Natural Computing (2025) 24:483–496
https://doi.org/10.1007/s11047-025-10016-x

A hybrid evolutionary approach for lexicographic green flexible jobshop with interval uncertainty

Sezin Afşar[1] · **Jorge Puente**[1] · **Juan José Palacios**[1] · **Inés González-Rodríguez**[2] · **Camino R. Vela**[1]

Accepted: 8 April 2025 / Published online: 19 May 2025

Abstract
This article addresses the flexible job shop problem with uncertain processing times modelled by intervals. Due to climate change and the need for energy efficiency, there is an increasing interest in sustainability in addition to traditional production-related objectives such as makespan. In this work, we tackle a lexicographical goal programming scenario minimising makespan firstly and total energy consumption lately. We propose a hybrid evolutionary algorithm based on a genetic algorithm, incorporating heuristic seeding and a post-processing step using constraint programming. The experimental study shows that the proposed approach is able to meet tighter makespan goals than previously published methods, while offering a 32% improvement in energy consumption when goals are met.

Keywords Flexible job shop scheduling · Intervals · Heuristic seeding · Constraint programming · Lexicographic goal programming

1 Introduction

Scheduling problems concentrate on the allocation of limited resources in various tasks while respecting specific constraints Pinedo (2022). One of the best-known applications of scheduling is in manufacturing, for example, in the weapon industry Chen et al. (2012), the automotive parts industry Alvarez-Valdes et al. (2005), healthcare Xiang (2017) or steel production (Iannino et al. 2022; Torres et al. 2024). However, scheduling is also employed in many other areas such as transportation and logistics (Ahn et al. 2017; Guo et al. 2020; Jin 2024), charging electric vehicles Mencía et al. (2019), and, more recently, cloud computing Barredo and Puente (2023).

Historically, scheduling has aimed at optimizing production metrics, with one of the primary objectives being the reduction of makespan, or the duration required to complete all tasks. However, the increasing urgency to address climate change and improve energy efficiency has shifted focus toward sustainability in scheduling practices (Li et al. 2015; Gahm et al. 2016; Alvarez-Meaza et al. 2021). Manufacturing alone accounts for approximately 37% of global resource use Li and Wang (2022), highlighting the pressing need to incorporate environmental considerations. Furthermore, the 2024 report of the International Energy Agency (2024) anticipates a significant rise in electricity demand from data centres, which are expected to double their consumption by 2026 due to the growing reliance on AI and digital infrastructure. As a result, green scheduling is becoming increasingly crucial in cloud computing systems Ghafari et al. (2022).

Energy-related objectives are generally balanced with more conventional production-focused goals, such as minimizing makespan or tardiness Para et al. (2022). A frequently used strategy in the literature is to consider all objectives as equally

✉ Camino R. Vela
crvela@uniovi.es

Sezin Afşar
afsarsezin@uniovi.es

Jorge Puente
puente@uniovi.es

Juan José Palacios
palaciosjuan@uniovi.es

Inés González-Rodríguez
gonzalezri@unican.es

[1] Department of Computer Science, University of Oviedo, Campus of Gijón, 33204 Gijón, Spain

[2] Department of Mathematics, Statistics and Computation, University of Cantabria, 39005 Santander, Spain

important and employ a Pareto-based approach Afşar et al. (2022). However, in real-world production environments, objectives typically have differing levels of importance, which makes a hierarchical approach, such as lexicographic, more appropriate to balance these potentially conflicting objectives. In practice, decision-makers might be willing to accept a slight compromise in the main objective if it results in significant gains in the remaining ones. Lexicographic goal programming can address such scenarios, but determining appropriate goal values for each objective is often context-specific and challenging to define in advance. Moreover, Pareto-optimal solutions are closely related to lexicographically optimal solutions; in fact, a lexicographically optimal solution is always non-dominated. Similarly, optimal solutions to a lexicographic goal-programming problem are non-dominated when minimizing deviations from target values Ehrgott (2005).

The prioritization of makespan over energy efficiency in the objective function is motivated by practical considerations in industrial settings. Green objectives, such as energy efficiency, cannot be pursued in isolation, as industries cannot sacrifice profitability for energy savings at any cost Guggeri et al. (2023). Multi-objective optimization plays a critical role in balancing these competing goals, and research is essential to explore methods that achieve better trade-offs. In practice, customer satisfaction (often reflected by makespan) may take precedence over energy efficiency in certain contexts. By prioritizing makespan as the primary objective and energy efficiency as the secondary one, our approach reflects the real-world trade-offs faced by industries and aligns with the broader motivation of balancing operational and environmental objectives.

In scheduling, the job shop problem (JSP) and its many variants represent an active area of research, modelling diverse technical and social applications Xiong et al. (2022). In this problem, tasks are grouped into jobs, with tasks within the same job requiring completion in a predetermined order on specific machines. A notable variant is the flexible job shop scheduling problem (FJSP), which permits certain tasks to be processed on multiple machines (Xie et al. 2019; Dauzère-Pérès et al. 2024). This added flexibility introduces variability in both processing times and energy use based on the choice of machine.

Stochastic modelling is a common method for dealing with uncertain task durations, where these durations are expressed as probability distributions. However, collecting enough data to accurately estimate such distributions is often challenging, and even with reliable data, the computational requirements can be prohibitive. To overcome these challenges, fuzzy numbers are frequently used as an alternative, particularly in FJSP. This approach reduces the need for detailed data and offers a more computationally efficient solution (Palacios et al. 2015; García Gómez et al. 2023). For situations where prior knowledge is extremely limited, interval-based modelling is another viable option, using only lower and upper bounds to define task durations. These intervals are often treated as uniform probability distributions in the absence of additional information. Despite its simplicity, this method provides robust solutions that are less sensitive to minor fluctuations in task durations Díaz et al. (2022).

Even the simplest variants of the Job Shop Problem (JSP) are NP-hard Lenstra et al. (1977), which justifies the need of approximate metaheuristic search algorithms as effective solution methods Chaudhry and Khan (2016), Fazel Zarandi et al. (2020). These algorithms explore various possible schedules and evaluate their quality based on criteria like makespan, energy consumption, due-date compliance, or machine utilization (Xie et al. 2019; Coelho et al. 2021). Moreover, heuristic seeding is used in the context of scheduling to increase the search quality of the metaheuristic algorithms by including promising solutions in the initial population in Burke et al. (1998), Puente et al. (2003), Palacios et al. (2015).

Despite the interval approach being a basic model for uncertainty, research on interval-based JSP and FJSP is still limited. In Lei (2012), a genetic algorithm is introduced to reduce tardiness in a JSP with interval-based due dates and processing times. A hybrid method combining particle swarm optimization and genetic algorithm is proposed in Li et al. (2019) to solve a FJSP with interval processing times, situated within a larger integrated planning and scheduling context. Multi-objective interval JSPs are analysed in Lei (2013) and Lei and Guo (2015). In Lei (2013), an artificial bee colony algorithm is used to minimise makespan and total tardiness in a JSP involving flexible maintenance and non-resumable jobs. Lei and Guo (2015) focuses on minimizing the carbon footprint and makespan in a dual-resource constrained JSP with heterogeneous resources using a dynamic neighbourhood search in a lexicographic order. More recently, Díaz et al. (2023a) examines a JSP with interval task durations and proposes a memetic algorithm that combines artificial bee colony and local search. The same problem is addressed in Díaz et al. (2023b) with a modified artificial bee colony algorithm that mimics the seasonal behaviour of honeybees for better intensification.

In this work, we address a multi-objective flexible job shop problem with interval processing times. A lexicographic goal programming approach is adopted to minimise makespan and total energy consumption. A genetic algorithm (GA) was proposed in Afşar et al. (2024) to tackle this problem. Here we enhance that method by incorporating heuristic seeding strategies and a post-processing step using a constraint programming model. Our contributions can be summarised as follows:

- Two new heuristic seeding strategies are defined. They are used in isolation and in combination for generating good quality initial population for the GA.
- A constraint programming model is solved using a commercial solver as a final improvement to the GA solutions.

- An extensive experimental study is performed to show that the hybrid algorithm manages to meet more makespan goals and improve the energy efficiency by 32% on average with respect to the previously published methods Afşar et al. (2024).

The problem is defined in Sect. 2. Section 3 provides the detailed explanation of our solution methodology. The experimental study and its results are discussed in Sect. 4, and the conclusions are summarised in Sect. 5.

2 Problem definition

In the deterministic job shop problem (JSP), a set of n jobs, denoted by J, needs to be completed. Each job j comprises a sequence of tasks $O_j = o_{j1}, \ldots, o_{jN_j}$, where each task must be processed on a specific machine from the set of m machines M. Furthermore, the tasks within a job must adhere to a predefined order, known as *job precedence constraints*.

In the flexible job shop problem (FJSP), each task o_{ji} can be assigned to any machine k within a designated set of alternatives, $M(o_{ji})$. The duration of processing a task o_{ji} varies based on the machine selected and is represented as d_{jik}. Tasks are processed without interruption, and no machine is allowed to handle multiple tasks simultaneously.

In the energy-aware version of the FJSP, processing time is not the only parameter influenced by the task and the machine. The *active* power consumption of a machine $k \in M(o_{ji})$ varies depending on the task o_{ji} it performs and is represented as AP_{jik}. Additionally, each machine k requires a specific amount of *passive* power, denoted by PP_k, while it is turned on.

A feasible solution to this problem comprises two parts: $\boldsymbol{\tau}$, which assigns each task to a specific machine, and $\boldsymbol{s}$, a schedule that determines the start times of tasks on their assigned machines while adhering to all constraints. This study optimises two objectives in a lexicographic goal programming manner: first we minimise the makespan, and when it reaches the goal we continue with minimising overall energy consumption.

In practical scenarios, the exact duration required to complete a task is often uncertain and cannot be determined in advance. Typically, only approximate estimates of the time are available. When the minimum and maximum durations for each task are known, this uncertainty can be modelled as a closed interval, expressed as $\mathbf{a} = [a^1, a^2] = \{x \in \mathbb{R} : a^1 \leq x \leq a^2\}$. In this study, we refer to the FJSP with interval-based uncertainty as the *Interval Flexible Job Shop* (IFJSP).

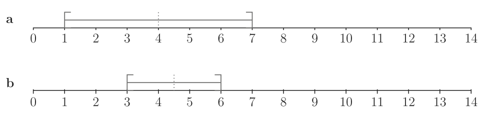
To evaluate the objective functions for a schedule with interval processing times, we utilize arithmetic operations such as *addition* and *maximum* between intervals, as well as *addition* and *multiplication* of a scalar with an interval. Let $\mathbf{a} = [a^1, a^2]$ and $\mathbf{b} = [b^1, b^2]$ represent two intervals. The addition operation is defined as $\mathbf{a} + \mathbf{b} = [a^1 + b^1, a^2 + b^2]$, and the maximum is calculated as $\max(\mathbf{a}, \mathbf{b}) = [\max(a^1, b^1), \max(a^2, b^2)]$. For an interval $\mathbf{a}$ and a scalar $r \in \mathbb{R}$, the addition and multiplication are expressed as $\mathbf{a} + r = [a^1 + r, a^2 + r]$ and $r\mathbf{a} = [ra^1, ra^2]$, respectively. To illustrate the operations between intervals, we show in Fig. 1 the intervals obtained by using the *addition*, *maximum*, and *multiplication* of a scalar with an interval for the intervals $\mathbf{a} = [1, 7]$ and $\mathbf{b} = [3, 6]$

The set of closed intervals is not endowed with a natural total ordering. However, in the context of interval scheduling it is necessary to order intervals for two different purposes: to ensure the *feasibility* of a solution and to guarantee its *optimality*. Regarding the former, for precedence and non-overlapping constraints to hold, if two tasks are consecutively scheduled, the first task must finish before the second one starts. Taking into account the interval arithmetic, this translates into a component-by-component comparison. Specifically, we say that $\mathbf{a} \leq_C \mathbf{b}$ holds if and only if $a^1 \leq b^1$ and $a^2 \leq b^2$. Regarding the latter, a ranking method is needed to compare solutions in terms of the objective function value and decide which one is *the optimal* one. Several ranking methods have been proposed in the literature (Karmakar and Bhunia 2012). Among these, the midpoint (MP) ranking has gained popularity in recent interval scheduling studies (e.g., Díaz et al. 2023b), In this approach, $\mathbf{a} \leq_{MP} \mathbf{b}$ holds if and only if $m(\mathbf{a}) \leq m(\mathbf{b})$, where $m(\mathbf{a}) = (a^1 + a^2)/2$. In the example in Fig. 1a, $m(\mathbf{a})$ and $m(\mathbf{b})$ are represented with a red dashed line; we can easily see that $\mathbf{a} \leq_{MP} \mathbf{b}$.

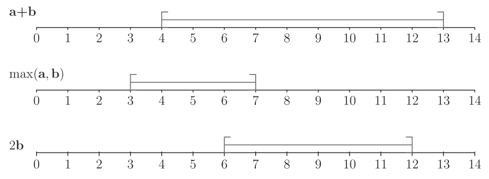
2.1 The objectives

The primary objective is to minimise the makespan, which is the completion time of all jobs. Let $\boldsymbol{s}_{jN_j} = [s^1_{jN_j}, s^2_{jN_j}]$ denote the starting time of the final task of job j. The makespan can

Fig. 1 Arithmetic operations on intervals



(a) Intervals $\mathbf{a} = [1, 7]$ and $\mathbf{b} = [3, 6]$



(b) Interval operations $\mathbf{a} + \mathbf{b} = [4, 13]$, $\max(\mathbf{a}, \mathbf{b}) = [3, 7]$ and $2\mathbf{b} = [6, 12]$

then be expressed as $\boldsymbol{C}_{\max} = \max\{\boldsymbol{s}_{jN_j} + \boldsymbol{d}_{jN_jk_{jN_j}} : j \in J\}$. Since it is represented as an interval, we apply the midpoint ranking method to determine the best makespan value.

In energy-aware scheduling, various approaches have been proposed to factor in energy consumption during schedule optimization. For instance, in González et al. (2017) the authors focus on the energy consumed while machines are idle, while in García Gómez et al. (2023) it is assumed that all machines remain continuously *on* from start to finish of the schedule. In this study, we assume each machine $k \in M$ begins in the *off* state and consumes no energy until it is turned *on* by an operator. It is reasonable to assume that workers activate machines only when necessary, specifically when the first task assigned to the machine, f_k, is ready to start, denoted by $s^1_{f_k}$, where $\boldsymbol{s}_{f_k} = [s^1_{f_k}, s^2_{f_k}]$. Once turned *on*, the machine remains operational until its last task, l_k, is completed, and then it is switched *off* after the completion time, $c^2_{l_k} = s^2_{l_k} + d^2_{l_k}$, where $\boldsymbol{c}_{l_k} = \boldsymbol{s}_{l_k} + \boldsymbol{d}_{l_k}$ based on the assignment $\boldsymbol{\tau}$ and schedule $\boldsymbol{s}$. During this period, the machine consumes *passive power* PP_k per unit of time, and its passive energy consumption is computed as $PE_k = PP_k(c^2_{l_k} - s^1_{f_k})$. When the machine is *active*, performing a task o_{ji}, it consumes energy at a rate of AP_{jik} per unit of time. The active energy consumption for machine k is calculated as $\boldsymbol{AE_k} = \sum_{j,i,\tau_{ji}=k}(AP_{jik}\ \boldsymbol{d_{jik}})$, where $j \in J$ and $i \in O_j$. Active energy consumption depends on both the machine and the task's uncertain duration, whereas passive energy is deterministic, based on the operator?s decisions regarding the machine's operation. The total energy consumption is then defined as $\boldsymbol{TE} = \sum_{k \in M}(PE_k + \boldsymbol{AE_k})$. Furthermore, passive energy can be broken down into two components: the energy used during task execution and the *idle energy*, which is the energy consumed when the machine is idle with certainty.

2.2 Constraint programming model

Constraint programming (CP) is a mathematical modelling approach that represents problems through a set of constraints that need to be satisfied. These problems are usually formulated using variables with defined domains and associated constraints. CP solvers search for feasible variable assignments and employ inference methods, like constraint propagation, to narrow the search space and find solutions more efficiently Rossi et al. (2008).

IBM ILOG CP Optimizer (CPO) Laborie et al. (2018) is a CP solver that represents tasks using an *interval variable*[1]. An interval variable denotes a time span characterized by its duration, i.e. the processing time of a task, and a start time that will be determined during the resolution phase. Interval variables can be optional, meaning they may or may not be included in the final solution. If an optional interval variable is not present, it is disregarded in any expressions or constraints it is part of. This allows constraints and objectives to be defined for all interval variables, even if some are not included in the final solution. Another type of variable introduced in the CPO is *sequence variable*, which refers to a collection of (optional) interval variables that are initially unordered but will be arranged into a specific sequence in the solution. Sequence variables allow us to create a set of interval variables and impose constraints on them in a concise way. For instance, it is possible to model a machine or resource as a *sequence of intervals*, and then use the `noOverlap` method to forbid the tasks to overlap on the machine.

In our problem, since the task duration on machine k is represented as an interval, i.e., $\boldsymbol{d}_{jik} = [d^1_{jik}, d^2_{jik}]$, we model each task as a pair of optional interval variables, namely x^1_{jik} and x^2_{jik}, with the corresponding durations. Together they represent the interval $\boldsymbol{x}_{jik}$, which is the potential starting time of task o_{ji} on machine k if machine k is selected to process this task. Likewise, the interval variables s^1_{ji} and s^2_{ji} represent the effective starting time of task o_{ji}, the interval $\boldsymbol{s}_{ji}$. The interval variables y^1_k and y^2_k stand for the machine k whereas q^1_k and q^2_k are the sequence variables that include y^1_k and y^2_k, respectively.

$C^1_{\max}$ and $C^2_{\max}$ are the integer variables corresponding to lower and upper bounds of the makespan. AE^1_{jik} and AE^2_{jik} represent the active energy consumption of machine k while processing the task o_{ji}, PE_k is the passive energy use of machine k while it is on and finally TE^1_k and TE^2_k denote the total energy consumption of machine k.

$$\min\{m(\boldsymbol{C}_{\max}), m(\boldsymbol{TE})\}$$

$$\text{s.t.} \tag{1}$$

$$C^l_{\max} \geq \mathtt{endOf}(s^l_{jN_j}) \quad \forall j \in J)\forall l \in \{1,2\}$$

$$\mathtt{alternative}(s^l_{ji}, \{x^l_{jik}, k \in M(o_{ji})\})\forall j \in J, i \in O_j, l \in \{1,2\} \tag{2}$$

$$\mathtt{startBeforeStart}(s^1_{ji}, s^2_{ji})\forall j \in J, i \in O_j \tag{3}$$

$$\mathtt{endBeforeStart}(s^l_{ji}, s^l_{j,i+1})\forall j \in J, i \in O_j, iN_j, l \in \{1,2\}$$

$$\mathtt{noOverlap}(\mathtt{q}^1_\mathtt{k})\forall \mathtt{k} \in \mathtt{M}, \mathtt{l} \in \{1,2\} \tag{4}$$

$$\mathtt{sameSequence}(q^1_k, q^2_k)\forall k \in M \tag{5}$$

$$PE_k = PP_k * (\mathtt{endOf}(\mathtt{span}(y^2_k)) - \mathtt{startOf}(\mathtt{span}(y^1_k)))\forall k \in M \tag{6}$$

$$AE^l_k = \sum_{j \in J, i \in O_j} AP_{jik} * d^l_{jik} * \mathtt{presenceOf}(x^l_{jik})\forall k \in M, l \in \{1,2\} \tag{7}$$

$$TE^l = \sum_{k \in M} PE_k + AE^l_k \forall l \in \{1,2\} \tag{8}$$

[1] It is important to distinguish this from the uncertainty in processing times, which is represented as intervals.

Equation (1) defines the makespan, which takes the value of the maximum completion time across all tasks for each component l. In constraint (2), the `alternative` expression of CPO is used to ensure that only one of the optional intervals from $\{x^l_{jik}, k \in M(o_{ji})\}$ is chosen, and that it starts and ends with s^l_{ji}. This constraint guarantees that a task is assigned to one of the available machines. The order between the components of $\boldsymbol{s_{ji}}$ is enforced by constraint (3). The task precedence constraints of a job are respected due to the constraint (3). Constraint (4) ensures that tasks executed on the same machine do not overlap. The *sequence variables* q^1_k and q^2_k, along with the `sameSequence` expression in constraint (5), enforce a consistent processing order for tasks in both y^1_k and y^2_k. Equation (6) calculates the passive energy consumption for machine k by using the `span` expression, which defines an interval from the start of the first task to the end of the last task assigned to machine k. Equation (7) determines the active energy consumption for machine k by using the `presenceOf` expression, which outputs a value of 1 if machine k is processing task o_{ji} and 0 otherwise. Finally, Eq. (8) calculates the total energy consumption across all machines and tasks.

In Sect. 4, constraints (1)–(8) are solved by minimising the makespan ($m(\boldsymbol{C}_{\max})$) and the total energy ($m(\boldsymbol{TE})$) separately. These single-objective models are referred to as CP_{MS} and CP_{TE}, respectively.

3 Solution methodology

Flexible job shop problem with interval processing times that minimises makespan and total energy use by a lexicographic goal programming approach is tackled in this study. In Afşar et al. (2024) a genetic algorithm (GA) has been proposed to solve this problem. Although the GA provides a good starting point, it is possible to improve its results significantly. In this work, we propose a hybrid evolutionary algorithm (HEA) that has a GA at its core whilst incorporating two additional strategies: heuristic seeding and post-processing.

The HEA starts by generating a pool of solutions using one of the heuristic seeding methods that are explained in more detail in Sect. 3.1. If the method is used to create less than 100% of the population, then the rest is generated randomly. Afterwards, the GA is applied to the population until maximum number of generations ($\max_{iter}$) is reached. More detailed information on the operators of the GA can be found in Afşar et al. (2024). Then, a post-processing step that is described in Sect. 3.2 is applied to the best solution found by the GA. A pseudo-code description of the HEA can be found in Algorithm 1.

Algorithm 1 Hybrid Evolutionary Algorithm (HEA)

```
1: $Best \leftarrow \emptyset$
2: Apply HS to obtain a pool $P_0$ of heuristic initial solutions.
3: Complete $P_0$ with random solutions if needed and evaluate all chromosomes of $P_0$
4: $i \leftarrow 0$
5: while $i < \max_{iter}$ do
6:     $\mathrm{Off}(P_i) \leftarrow$ Apply selection, crossover and mutation operators to $P_i$
7:     Evaluate $\mathrm{Off}(P_i)$
8:     $P_{i+1} \leftarrow$ Apply 4:2 tournament without repetition to $P_i \cup \mathrm{Off}(P_i)$
               using lexicographic comparison
9:     Update $Best$ with the best solution in $P_{i+1}$
10:    $i \leftarrow i+1$
11: if $m(\boldsymbol{C_{max}}(Best)) \leq Goal$ then
12:     $Best \leftarrow$ solve $CP_{TE}(Best, Goal)$
13: else
14:     $Best \leftarrow$ solve $CP_{MS}(Best)$
15: return $Best$
```

3.1 Heuristic seeding

Typically a genetic algorithm starts with a randomly-generated initial population and evolves throughout a number of iterations to optimise an objective function. One of the methods to improve its results is to use a heuristic algorithm while generating the initial population, called as *heuristic seeding* Puente et al. (2003) in the literature. Heuristic seeding in the context of scheduling entails building a feasible schedule from scratch by iteratively assigning a starting time to a non-scheduled operation regarding problem-specific constraints.

In this study, we start from a randomly generated permutation σ of the operations that represents a feasible operation processing order, i.e., respecting the job precedence constraints. Then we proceed with assigning a machine and a starting time using an insertion policy to each operation in permutation σ in order. This way we ensure that when the first unscheduled operation o_{ji} is selected to be scheduled, all the predecessor operations in its job are already scheduled, that is, $\forall i' < i, \exists \tau_{ji'}$ and $\mathbf{s}_{ji'}$ unless i is the first operation in the job j, in which case we consider $\mathbf{s}_{ji'} = \mathbf{0}$, and $\mathbf{d}_{ji'} = \mathbf{0}, i' < i$.

3.1.1 Makespan oriented heuristic (H1)

The first heuristic algorithm (H1) focuses on minimising makespan while assigning a machine k and a starting time $\mathbf{s}_{ji}$ for an operation $o_{ji} \in \sigma$.

Given a partial schedule, let us assume that there are two operations $o_{j'i'}$ and $o_{j''i''}$ consecutively scheduled in machine k (i.e., $\tau_{j'i'} = \tau_{j''i''} = k$) that come before o_{ji} in σ and such that $\mathbf{c}_{j''i''} >_C \mathbf{c}_{j'i'} \geq_C \mathbf{c}_{j,i-1}$. In other words, both $o_{j'i'}$ and $o_{j''i''}$ would be completed after the job predecessor of o_{ji}, namely $o_{j,i-1}$.

If there exists at least one feasible insertion gap for o_{ji} in machine k, then the earliest starting time for o_{ji} in k would be

$$\mathbf{est}_{jik} = \min\{\mathbf{c}_{j'i'} : \mathbf{c}_{j'i'} + \mathbf{d}_{jik} \leq_C \mathbf{s}_{j''i''}\} \quad (9)$$

If there is no such gap, then the earliest starting time for o_{ji} would be

$$\mathbf{est}_{jik} = \max\{\mathbf{c}_{j'i'} : \mathbf{c}_{j'i'} \geq_C \mathbf{c}_{j,i-1}\} \quad (10)$$

The task is then scheduled at machine $k^* \in M(o_{ji})$, where:

$$\mathbf{est}_{jik^*} + \mathbf{d}_{jik^*} \leq_C \mathbf{est}_{jik} + \mathbf{d}_{jik} \ \forall k \in M(o_{ji}) \ k \neq k^* \quad (11)$$

Once the machine k^* where o_{ji} is going to be executed is selected, we assign $\mathbf{s}_{ji} = \mathbf{est}_{jik^*}$ and $\tau_{ji} = k^*$ and continue to schedule the next operation in σ.

Algorithm 2 shows the pseudo-code of a function calculating $\mathbf{est}_{jik}$ for a given task o_{ji} and machine k. Based on it, the pseudocode of H1 is detailed in Algorithm 3.

Algorithm 2 Function $est(O_s, o_{ji}, k)$ to find $\mathbf{est}_{jik}$

```
Require: A set of scheduled tasks $O_s$, an operation $o_{ji}$, a machine $k$
  if $i > 0$ then
    est = $\mathbf{c}_{ji-1}$
  else
    est = $[0, 0]$
  $P \leftarrow \{o_{j'i'} \in O_s : \mathbf{c}_{j'i'} \geq_c \mathbf{est} \wedge \tau_{j'i'} = k\}$
  Sort $P$ in ascending order by $\mathbf{c}_{j'i'}$ using $\leq_{MP}$
  gap ← false
  while $P \neq \emptyset$ and not gap do
    $o_{j'i'}$ ← First element in $P$
    if $\mathbf{est} + \mathbf{d}_{jik} \leq_c \mathbf{s}_{j'i'}$ then
      gap ← true
    else
      est ← $\mathbf{c}_{j'i'}$
  return est
```

Algorithm 3 Makespan Oriented Heuristic (**H1**)

```
**Require:** An IFJSP instance, a permutation $\sigma$ of tasks
  $O_s \leftarrow \emptyset$
  **for all** $o_{ji} \in \sigma$ **do**
    $\mathbf{Best} \leftarrow [\infty, \infty]$
    **for all** $k \in M(o_{ji})$ **do**
      $\mathbf{start} \leftarrow est(O_s, o_{ji}, k)$
      **if** $\mathbf{start} \leq_{MP} \mathbf{Best}$ **then**
        $\mathbf{Best} \leftarrow \mathbf{start}$
        $\tau_{ji} \leftarrow k$
        $\mathbf{s}_{ji} \leftarrow \mathbf{start}$
        $\mathbf{c}_{ji} \leftarrow \mathbf{start} + \mathbf{d}_{jik}$
    $O_s \leftarrow O_s \cup o_{ji}$
  **return** The machine assignment $\tau$ and starting times $\mathbf{s}$ of each task
```

3.1.2 Active energy oriented heuristic (H2)

The second heuristic concentrates on minimising the active energy consumption whilst watching over the completion time of the current partial schedule. In order to do that, it looks for a machine k to process o_{ji} where the active energy consumption is the smallest and that does not increase the maximum completion time of the current partial schedule if possible.

Let us denote the completion time of the machine k in the current schedule as $\mathbf{cc}_k$, and the maximum completion time of all machines in the current schedule as $\mathbf{CC}_{max}$:

$$\mathbf{cc}_k = \max\{\mathbf{c}_{j'i'} : \tau_{j'i'} = k, o_{j'i'} \prec_\sigma o_{ji}\} \tag{12}$$

$$\mathbf{CC}_{max} = \max_{k \in M}\{\mathbf{cc}_k\} \tag{13}$$

And let us define a set of machines $M'(o_{ji}) \subseteq M(o_{ji})$ as:

$$M'(o_{ji}) = \{k \,:\, \mathbf{est}_{jik} + \mathbf{d}_{jik} \leq_{MP} \mathbf{CC}_{max}\} \tag{14}$$

If $M'(o_{ji}) \neq \emptyset$, we select a machine $k \in M'(o_{ji})$ such that:

$$AP_{jik}\mathbf{d_{jik}} \leq_{MP} AP_{jik'}\mathbf{d_{jik'}} \;\; \forall k' \in M'(o_{ji}), k' \neq k \tag{15}$$

Otherwise, we select a machine k applying Eq. (15) on the set $M(o_{ji})$ instead of $M'(o_{ji})$. As before, once the machine k where o_{ji} is going to be executed is decided, we assign $\mathbf{s}_{ji} = \mathbf{est}_{jik}$ and $\tau_{ji} = k$, and continue to schedule the next operation in σ.

The pseudocode of H2 can be found in Algorithm 4.

Table 1 $UB_{C_{max}}$ values for each instance

Inst	$UB_{C_{max}}$	Inst	$UB_{C_{max}}$	Inst	$UB_{C_{max}}$	Inst	$UB_{C_{max}}$
07a-l	2187	10a-l	2178	13a-l	2161	16a-l	2148
08a-m	2061	11a-m	2017	14a-m	2161	17a-m	2088
09a-h	2061	12a-h	1969	15a-h	2161	18a-h	2057

Algorithm 4 Active Energy Oriented Heuristic (**H2**)

```
Require: An IFJSP instance, a permutation $\sigma$ of tasks
  $O_s \leftarrow \emptyset$
  $\mathbf{CC} \leftarrow [0, 0]$
  for all $o_{ji} \in \sigma$ do
    $\mathbf{BestAP} \leftarrow [\infty, \infty]$
    $delayed \leftarrow$ true
    for all $k \in M(o_{ji})$ do
      $\mathbf{start} \leftarrow est(O_s, o_{ji}, k)$
      if $\mathbf{start} + \mathbf{d}_{jik} \leq_{MP} \mathbf{CC}$ then
        if $AP_{jik}\mathbf{d}_{jik} <_{MP} \mathbf{BestAP}$ or $delayed$ then
          $\mathbf{BestAP} \leftarrow AP_{jik}\mathbf{d}_{jik}$
          $\tau_{ji} \leftarrow k$
          $\mathbf{s}_{ji} \leftarrow \mathbf{start}$
          $\mathbf{c}_{ji} \leftarrow \mathbf{start} + \mathbf{d}_{jik}$
          $delayed \leftarrow$ false
      else if $delayed$ and $AP_{jik}\mathbf{d}_{jik} <_{MP} BestAP$ then
        $BestAP \leftarrow AP_{jik}\mathbf{d}_{jik}$
        $\tau_{ji} \leftarrow k$
        $\mathbf{s}_{ji} \leftarrow \mathbf{start}$
        $\mathbf{c}_{ji} \leftarrow \mathbf{start} + \mathbf{d}_{jik}$
  $O_s \leftarrow O_s \cup o_{ji}$
  return The machine assignment $\tau$ and starting times $\mathbf{s}$ of each task
```

With these heuristic seeding strategies we can generate the whole initial population or a portion of it. To obtain a new heuristic solution we only need a new random σ.

3.2 Post-processing

Mathematical models are frequently employed to identify the optimal solution for small-scale instances in scheduling problems (Öztop et al. 2020; Fattahi et al. 2007; Lunardi et al. 2020). However, for larger instances, solvers fail to reach even a feasible solution within reasonable time limits. Consequently, metaheuristics have emerged as the most successful approach in such cases.

In an effort to achieve the optimal balance between both worlds, we have integrated a commercial CP solver, specifically IBM ILOG CP Optimizer (CPO) (IBM 2020) in our work-flow. This solver receives the best solution obtained through the evolutionary process, *Best*, as a starting point, and solves the CP model given in Sect. 2.2 enhancing the precision and reliability of our solutions.

Given that we are solving a lexicographical goal-programming problem, CPO solves the CP_{MS} model when *Best* does not meet the goal. If *Best* reaches the makespan goal, then CPO solves the CP_{TE} model along with an additional constraint:

$$m(\mathbf{C}_{\max}) \leq Goal. \quad (16)$$

The benefit of this additional step is twofold: if the evolutionary algorithm has not yet achieved the makespan goal, it helps reaching this target; if the goal has already been met, it further minimizes the total energy consumption while ensuring that the makespan does not exceed the goal value thanks to constraint (16).

Table 2 LBs for $m(\mathbf{TE})$ values

Inst	LB_{TE}	Inst	LB_{TE}	Inst	LB_{TE}	Inst	LB_{TE}
07a-l	5,074,850	10a-l	4,862,820	13a-l	6,797,380	16a-l	6,408,300
08a-m	4,280,450	11a-m	4,636,950	14a-m	6,382,990	17a-m	5,686,210
09a-h	4,361,320	12a-h	4,083,080	15a-h	5,267,490	18a-h	5,379,080

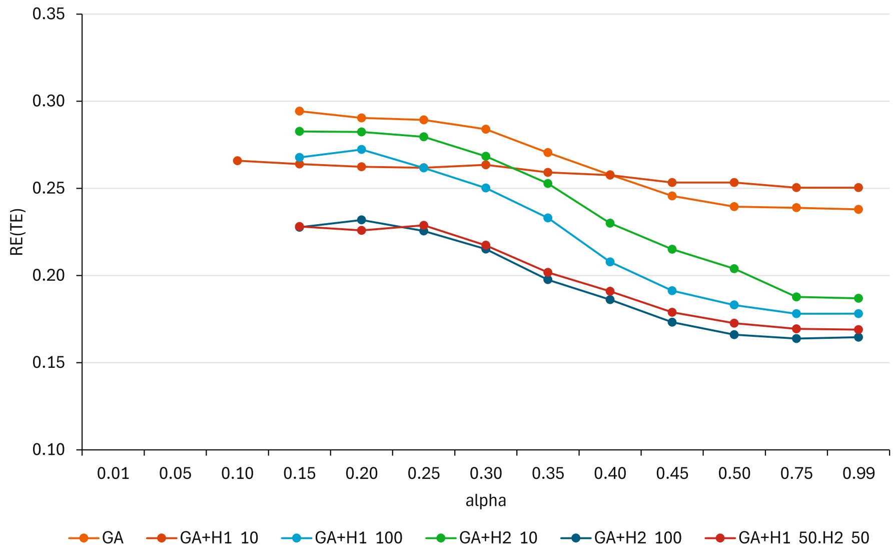


Fig. 2 GA random with initial population versus GA with different initial populations

4 Experimental study

In this section we present the empirical evaluation of the proposed Heuristic Evolutionary Algorithm (HEA). Our method has been built over the foundations of the Genetic Algorithm (GA) from Afşar et al. (2024), which to the best our knowledge, constitutes the current state of the art for our problem together with CP Optimizer (CPO). We shall conduct a first set of experiments to evaluate the impact of including heuristic seeding in the initial population. Once the best strategy is found in that regard, we include our second enhancement: the constraint-programming post-processing. The final algorithm, including both improvements, is then compared with significant longer runs of CPO solving the presented CP model.

In the course of these experiments, we will employ the 12 instances from Afşar et al. (2024), which are built on the instances used in García Gómez et al. (2023) for the flexible job shop with fuzzy duration. These instances are, in turn, adaptations of the well-known test-bed presented in Dauzère-Pérès and Paulli (1997). Instances $\{07a, \ldots, 12a\}$ have 15 jobs spanning 293 tasks to be executed using 8 resources whereas instances $\{13a, \ldots, 18a\}$ are larger, having 20 jobs, 387 tasks and 10 resources. Besides size, flexibility also plays an important role in instance difficulty. We divide the instances in three groups based on the average number of alternative machines per task, μ: low flexibility instances (l) are $\{07a, 10a, 13a, 16a\}$ with $\mu \leq 1.5$, medium flexibility instances (m) are $\{08a, 11a, 14a, 17a\}$ with $1.5 < \mu \leq 3.0$ and high flexibility ones (h) are $\{09a, 12a, 15a, 18a\}$ with $\mu > 3.0$. Higher flexibility levels mean larger solution spaces, so type h instances are expected to be more difficult. In the following, we append the flexibility level to each instance name for the sake of clarity (e.g. 08a-m).

For each instance, the makespan target is set to $Goal = (1 + \alpha)UB_{C_{max}}$, where $0 < \alpha < 1$. The value $UB_{C_{max}}$ is obtained by solving the CP model defined in Sect. 2.2 using IBM ILOG CP Optimizer 12.9 during 6 h. The corresponding $UB_{C_{max}}$ values are given in Table 1.

Obviously, the larger is α, the more attainable is the goal for $\mathbf{C_{max}}$. Given the nature of the problem, this would lead to a larger search space to find solutions with good **TE** values that still meet the $\mathbf{C_{max}}$ goal, putting more emphasis on this objective function. Conversely, when α is small, the focus never shifts from minimising $\mathbf{C_{max}}$, thus ignoring **TE** or having little room to find solutions with good **TE** values. For this study, we employ thirteen different values of α: 0.01, 0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45, 0.50, 0.75 and 0.99.

For the sake of fair comparisons with the previously-published *GA* method from Afşar et al. (2024), we use the same population size and stopping criterion. That is, a population size of 100, and 10,000 generations as stopping criterion. To ensure the reliability of the results, each

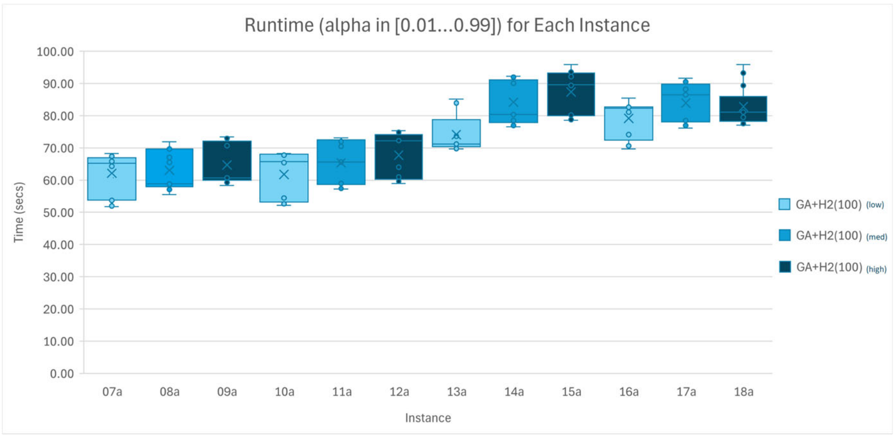


Fig. 3 Runtimes of GA+H2 (100) with respect to instance flexibility levels

method is run ten times. All experiments are run on a Linux machine equipped with two Intel Xeon Gold 6240 processors and 128GB of RAM. The full set of results is available at the following URL: http://di002.edv.uniovi.es/iscop.

Due to the goal programming nature of our problem, the main target is always to meet the makespan goal. Once reached, the goodness of a solution relies on the value of the secondary objective function, that is **TE**. Therefore, to compare the results of different methods over different instances, we shall first assess for which instances and α values the method is capable of finding solutions that meet the $\mathbf{C_{max}}$ goal. For the instances where the goal is met, we compare solutions based on the *relative error* (RE) between the midpoint of the **TE** values and the best known lower bound, LB_{TE}, defined as:

$$RE(TE) = \frac{m(\mathbf{TE}) - LB_{TE}}{LB_{TE}}.$$

As in Afşar et al. (2024), the LB_{TE} values are obtained with a heuristic method for which all tasks are processed on their most *energy-efficient* machines (i.e., those with the lowest active energy consumption for the task) without pausing the machine between tasks. The LB_{TE} values provides by this method are reported in Table 2.

4.1 Heuristic seeding analysis

Firstly, we test the contribution of the heuristic seeding in the Genetic Algorithm. To this purpose, we compare the two proposed heuristics, the Makespan Oriented Heuristic (H1) and the Active Energy Oriented Heuristic (H2), using them to generate different percentages of the initial population. The remaining solutions are generated at random, as in Afşar et al. (2024). We test the following setups:

- All initial solutions randomly generated (*GA* from Afşar et al. (2024))
- 10% with heuristic H1 and 90% at random (H1_10)
- 100% with heuristic H1 (H1_100)
- 10% with heuristic H2 and 90% at random (H2_10)
- 100% with heuristic H2 (H2_100)
- 50% with heuristic H1 and 50% with heuristic H2 (H1_50.H2_50)

In Fig. 2 we plot the average *RE*(*TE*) values obtained with all methods in the high flexibility instances for all α values. If a method does not reach the $\mathbf{C_{max}}$ goal in all *h*-type instances for a given α, its *RE*(*TE*) is not plotted, since it is considered irrelevant given that $\mathbf{C_{max}}$ remains the main objective function. The figure focuses only on high flexibility instances because they are considered more difficult, having larger solution spaces. However, the figure is quite representative of the performance on the other instances. If something, the differences in lower flexibility instances are smaller.

It is easy to see that the tightest goals (smallest α values) are hard to achieve by the *GA* with and without heuristic seeding. Only one of the variants of *GA* is able to reach the target in all considered instances for $\alpha = 0.10$. For all other variants, the goal is always met for instances with $\alpha \geq 0.15$. However, we have seen that when *GA* is seeded with H1, especially with H1_100, it tends to show premature convergence. This might be a consequence of having initial solutions so focused on $\mathbf{C_{max}}$ that the population rapidly converges to areas of the search space that are good in makespan, but then lacks the diversity to take advantage of a more flexibility-oriented objective function such as **TE**. Note that solutions with good makespan do indeed have a reasonably good energy value, so good makespan-oriented solutions can easily pose as local minimum for **TE**. This is also supported by the fact that H1_10 is actually the only variant of *GA* meeting the goal when $\alpha = 0.10$ and, in general, when α is small and most of the search is devoted to optimise $\mathbf{C_{max}}$, the variants with H1 offer good results. However, they struggle to optimise **TE** once the goal is achieved.

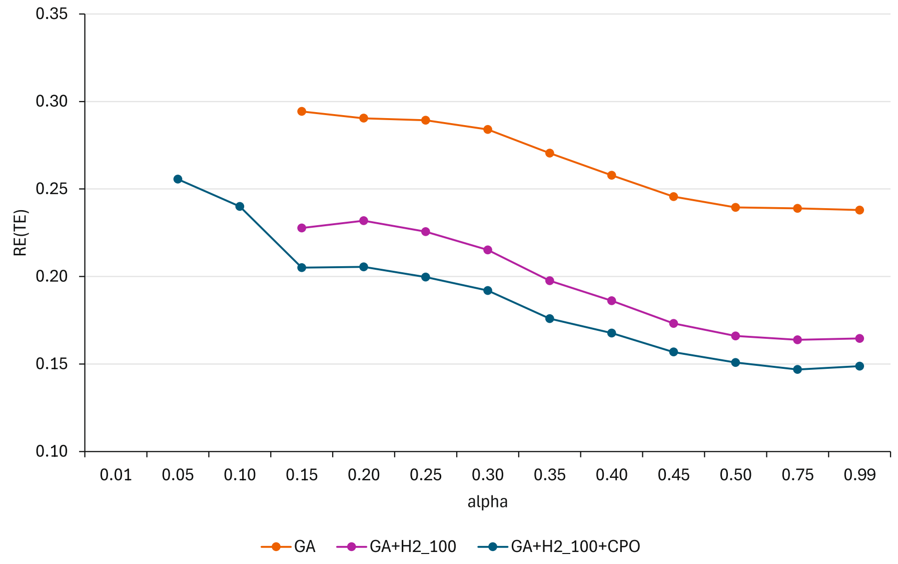


Fig. 4 GA with and without post-processing

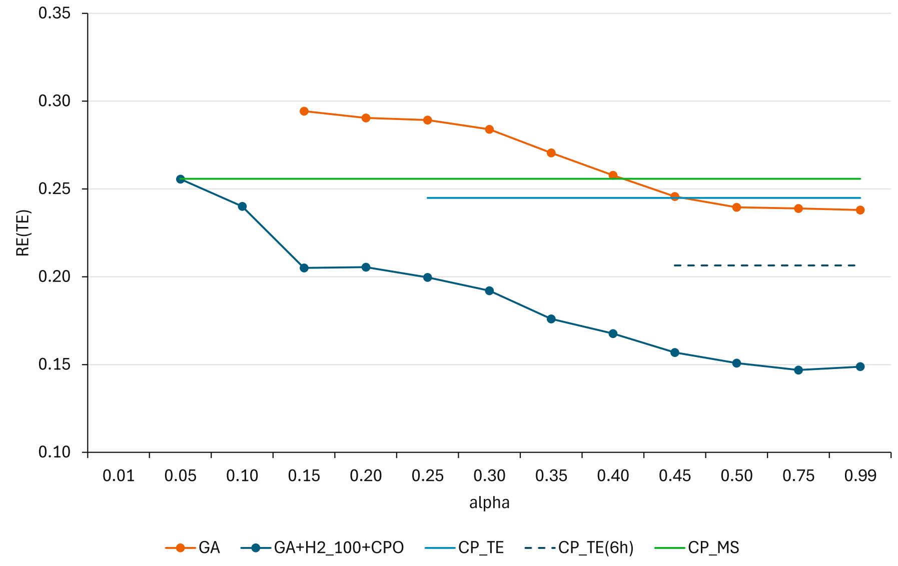


Fig. 5 HEA versus GA without post-processing

On the other hand, when including the heuristic H2, we can see that the goal is also achieved for all $\alpha \geq 0.15$, but the obtained **TE** values are better. When α increases and the makespan goal is easier to reach, the use of H2 starts showing bigger differences, using the extra runtime to focus better on energy optimisation. In that regard, the best values are achieved when the initial population is generated either 50% with each heuristic, or completely with H2, being slightly better in the latter. In consequence, this will be our heuristic seeding method for the GA in the rest of the experiments.

Before analyzing the contribution of the post-processing step, we first discuss the efficiency and scalability of the selected setup, namely the GA with an initial population 100% generated by H2, hereafter referred to as GA+H2 (100). Figure 3 presents the runtimes of each instance as a boxplot across all values of α. Instances with low, medium, and high flexibility levels are depicted in light, medium, and dark blue, respectively. As expected, smaller instances require less computational time compared to larger ones. The method escalates linearly with the size, increasing the runtime 29% in average from smaller instances (293 tasks) to larger ones (387 tasks). Additionally, the flexibility level of an instance significantly impacts the runtime: even among instances of the same size, higher flexibility leads to longer execution times due to the increased number of options and, consequently, a larger solution space. Furthermore, the lower end of each boxplot corresponds to the lowest α values, while the upper end represents higher α values. This indicates that a more relaxed makespan goal also results in a more computationally challenging problem.

4.2 Constraint programming post-processing

The second improvement we propose to complete our Hybrid Evolutionary Algorithm (*HEA* from now) is the post-processing of solutions using Constraint Programming. Since our approach is meta-heuristic, we consider that runtime is an important factor. Therefore, for the post-processing we run CPO with a time limit of 300 s in total, returning the best upper bound found within that period.

Figure 4 shows the average *RE*(*TE*) values achieved by *HEA* and the *GA* with H2_100 heuristic seeding without the post-processing. As before, the average is computed across all high flexibility instances for all α values, and the *RE*(*TE*) values are only plotted if the method reaches the $\mathbf{C_{max}}$ goal in all considered instances for a given α. Nevertheless, the figure is also representative of the performance of all methods in the low and medium flexibility instances. The advantages of including the post-processing are very clear in the figure. The full *HEA* is able to meet more restrictive $\mathbf{C_{max}}$ goals than the *GA* from Afşar et al. (2024) and the *GA* with heuristic seeding alone. In fact, it is able to reach the goal in all instances with $\alpha \geq 0.05$, whereas the other methods can do so, only when $\alpha \geq 0.15$. Even when all methods can achieve the goal ($\alpha \geq 0.15$), *HEA* finds better *RE*(*TE*) values for all α values.

4.3 Comparisons with the state-of-the-art algorithms

To the best of our knowledge, the only known results for our problem are those obtained by the *GA* in Afşar et al. (2024), which we complement by solving the CP Model of the problem using the commercial solver IBM ILOG CP Optimizer (*CPO*). Since we run the solver for 300 s as part of the post-processing in *HEA*, we give it the same time to solve the full model optimising either $\mathbf{C_{max}}$ (CP_{MS}) or **TE** (CP_{TE}). As in previous sections, Fig. 5 shows the average *RE*(*TE*) values obtained with the different methods over the high-flexibility instances when the $\mathbf{C_{max}}$ is met in all of the instances for a given α.

We can appreciate that when *CPO* optimises makespan, it reaches the goal for the same instances as *HEA* ($\alpha = 0.05$), which means that *HEA* is as good as *CPO* regarding the main objective function. In fact, the **TE** values of the CP solutions are quite good, even through CP only optimises makespan. This is similar to the behaviour observed when testing the H1 heuristic seeding in previous sections: that is, minimising makespan leads to promising solutions in terms of **TE**, but does not leave room to improve beyond that.

When *CPO* is used to optimise **TE**, it obtains better results than CP_{MS} in this objective function, but it does not reach the makespan goals when they are more restrictive, as expected. What is more significant is that even optimising only **TE**, *CPO* cannot offer results as good as those obtained by *HEA*. We have included an extra test in the figure, which is letting *CPO* run for 6 h when optimising **TE**. Given the extra time, *CPO* find solutions with much better **TE** values at the cost of sacrificing meeting the $\mathbf{C_{max}}$ goal (is only met for $\alpha \geq 0.45$). But even in this case, the results are far from the results obtained with *HEA* for the same α values.

These experiments show that when *CPO* is not fed with a good solution from the *GA* component, it cannot reach solutions as good as those obtained by the *HEA*. In fact, we have seen that *HEA* is capable of meeting the same $\mathbf{C_{max}}$ goals as *CPO* optimising only makespan, and at the same time obtain better **TE** values than *CPO* optimising specifically this objective function for a longer time. They also illustrate that the problem of reducing energy consumption while maintaining an acceptable makespan value is not trivial, especially when the $\mathbf{C_{max}}$ thresholds are tighter.

5 Conclusions and future work

This work addresses a green flexible job shop scheduling problem, aiming to minimise both makespan and total energy consumption which reflects the real-world trade-offs faced by production facilities and aligns with the broader motivation of balancing operational and environmental objectives. As these objectives typically have differing levels of importance for industrial decision-makers, a lexicographic approach is proposed to find a compromise between these competing objectives. However, a strict lexicographic approach may overemphasize the primary objective and neglect the others. In practice, decision-makers might be willing to accept a slight compromise in the main objective if it results in significant gains in the remaining ones. Therefore, lexicographic goal programming is chosen to tackle this problem.

Although makespan is a well-studied objective in the literature, energy objectives are relatively new in the scheduling field. In our work, to calculate total energy consumption, we assume that operators turn on a machine with the beginning of the first task scheduled on it and turn it off when the last one is completed. To solve the problem, we have extended the Genetic Algorithm from Afşar et al. (2024) in two ways. We have included a heuristic seeding, proposing two different heuristic strategies to create higher quality solutions in the initial population: one more makespan oriented and the other more energy oriented. A post-processing based on Constraint Programming is also included at the end of the genetic algorithm. The empirical study shows that among the heuristic strategies, the energy-oriented heuristic seeding is capable of meeting quite restrictive makespan goals while offering large improvements in energy optimisation. The post-processing improves the algorithm even further, allowing it to meet very restrictive goals while improving energy consumption at the same time. In a comparison with the state of the art and the commercial solver CP Optimizer, our method is capable of meeting the same goals as *CPO* in the main objective function while obtaining more energy efficient solutions than *CPO* optimising specifically this objective function for 6 h. In summary, we have proposed a well-balanced method that matches the best results on the primary objective function but takes better advantage of the goal programming setting to then outperform methods that are specific to that function.

Green scheduling is an evolving field with numerous avenues yet to be explored. One promising direction for future research involves developing energy consumption models that account for the intermittent nature of renewable energy generation. Additionally, investigating the influence of fluctuating energy prices on scheduling decisions could yield valuable insights. Finally, integrating additional scheduling constraints, such as setup times, into the green flexible job shop scheduling problem presents another worthwhile area for further study.

Acknowledgements This research has been supported by the Spanish Government under research Grants TED2021-131938B-I00 and PID2022-141746OB-I00.

Author contributions S.A. Conceptualization, methodology, writing—original draft, software, formal analysis, investigation, data curation, visualization. J.P.: Supervision, software, writing—review, methodology. J.J.P.: Methodology, writing—review and editing, investigation, visualization. I.G-R.: Supervision, conceptualization, writing—review and editing, methodology. C.R.V: Supervision, writing—review and editing, conceptualization, methodology.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature.

Data availability http://di002.edv.uniovi.es/iscop.

Declarations

Competing interests The authors declare that they have no competing interests.

References

Afşar S, Palacios JJ, Puente J, Vela CR, Gonz'alez-Rodríguez I (2022) Multi-objective enhanced memetic algorithm for green job shop scheduling with uncertain times. Swarm Evol. Comput. 68:101016. https://doi.org/10.1016/j.swevo.2021.101016

Afşar S, Puente J, Palacios JJ, González-Rodríguez I, Vela CR (2024) A genetic approach to green flexible job shop problem under uncertainty. In: International work-conference on the interplay between natural and artificial computation, pp 183–192. Springer

Ahn S, Lee S, Bahn H (2017) A smart elevator scheduler that considers dynamic changes of energy cost and user traffic. Integr. Comput. Aid. Eng. 24:187–202. https://doi.org/10.3233/ICA-170539

Alvarez-Meaza I, Zarrabeitia-Bilbao E, Rio-Belver R-M, Garechana-Anacabe G (2021) Green scheduling to achieve green manufacturing: pursuing a research agenda by mapping science. Technol. Soc. 67:101758. https://doi.org/10.1016/j.techsoc.2021.101758

Alvarez-Valdes R, Fuertes A, Tamarit JM, Giménez G, Ramos R (2005) A heuristic to schedule flexible job-shop in a glass factory. Eur. J. Oper. Res. 165(2):525–534. https://doi.org/10.1016/j.ejor.2004.04.020

Barredo P, Puente J (2023) Precise makespan optimization via hybrid genetic algorithm for scientific workflow scheduling problem. Nat. Comput. 22(4):615–630

Burke EK, Newall JP, Weare RF (1998) Initialization strategies and diversity in evolutionary timetabling. Evol. Comput. 6(1):81–103

Chaudhry IA, Khan AA (2016) A research survey: review of flexible job shop scheduling techniques. Int. Trans. Oper. Res. 23(3):551–591. https://doi.org/10.1111/itor.12199

Chen JC, Wu C-C, Chen C-W, Chen K-H (2012) Flexible job shop scheduling with parallel machines using genetic algorithm and grouping genetic algorithm. Expert Syst. Appl. 39(11):10016–10021. https://doi.org/10.1016/j.eswa.2012.01.211

Coelho P, Pinto A, Moniz S, Silva CA (2021) Thirty years of flexible job-shop scheduling: a bibliometric study. Procedia Comput. Sci. 180(C):787–796. https://doi.org/10.1016/j.procs.2021.01.329

Lenstra JK, Rinnooy Kan AHG, Brucker P (1977) Complexity of machine scheduling problems. Ann. Discrete Math. 1:343–362

Dauzère-Pérès S, Paulli J (1997) An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using Tabu search. Ann. Oper. Res. 70(3):281–306

Dauzère-Pérès S, Ding J, Shen L, Tamssaouet K (2024) The flexible job shop scheduling problem: a review. Eur. J. Oper. Res. 314(2):409–432. https://doi.org/10.1016/j.ejor.2023.05.017

Díaz H, Palacios JJ, Díaz I, Vela CR, González-Rodríguez I (2022) Robust schedules for tardiness optimization in job shop with interval uncertainty. Logic J IGPL. https://doi.org/10.1093/jigpal/jzac016

Díaz H, Palacios JJ, González-Rodríguez I, Vela CR (2023) Fast elitist ABC for makespan optimisation in interval JSP. Nat. Comput. 22(4):645–657. https://doi.org/10.1007/s11047-023-09953-2

Díaz H, Palacios JJ, González-Rodríguez I, Vela CR (2023) An elitist seasonal artificial bee colony algorithm for the interval job shop.

Integr. Comput. Aid. Eng. 30(3):223–242. https://doi.org/10.3233/ICA-230705

Ehrgott M (2005) Multicriteria optimization, 2nd edn. Springer

Fattahi P, Mehrabad MS, Jolai F (2007) Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. J. Intell. Manuf. 18:331–342. https://doi.org/10.1007/s10845-007-0026-8

Fazel Zarandi MH, Sadat Asl AA, Sotudian S, Castillo O (2020) A state of the art review of intelligent scheduling. Artif. Intell. Rev. 53(1):501–593. https://doi.org/10.1007/s10462-018-9667-6

Gahm C, Denz F, Dirr M, Tuma A (2016) Energy-efficient scheduling in manufacturing companies: a review and research framework. Eur. J. Oper. Res. 248(3):744–757. https://doi.org/10.1016/j.ejor.2015.07.017

García Gómez P, Vela CR, González-Rodríguez I (2023) Neighbourhood search for energy minimisation in flexible job shops under fuzziness. Nat. Comput. 22(4):685–704. https://doi.org/10.1007/s11047-023-09967-w

Ghafari R, Kabutarkhani FH, Mansouri N (2022) Task scheduling algorithms for energy optimization in cloud environment: a comprehensive review. Clust. Comput. 25(2):1035–1093. https://doi.org/10.1007/s10586-021-03512-z

González MA, Oddi A, Rasconi R (2017) Multi-objective optimization in a job shop with energy costs through hybrid evolutionary techniques. In: Proceedings of the 27th international conference on automated planning and scheduling (ICAPS-2017), pp 140–148

Guggeri EM, Ham C, Silveyra P, Rossit DA, Piñeyr P (2023) Goal programming and multi-criteria methods in remanufacturing and reverse logistics: systematic literature review and survey. Comput. Ind. Eng. 185:109587. https://doi.org/10.1016/j.cie.2023.109587

Guo W, Xu P, Zhao Z, Wang L, Zhu L, Wu Q (2020) Scheduling for airport baggage transport vehicles based on diversity enhancement genetic algorithm. Nat Comput. https://doi.org/10.1007/s11047-018-9703-0

Iannino V, Colla V, Maddaloni A, Brandenburger J, Rajabi A, Wolff A, Ordieres J, Gutierrez M, Sirovnik E, Mueller D, Schirm C (2022) A hybrid approach for improving the flexibility of production scheduling in flat steel industry. Integr. Comput. Aid. Eng. 29(4):367–387. https://doi.org/10.3233/ICA-220685

IBM: IBM CPLEX Optimizer (2020). https://www.ibm.com/analytics/cplex-optimizer

IEA: Electricity market report: Analysis and forecast to 2026. Technical report, International Energy Agency (2024). Accessed 22 Oct 2024. https://iea.blob.core.windows.net/assets/6b2fd954-2017-408e-bf08-952fdd62118a/Electricity2024-Analysisandforecastto2026.pdf

Jin X (2024) Application of metaheuristic algorithm in intelligent logistics scheduling and environmental sustainability. Intell. Decis. Technol. 18(3):1727–1740. https://doi.org/10.3233/IDT-240280

Karmakar S, Bhunia AK (2012) A comparative study of different order relations of intervals. Reliab. Comput. 16:38–72

Laborie P, Rogerie J, Shaw P, Vilím P (2018) IBM ILOG CP optimizer for scheduling. Constraints 23(2):210–250. https://doi.org/10.1007/s10601-018-9281-x

Lei D (2012) Co-evolutionary genetic algorithm for fuzzy flexible job shop scheduling. Appl. Soft Comput. 12:2237–2245. https://doi.org/10.1016/j.asoc.2012.03.025

Lei D (2013) Multi-objective artificial bee colony for interval job shop scheduling with flexible maintenance. Int. J. Adv. Manuf. Technol. 66:1835–1843. https://doi.org/10.1007/s00170-012-4463-y

Lei D, Guo X (2015) An effective neighborhood search for scheduling in dual-resource constrained interval job shop with environmental objective. Int. J. Prod. Econ. 159:296–303. https://doi.org/10.1016/j.ijpe.2014.07.026

Li M, Wang G-G (2022) A review of green shop scheduling problem. Inf. Sci. 589:478–496. https://doi.org/10.1016/j.ins.2021.12.122

Li XX, Li WD, Cai XT, He FZ (2015) A hybrid optimization approach for sustainable process planning and scheduling. Integr. Comput. Aid. Eng. 22(4):311–326

Li K, Chen J, Fu H, Jia Z, Fu W (2019) Uniform parallel machine scheduling with fuzzy processing times under resource consumption constraint. Appl. Soft Comput. 82:105585. https://doi.org/10.1016/j.asoc.2019.105585

Lunardi WT, Birgin EG, Laborie P, Ronconi DP, Voos H (2020) Mixed integer linear programming and constraint programming models for the online printing shop scheduling problem. Comput Oper Res. https://doi.org/10.1016/j.cor.2020.105020

Mencía C, Sierra MR, Mencía R, Varela R (2019) Evolutionary one-machine scheduling in the context of electric vehicles charging. Integr. Comput. Aid. Eng. 26(1):49–63. https://doi.org/10.3233/ICA-180582

Öztop H, Tasgetiren MF, Kandiller L, Eliiyi DT, Gao L (2020) Ensemble of metaheuristics for energy-efficient hybrid flowshops: makespan versus total energy consumption. Swarm Evol. Comput. 54:100660. https://doi.org/10.1016/j.swevo.2020.100660

Palacios JJ, González MA, Vela CR, González-Rodríguez I, Puente J (2015) Genetic Tabu search for the fuzzy flexible job shop problem. Comput. Oper. Res. 54:74–89. https://doi.org/10.1016/j.cor.2014.08.023

Para J, Del Ser J, Nebro AJ (2022) Energy-aware multi-objective job shop scheduling optimization with metaheuristics in manufacturing industries: a critical survey, results, and perspectives. Appl Sci. https://doi.org/10.3390/app12031491

Pinedo ML (2022) Scheduling. Theory, algorithms, and systems, 6th edn. Springer. https://doi.org/10.1007/978-3-031-05921-6

Puente J, Vela CR, Prieto C, Varela R (2003) Hybridizing a genetic algorithm with local search and heuristic seeding. In: Artificial neural nets problem solving methods (IWANN 2003), vol LNCS 2687, pp 329–336. Springer

Rossi F, Van Beek P, Walsh T (2008) Constraint programming. Found. Artif. Intell. 3:181–211

Torres N, Greivel G, Betz J, Moreno E, Newman A, Thomas B (2024) Optimizing steel coil production schedules under continuous casting and hot rolling. Eur. J. Oper. Res. 314(2):496–508. https://doi.org/10.1016/j.ejor.2023.10.005

Xiang W (2017) A multi-objective ACO for operating room scheduling optimization. Nat. Comput. 16:607–617

Xie J, Gao L, Peng K, Li X, Li H (2019) Review on flexible job shop scheduling. IET Collab. Intell. Manuf. 1(3):67–77. https://doi.org/10.1049/iet-cim.2018.0009

Xiong H, Shi S, Ren D, Hu J (2022) A survey of job shop scheduling problem: the types and models. Comput. Oper. Res. 142:105731. https://doi.org/10.1016/j.cor.2022.105731