the energy-efficient job shop scheduling problem: a multi-objective genetic algorithm with enhanced local search for minimizing the total weighted tardiness and total energy consumption. J. Clean. Prod. 112, 3361–3375.

Zhang, Q., Li, H., 2007. MOEA/D: a multiobjective evolutionary algorithm based on decomposition. IEEE Trans. Evol. Comput. 11 (6), 712–731.

Zhang, Z., Wu, L., Peng, T., Jia, S., 2019. An improved scheduling approach for minimizing total energy consumption and makespan in a flexible job shop environment. Sustainability 11 (1), 179.

Zhu, G.-Y., He, L.-J., Ju, X.-W., Zhang, W.-B., 2018. A fitness assignment strategy based on the grey and entropy parallel analysis and its application to MOEA. European J. Oper. Res. 265 (3), 813–828.

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

Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., da Fonseca, V.G., 2003. Performance assessment of multiobjective optimizers: An analysis and review. IEEE Trans. Evol. Comput. 7 (2), 117–132.