Energy-Aware Double-Flexible Job Shop Scheduling with Machine Modes and Setup Times: A Real-World Industrial Case Study using Constraint Programming

Francesco Zuccato[1], Patrick Rodler[1], Gerhard Friedrich[1], Konstantin Schekotihin[1] and Richard Comploi-Taupe[2]

[1]University of Klagenfurt, Klagenfurt, Austria
[2]Siemens AG Österreich, Vienna, Austria

Abstract

Sustainable manufacturing requires energy-efficient scheduling, especially in metalworking, where machines like bandsaws consume significant power. Based on a real-world use case from an Austrian steel-cutting company, we extend the Energy-Aware Double-Flexible Job-Shop Scheduling Problem (E-DFJSP)—which already comprises machine and worker flexibility—with machine modes, as well as setup and transport operations. To tackle this problem, we propose a Constraint Programming (CP) model, implemented using the state-of-the-art IBM CP Optimizer (CPO), to minimize job tardiness, energy consumption, and makespan.

We evaluate our approach on two datasets representing present and future production scenarios, each with up to 500 jobs. While CPO fails to find feasible solutions for several large instances in the less flexible scenario, it successfully solves all instances in the more flexible one, indicating that higher resource flexibility improves search performance. In terms of solution quality, CPO demonstrates stronger scalability in makespan than in tardiness minimization, with the addition of machines and workers further reducing the optimal makespan and easing problem-solving. Finally, we identified a bug in CPO affecting $staticLex$ (fixed-priority multi-objective minimization) when combined with tolerance-based early stopping; to avoid it, we optimized each goal separately.

Keywords

Real-World Industrial Use Case, Flexible Job Shop Scheduling, Constraint Programming, Energy-Aware Scheduling, Multi-Objective Decision-Making, Metalworking Industries, IBM CP Optimizer, Double Flexible, Dual-Resource

1. Introduction

With growing global awareness of sustainability and environmental concerns, energy minimization has emerged as a critical objective across various sectors [1]. Metalworking operations, including sawing, grinding, and milling, are known for their significant energy consumption [2]. For example, a typical sawing machine requires 8.4 MWh per year, and a large metalworking facility may house 2500 machines, which, combined, would consume around 21 GWh per year, a value in the same order as 4,750 average Austrian households per year. In this work, we consider bandsaws, whose operations are determined by two key parameters—*machine mode*: feed rate V_f and blade speed V_c. Both parameters influence operation time, energy use, tool wear, and product quality. Selecting optimal parameters is complex, and most industries still use fixed values from lookup tables based on materials and dimensions [3, 4]. This leaves significant optimization potential, as the choice of a machine mode can affect the solution quality: slower modes may cause delays, while faster modes may increase energy cost and tool wear.

To address this trade-off, we formulate and solve a scheduling problem, based on a real-world scenario emerging from the Austrian metal-cutting industry, with the goals of minimizing job tardiness, energy consumption, and overall makespan. We assume that, for each operation, multiple machines may be eligible, and for each machine, various alternative machine modes are available, each having a corresponding duration and consumed energy. In addition, since metal pieces must be manually loaded

ECAI Workshop on AI-based Planning for Complex Real-World Applications (CAIPI'25)
francesco.zuccato@aau.at (F. Zuccato); Patrick.Rodler@aau.at (P. Rodler); Gerhard.Friedrich@aau.at (G. Friedrich); Konstantin.Schekotihin@aau.at (K. Schekotihin); richard.taupe@siemens.com (R. Comploi-Taupe)
0009-0004-1985-6914 (F. Zuccato); 0000-0001-8178-4692 (P. Rodler); 0000-0002-1992-4049 (G. Friedrich); 0000-0002-0286-0958 (K. Schekotihin); 0000-0001-7639-1616 (R. Comploi-Taupe)

(unloaded) by workers on (from) the machines, we account for the availability of the workers as well. This is a challenging task since the Job-Shop Scheduling Problem (JSP) is *per se* an NP-hard problem [5], on top of which we incorporate the choice of the machine, of the machine mode, and of the worker in order to minimize the mentioned objectives. As detailed in Sec. 2, and to the best of the authors' knowledge, no work has yet undertaken the Energy-Aware Double-Flexible Job-Shop Scheduling Problem (E-DFJSP) [6] when extended with either machine modes, setup operations, or transport operations. Hence, this work is a first step towards closing a gap in the research literature concerning the integrated consideration of time and resource efficiency in a flexible production scheduling context.

The common methods used to tackle scheduling problems are Mixed Integer Programming (MIP), Answer Set Programming (ASP), and Constraint Programming (CP). Recent studies, e.g., [7], have shown that the MIP approach faces scalability issues when applied to scheduling problems. Other works, such as [8, 9], relied on `clingo-DL` [10], an extension of ASP, to solve large-scale scheduling problems. Yet, despite its effectiveness, ASP is designed for discrete problems, while in many scheduling applications, some variables and goals, such as the energy consumption, are intrinsically real numbers.

Therefore, we propose a novel CP model to solve the E-DFJSP problem. We employ IBM CP Optimizer (CPO) [11] as a solver, since CPO is one of the most popular state-of-the-art CP-based tools for modeling and solving scheduling problems [12, 13, 14, 15, 16, 17]. To evaluate the model, we introduce two datasets that reflect the structure of the E-DFJSP for two different use cases within the steel-cutting context of our industrial partner. Each dataset comprises between 50 and 500 jobs. The upper bound is used to assess the scalability of our approach and is more than twice the number of cutting jobs required in the actual use case. In the ***P**resent* use case dataset P, workers are preassigned to machines and jobs. In contrast, the ***F**uture* use case F represents a prospectively planned, more general production scenario where these restrictions are relaxed, giving rise to a general E-DFJSP problem.

As this study introduces a novel problem, there are currently no established state-of-the-art methods available for direct comparison. The model is evaluated by comparing its performance on both datasets within a timeout of 10 minutes under two different tolerance levels, 0% and 10%, representing relative deviations from the global minimum that are still considered optimal. Our most significant finding is that a valid schedule is always obtained for dataset F. Moreover, in more than half of the runs on F, tardiness is optimized, and in some cases, energy is as well. By contrast, although dataset P is a special case of F with lower flexibility, CPO does not always find a solution for its instances, particularly for the 500-job cases. Nevertheless, in most of the 50-job instances of P, at least one goal is optimized. Finally, while makespan is rarely fully optimized in either dataset, its values remain consistently reasonable.

The remainder of the paper is organized as follows. A literature review is presented in Sec. 2, the basics of JSP are given in Sec. 3, the industrial use case is described in Sec. 4, and the CP model is proposed in Sec. 5. Our evaluation is illustrated in Sec. 6: the datasets are described in Sec. 6.1, the experiments' design is given in Sec. 6.2, and the results are discussed in Sec. 6.3. Finally, Sec. 7 concludes.

2. Literature Review

Flexible Job Shop Scheduling (FJSP) has been widely studied over the past 50 years [18]. In recent years, multiple works have tackled the Energy-Aware FJSP (e.g., [19, 20]), mostly by relying on inexact methods, and in particular on genetic algorithms (GA), as they proved to be effective in many domains. The following works share the same objectives—the energy consumption and the makespan—and are single-resource, i.e., each operation requires only one resource, such as a machine, a crane, etc. In [21], a genetic algorithm and a CP encoding were proposed for addressing a JSP problem, i.e., not-flexible, but with controllable machine modes. [19] introduced a genetic algorithm to solve the FJSP with multiple machine modes, with the additional goal of minimizing the noise emission caused by milling and drilling machines. [20] proposed a hybrid method for solving an FJSP with transportation constraints, by including machines and cranes flexibility. [22] suggested a hybrid method for tackling a FJSP with setup and transport operations, where the means of transport were autonomous vehicles. [23] proposed a metaheuristic algorithm for solving an FJSP with transport time and sequence-dependent setup times,

but without an explicit means of transport. Using a cooperative evolutionary algorithm, [24] tackled an FJSP with machines, cranes, and sequence-dependent setup times. Two studies considered the Energy-Aware Multi-Resource FJSP. In [6], the E-DFJSP was solved, as both workers and machines were included, and a hybrid algorithm was employed to minimize makespan, the maximum total labor cost, and multiple green objectives. Similarly, [25] discussed a Multi-Resource FJSP with both machines and workers. Yet, neither of them considered transport or setup operations, or machine modes.

In recent years, some works based on Constraint Programming (CP) have emerged. Notably, the following studies minimize energy's Time-Of-Use (TOU) rather than the pure energy consumption. The TOU weights the energy consumption by its average cost over a fixed period of time, e.g., an hour. In [26], a model for solving the FJSP with TOU-optimization was proposed. Similarly, [13] studied the FJSP with TOU and preventive maintenance of the machines. Also, in [16], a comprehensive model was introduced, which accounts for real-time pricing, power peak demand, renewable energy sources, and carbon emissions, as well as setup times and machine availability. For readers more deeply interested in energy-aware scheduling, [16] provides an extensive introduction, and [27] a noteworthy review.

3. Basics

The Flexible Job Shop Scheduling Problem (FJSP) [28] is defined by a set of *machines* (or more generally of *resources*) $M = \{M_1, ..., M_Q\}$, and a set of *jobs* $J = \{J_1, ..., J_N\}$. Each job $j \in J$ consists of a sequence of N_j *operations* $[o_{j1}, ..., o_{jN_j}]$. The set of operations is defined as $O = \{o_{ji} : 1 \leq j \leq N, 1 \leq i \leq N_j\}$. Each operation $o_{ji} \in O$ has an associated processing time pt_{ji} and a set of eligible machines $EM_{ji} \subseteq M$ able to process o_{ji}. It is typically assumed that *(i)* each machine can process up to one operation at a time; *(ii)* any job can be processed independently from the others; and *(iii)* the execution of an operation cannot be interrupted (no *preemption*). For every operation o_{ji} we denote by s_{ji} its assigned start time; by $c_{ji} = s_{ji} + pt_{ji}$ its completion time; by $o_{j(i-1)}$ for $i > 1$ its *predecessor* operation; and by $o_{j(i+1)}$ for $i < N_j$ its *successor* operation. An assignment of each operation to a machine and a start time that does not violate any of the following constraints is called a *schedule*: *(a)* each operation must end before its successor may start, and *(b)* each operation must be assigned to exactly one of its eligible machines.

To broaden the applicability of the FJSP, many extensions have been proposed. In the *Multi-Resource* FJSP (MR-FJSP) [9], each operation can require multiple resources of the same or of different types, such as a machine and a human operator. Another extension is the so-called FJSP with *variable machine speeds*[1], where the speed at which the machine carries out the operation must be chosen from a set of possible machine modes, leading to a *dynamic* duration of operations [19]. Usually, machine modes are not directly treated as a continuous decision variable, but are assumed to have a set of valid values given *a priori*. Also, different modes may be associated with different costs, quantifying, e.g., the *energy consumption* of, or the *caused wear* on the machine. Additional extensions may consider *transport* times, when operations o_{ji} and $o_{j(i+1)}$ are assigned to two different machines, and/or *setup* times, which may be required before executing an operation. In most cases, the setup time solely depends on the operation and on the machine, i.e., it can be expressed for operation o_{ji} on machine m as st_{jim} and thus can be assumed to be directly expressed by means of the processing time pt_{jim}.

4. The Industrial Use Case

The real-world industrial use case considered in this work originates from an Austrian steel-cutting facility, where each job comprises one or more cutting operations that segment large raw pieces into smaller ones meeting specific customer requirements. The factory operates several cutting machines, each equipped with a bandsaw. Jobs are assigned to workers, who in turn are assigned to machines.

A high-level description of a job is the following. Each job begins with the worker loading the input piece on the machine, setting the appropriate parameters, and starting the cutting operation. Once the

[1]Different works may refer to it with different namings, such as *controllable* in place of *variable*, and similarly *machining speeds*, *processing speeds*, or simply *modes* instead of *machine speeds*.

cut is finished, the operator has to unload the cut pieces. These can be either ready for delivery or must be cut again. In the latter case, whenever a different machine is required to perform the next cut, the time necessary for transporting the piece between the machines must be considered.

This process repeats daily, with hundreds of jobs, tens of machines, and tens of workers. Coordinating and planning the resources and operations in such a scenario constitutes a complex task to be addressed by a manufacturing company. As outlined in the introduction, the integration of multiple resource types, e.g., machines and workers, with setup and transport times has not yet been addressed within the context of energy-aware FJSP. This combination, however, is typical in various industrial contexts.

The industrial use case we address in this work is precisely characterized by the following assumptions:

1. All resources, machines M and workers W, are available for the entire scheduling horizon, i.e., tool wear, machine maintenance, or workers' shifts are not considered.
2. All jobs are released before the start of the scheduling horizon.
3. Each job $j \in J$ is composed of a sequence of *tasks*, where each task is a sequence of three operations—load, cut, and unload—associated with one and the same cut. Thus, job j has form $(\mathsf{load}_1, \mathsf{cut}_1, \mathsf{unload}_1, \ldots, \mathsf{load}_c, \mathsf{cut}_c, \mathsf{unload}_c)$ where c is the number of *tasks* and $N_j = 3 * c$.
4. The cut operations represent the $processing$ operations and require exactly one machine.
5. The $setup$ operations, i.e., load and unload, require exactly one machine and exactly one worker.
6. Each job $j \in J$ is preassigned to one, and only one, worker $w \in W$.
7. Each machine $m \in M$ is preassigned to one, and only one, worker $w \in W$. The set of machines assigned to worker w is referred to as M_w.
8. For every operation $o_{ji} \in O$ a non-empty set of eligible machines EM_{ji} is given. Each eligible machine $m \in EM_{ji} \subseteq M_w$ is preassigned to the same worker w, to whom job $j \in J$ is assigned.
9. For each processing operation $o_{ji} \in O$ and for each eligible machine $m \in EM_{ji}$, a non-empty set of machine modes $MD_{ji,m}$ is given, each allowing to accomplish the operation o_{ji} on machine m and implying a specific duration and energy consumption.
10. The times for load and unload depend only on the weight of the material that is cut.
11. The $transport$ time depends only on the weight of the material to be moved, as each machine set M_w is located closely enough that inter-machine distances can be considered negligible.
12. Any sequence of operations (1) unload, (2) $transport$—only if the subsequent operation is assigned to a different machine—and (3) load for one and the same job must be executed in immediate succession without interruptions.

Remark 1. Due to Assumptions 6 and 7, the assignment of a unique worker to every job and machine essentially leads to a partitioning of both the jobs and the machines. This has two implications. First, workers are not a flexible resource, and so the described problem is not a Double-Flexible JSP. Yet, the problem is still a Multi-Resource JSP, as setup operations must be assigned to a worker and a machine at a specific time point. Second, and more importantly, the overall scheduling problem can be decomposed into separate subproblems, one per worker. Therefore, given an input instance, one could basically preprocess it, create $|W|$ smaller and simpler subproblems, and solve each of them independently to obtain an overall schedule. However, in our evaluations (Sec. 6), to simulate a practical scenario where an a-priori problem analysis and decomposition is not always feasible, we solved each problem instance as-is, without applying any preprocessing. Moreover, to nevertheless obtain a more complete picture, we generate and analyze a second more general dataset, where Assumptions 6 and 7 are dropped and problem decompositions based on a partitioning of jobs and machines are no longer possible.

5. The Proposed Constraint Programming Model

The proposed model, based on Constraint Programming (CP), encodes and enables solving the Energy-Aware Double-Flexible JSP (E-DFJSP) [6], a variant of the JSP that incorporates multi-resource (workers and machines), routing flexibility, and energy minimization. Beyond that, we also consider the following extensions: controllable machine modes, sequence-independent setup operations, and transport operations. Note that the E-DFJSP comprises, in particular, the use case problem described in Sec. 4.

5.1. Model Notation

The model is implemented in IBM CPLEX Studio, and is solved by IBM CP Optimizer (CPO) [11]. CPO is probably the most popular CP-based tool for modeling and solving JSP problems [12, 13, 14, 15, 16, 17]. In particular, Da Col and Teppan [14] showed that CPO is the state-of-the-art tool for solving scheduling problems, as up to one thousand jobs could be successfully scheduled. Due to the absence of a standard notation for defining CP models, we employ the Optimization Programming Language (OPL). We summarize in Table 1 OPL's built-in functions and keywords used in this work.

Table 1
OPL functions and keywords available in CPO. Adapted from [16].

Function / Keyword	Description
$[\text{dvar}]\ \text{interval}\ x\ [\text{optional}]$	Defines an interval variable x, represented as a tuple $\langle start, end, duration \rangle$. If optional, it may not be present, i.e., the tuple could be undefined. If dvar then it is a decisional interval variable.
$\text{dvar sequence}\ s\ \text{in}\ X$	Defines a sequence s from a set of interval variables $X = \{x_1, x_2, ...\}$ to be ordered, which typically represents the queue of operations of a resource.
$x\ \text{in}\ l..r$	Limits the domain of interval var x to $[l, r]$ at declaration. The most classic use is 0..Horizon, where Horizon is a sufficiently large integer.
$x\ \text{size}\ s$	Sets the duration of interval var x to a fixed duration $s \in \mathbb{N}$ at declaration. Suitable also for optional interval variables, where, in such a case, each option has a corresponding fixed duration.
startOf(x)	Returns the start of interval x if present, otherwise an absent value.
endOf(x)	Returns the end of interval x if present, otherwise an absent value.
sizeOf(x)	Returns the duration of interval variable x. Useful to dynamically constrain the duration of x, depending on the value of some decision variables.
alternative(x, X, k)	Represents a constraint on an interval variable x and a set of optional intervals $X = \{x_1, x_2, \dots\}$ to ensure that *(i)* exactly k (1 by default) of the intervals in X are present if x is present, and *(ii)* both x and the k present variables in X share the same interval $\langle start, end, duration \rangle$.
presenceOf(x)	Boolean function indicating whether interval variable x is present. Useful to dynamically check which optional interval variable $x \in X$ is chosen.
endBeforeStart$(x, succ(x))$	Represents a constraint between interval variables x and its successor $succ(x)$, ensuring the successor cannot start until x ends.
startAtEnd$(succ(x), x)$	Represents a *no-wait* constraint between interval variables x and its successor $succ(x)$, ensuring the successor starts as soon as x ends.
noOverlap(s)	Ensures that all present variables $x_1, x_2, ...$ in sequence s do not overlap.
prev(s, x, y)	Ensures that if the interval variables $x, y \in s$ are present, then x is the predecessor of y in sequence s.
staticLex$(g_1, g_2, \dots)$	Enables a priority ordering for multicriteria optimization.

5.2. Parameters and Variables

As previously introduced, we recall that J, O, M, W stand for the sets of jobs, operations, machines, and workers, respectively. There are three operation types $OT = \{\mathsf{load}, \mathsf{cut}, \mathsf{unload}\}$, where load and unload are setup operations, while cut is a processing operation. We denote the set of setup and processing operations by SO and PO, respectively. The predecessor and the successor of each processing operation are setup operations (cf. Assumption 3 in Sec. 4). The set of resource types $RT = \{\mathsf{mch}, \mathsf{wrk}\}$ enables distinguishing between machines (mch) and workers (wrk). The set of resources R is defined by the set of all tuples $\langle m, \mathsf{mch} \rangle$, $\langle w, \mathsf{wrk} \rangle$ where $m \in M$ is a machine and $w \in W$ is a worker. To improve the readability, we define two labels: $type_o : O \to OT$, which, given an operation $o \in O$, outputs the corresponding operation type; and $type_r : R \to RT$ which, given a resource $r \in R$, outputs its resource type. For any operation $o \in O$, we define by R_o its set of eligible resources. For each processing operation $po \in PO$ and eligible machine $r = \langle m, \mathsf{mch} \rangle \in R_{po}$, a

predefined set of eligible machine modes $MD_{po,r}$ is available.[2] We treat machine modes as a finite set of possible feed rate (V_f) and blade speed (V_c) combinations, i.e., a set of tuples of the form $\langle V_f, V_c \rangle$.[3] Each machine mode $md \in MD_{po,r}$ is associated with duration $pt_{po,r,md}$ and processing energy $pe_{po,r,md}$. Since setup and transport times are only weight-dependent (cf. Assumptions 10 and 11), they can be represented as st_{so} and tt_{so}, respectively, where $so \in SO$ is a setup operation. Decision variables are highlighted in bold: $\boldsymbol{T}_o$ is the interval variable of every operation $o \in O$. Two optional interval variables are defined, which enable representing possible alternative ways to execute an operation, e.g., using a different resource. In particular, we define $\boldsymbol{PT}_{po,r,md}$ for every processing operation $po \in PO$, eligible resource $r \in R_{po}$, and available machine mode $md \in MD_{po,r}$. Similarly, we introduce the optional interval variable $\boldsymbol{AT}_{o,r}$ for all operations $o \in O$ and eligible resources $r \in R_o$. Both $\boldsymbol{PT}_{po,r,md}$ and $\boldsymbol{AT}_{o,r}$ are synchronized with $\boldsymbol{T}_o$. We also introduce, for each resource r, a sequence decision variable $\boldsymbol{s}_r$, which contains all the operations for which r is eligible. We also define a *derived* decision variable $\boldsymbol{C}_j$, which represents the completion time of each job $j \in J$. Table 2 summarizes the notation and the description of all the defined sets, parameters, and variables.

Table 2
Notations and definitions.

Notation	Definition
Indices and Sets	
J	Set of jobs $j \in J = \{j_1, \ldots, j_N\}$
O	Set of operations $o \in O = \{o_{ji} \mid 1 \leq j \leq N, 1 \leq i \leq N_j\} = PO \cup SO$
PO	Set of processing operations $po \in PO \subset O = \{o_{ji} \mid o_{ji} \in O, i \equiv 2 \pmod 3\}$
SO	Set of setup operations $so \in SO \subset O = \{o_{ji} \mid o_{ji} \in O, i \not\equiv 2 \pmod 3\}$
OT	Set of operation types $OT = \{\mathsf{load}, \mathsf{cut}, \mathsf{unload}\}$
RT	Set of resource types $rt \in RT = \{\mathsf{mch}, \mathsf{wrk}\}$
M	Set of machines $m \in M$
W	Set of workers $w \in W$
R	Set of resources $r \in R = \{\langle m, \mathsf{mch}\rangle : m \in M\} \cup \{\langle w, \mathsf{wrk}\rangle : w \in W\}$
R_o	Set of eligible resources $r \in R_o$ for operation $o \in O$
$MD_{po,r}$	Set of modes $md \in MD_{po,r}$ for processing operation $po \in PO$ on resource $r \in R_{po}$
Parameters	
$pt_{po,r,md} \in \mathbb{N}^+$	Processing time of operation $po \in PO$ on resource $r \in R_{po}$ with mode $md \in MD_{po,r}$
$pe_{po,r,md} \in \mathbb{R}^+$	Processing energy of operation $po \in PO$ on resource $r \in R_{po}$ with mode $md \in MD_{po,r}$
$st_{so} \in \mathbb{N}^+$	Setup time of operation $so \in SO$
$tt_{so} \in \mathbb{N}^+$	Transport time of operation $so \in SO$
$d_j \in \mathbb{N}^+$	Deadline of job $j \in J$
$type_o$	Type of operation $o = o_{ji} \in O$: equals load if $i \equiv 1 \pmod 3$, cut if $i \equiv 2 \pmod 3$, and unload if $i \equiv 0 \pmod 3$
$type_r$	Type of resource $r = \langle id, rt\rangle \in R$: equals $rt \in RT$.
Decision Variables	
$\boldsymbol{T}_o$	Interval representing start and end time of operation $o \in O$
$\boldsymbol{PT}_{po,r,md}$	Optional interval for operation $po \in PO$ on resource $r \in R_{po}$ with mode $md \in MD_{po,r}$
$\boldsymbol{AT}_{o,r}$	Optional interval for operation $o \in O$ using resource $r \in R_o$
$\boldsymbol{s}_r$	Sequence of optional intervals for resource r: $\forall o \in O\,[r \in R_o \iff \boldsymbol{AT}_{o,r} \in \boldsymbol{s}_r]$.
$\boldsymbol{C}_j$	Completion time of each job $j \in J$, defined as $\boldsymbol{C}_j = \mathit{endOf}(\boldsymbol{T}_{jN_j})$
Objectives	
$Tard$	Tardiness
PE	Total Processing Energy
C_{max}	Makespan

[2] Machine modes can be defined, e.g., based on domain expertise, engineering knowledge, or machine learning models.
[3] By using a discrete set of tuples to represent all relevant modes, no real-valued decision variables are required in the model. Note that CPO does not support real-valued decision variables.

5.3. Constraints

Constraints (1), (2), and (3) are used to assign resources to operations. Each processing operation po must be assigned to exactly one eligible machine $r \in R_{po}$ and must be executed according to the chosen machine mode $md \in MD_{po,r}$. Note, if $r \in R_{po}$ then $type_r = \mathsf{mch}$.

$$\mathit{alternative}(\boldsymbol{T}_{po}, \{\boldsymbol{PT}_{po,r,md} \,:\, r \in R_{po}, md \in MD_{po,r}\}) \quad \forall po \in PO \tag{1}$$

Recall from Assumptions 4, 5 that *setup* operations require both a machine and a worker, while *processing* operations only require a machine. Constraint (2) defines the worker assignment and therefore applies only to setup operations. For the present dataset, in which workers are preassigned, the choice of the worker is trivial. Nevertheless, the constraint is essential to synchronize the optional variables $\boldsymbol{AT}_{o,r}$ with the main variable $\boldsymbol{T}_o$. Constraint (3) specifies the machine assignment.[4]

$$\mathit{alternative}(\boldsymbol{T}_{so}, \{\boldsymbol{AT}_{so,r} \,:\, r \in R_{so}, type_r = \mathsf{wrk}\}) \quad \forall so \in SO \tag{2}$$

$$\mathit{alternative}(\boldsymbol{T}_{o}, \{\boldsymbol{AT}_{o,r} \,:\, r \in R_{o}, type_r = \mathsf{mch}\}) \quad \forall o \in O \tag{3}$$

Constraint (4) links the choices in Constraints (1) and (3), as one, and only one, machine must be used for each *task*. Due to Constraint (1), the sum on the right-hand side is always ≤ 1. Moreover, we associate the value $true$ ($false$) of the predicate $\mathit{presenceOf}(\cdot)$ with 1 (0).[5]

$$\begin{gathered}\mathit{presenceOf}(\boldsymbol{AT}_{jk,r}) = \sum_{md \in MD_{ji,r}} \mathit{presenceOf}(\boldsymbol{PT}_{ji,r,md}) \\ \forall o_{ji} \in O \; \forall r \in R_{ji} : type_{ji} = \mathsf{cut}, k \in \{i-1, i, i+1\}\end{gathered} \tag{4}$$

Constraints (5) and (6) ensure the correct duration for the setup operation and the transport operation. The duration of a setup operation $so \in SO$ is st_{so}, but, whenever an unload operation o_{ji} and its subsequent load operation $o_{j(i+1)}$ are assigned to different machines, the transport time tt_{ji} must be included. To this end, Constraint (5) states that the duration of setup operation $so \in SO$ must be at least as long as st_{so}. Constraint (6), on the other hand, handles the machine-switch case, in which the duration of the unloading operation is equal to the setup time st_{so} plus the transport time tt_{so}.

$$\mathit{sizeOf}(\boldsymbol{T}_{so}) \geq st_{so} \quad \forall so \in SO \tag{5}$$

$$\begin{gathered}\mathit{presenceOf}(\boldsymbol{AT}_{ji,r_1}) \wedge \mathit{presenceOf}(\boldsymbol{AT}_{j(i+1),r_2}) \implies \mathit{sizeOf}(\boldsymbol{T}_{ji}) = st_{ji} + tt_{ji} \\ \forall o_{ji} \in O \; \forall r_1 \in R_{ji} \; \forall r_2 \in R_{j(i+1)} : i < N_j, type_{ji} = \mathsf{unload}, r_1 \neq r_2\end{gathered} \tag{6}$$

Constraints (7) and (8) encode the intra-job precedence of operations. In particular, Constraint (7) is the classic precedence constraint, which is defined only between the end of a processing operation and the start of the subsequent unloading operation. In the other cases, *no-wait* constraints are defined, i.e., stricter precedence constraints that also enforce that two operations are executed in immediate succession. *No-waits* are defined *(i)* between load and cut —as it would be nonsensical to wait before starting an operation once loading is complete, unless additional costs such as time-of-use or peak-power costs are considered—, and *(ii)* between unload and load (according to Assumption 12).

$$\mathit{endBeforeStart}(\boldsymbol{T}_{ji}, \boldsymbol{T}_{j(i+1)}) \quad \forall o_{ji} \in O : type_{ji} = \mathsf{cut} \tag{7}$$

$$\mathit{startAtEnd}(\boldsymbol{T}_{j(i+1)}, \boldsymbol{T}_{ji}) \quad \forall o_{ji} \in O : type_{ji} \neq \mathsf{cut} \tag{8}$$

Constraint (9) guarantees that the operations of the same *task* (load, cut, unload) occur consecutively on the same machine (cf. Assumption 3). Note that it is defined *only* if both $\boldsymbol{AT}_{ji,r}$, and $\boldsymbol{AT}_{j(i+1),r}$

[4] Constraint (3) is redundant, since Constraint (1) already selects the machine for the processing operations (and consequently for the setup operations). The extra cost—along with that of synchronizing such choices in Constraint (4)—keeps each sequence $\boldsymbol{s}_r$ linear in $|O|$, by linking $\boldsymbol{s}_r$ to $\boldsymbol{AT}_{o,r}$ instead of $\boldsymbol{PT}_{o,r,md}$.

[5] For readability, we slightly abuse notation and write x_{ji} instead of $x_{o_{ji}}$ for variables associated with operation $o_{ji} \in O$.

are present and thus appear on sequence $\boldsymbol{s}_r$, i.e., if both o_{ji}, $o_{j(i+1)}$ have been assigned to resource r. Constraint (10) imposes that, if worker w is assigned to operations o_{ji} (unload) and $o_{j(i+1)}$ (load), the next operation for w after o_{ji} must be $o_{j(i+1)}$. Note, this is *always* the case for the present dataset. Constraints (9)-(10) are weaker than Constraint (8), but can be beneficial in terms of performance.[6]

$$\mathit{prev}(\boldsymbol{s}_r, \boldsymbol{AT}_{ji,r}, \boldsymbol{AT}_{j(i+1),r}) \quad \forall o_{ji} \in O\ \forall r \in R_{ji} \cap R_{j(i+1)} : i < N_j, \mathit{type}_r = \mathsf{mch} \tag{9}$$

$$\mathit{prev}(\boldsymbol{s}_r, \boldsymbol{AT}_{ji,r}, \boldsymbol{AT}_{j(i+1),r}) \quad \forall o_{ji} \in O\ \forall r \in R_{ji} \cap R_{j(i+1)} : i < N_j, \mathit{type}_r = \mathsf{wrk}, \mathit{type}_{ji} = \mathsf{unload} \tag{10}$$

Constraint (11) ensures that the operations assigned to the same resource r do not overlap.[7]

$$\mathit{noOverlap}(\boldsymbol{s}_r) \quad \forall r \in R \tag{11}$$

5.4. Objective Function

The objective function is defined by three objectives, each to be minimized, with a fixed order of priority: *(i)* tardiness ($Tard$), *(ii)* energy (PE), and *(iii)* makespan (C_{max}).

$$\text{minimize} \ \ \mathit{staticLex}(Tard, PE, C_{max}) \tag{12}$$

Tardiness measures the cumulative lateness of all given jobs J, and we measured it in 8-hour shifts, i.e., $\mathsf{SHIFT\_LEN} = 480\,[min]$. We compute it as the sum of all non-negative differences between the shift $\lceil \boldsymbol{C}_j / \mathsf{SHIFT\_LEN} \rceil$ where a job j is completed and the shift d_j where the job is due, over all jobs $j \in J$:

$$Tard = \sum_{j \in J} \max\left(\lceil \boldsymbol{C}_j / \mathsf{SHIFT\_LEN} \rceil - d_j, 0\right) \tag{13}$$

The processing energy of a single operation is computed by taking the consumed energy of the chosen machine mode. Note that, due to Constraint (1), exactly one of the *presenceOf*$(\cdot)$ predicates per operation must be true. Then, the total energy is obtained by summing over all the processing operations:

$$PE = \sum_{po \in PO} \sum_{r \in R_{po}} \sum_{md \in MD_{po,r}} \mathit{presenceOf}(\boldsymbol{PT}_{po,r,md}) * pe_{po,r,md} \tag{14}$$

The makespan is the total duration of the entire schedule:

$$C_{max} = \max_{j \in J} \boldsymbol{C}_j = \max_{j \in J} \mathit{endOf}(\boldsymbol{T}_{jN_j}) \tag{15}$$

6. Evaluation

6.1. Dataset

The presented model is tested on two similar problems, which resemble the real-world scenario of a steel-cutting company described in Sec. 4. To preserve the confidentiality of our industry partner's data, the datasets were generated using a random sampling procedure based on the real use case. In particular, we distinguish between the present and the future use case scenarios, where the former reflects current assumptions while the latter anticipates slight changes planned by the considered steel-cutting company. More specifically, in the *present use case scenario*, each machine is assigned to exactly one worker, while in the *future use case scenario*, multiple workers are allowed to operate the same machine. The single-worker-per-machine assumption in the former scenario implies that the scheduling problem can be decomposed into a set of independent subproblems (cf. Remark 1), while this is not possible in the latter scenario. As a result, analyzing this more general case is particularly interesting from both theoretical and practical perspectives. On the one hand, the problem becomes more complex, as it corresponds to a Double-Flexible JSP; on the other hand, introducing additional worker flexibility may lead to reductions in makespan and tardiness.

[6] Constraint (9) is only partially redundant, as it also covers the case in which $\mathit{type}_{ji} = \mathsf{cut}$.

[7] Alternatively—if no *prev* constraints are defined—, one could define a *cumulFunction* and limit its value ≤ 1. IBM guide suggests that including both may improve performance, but we did not observe so, and thus we only used *noOverlap*.

Table 3
The structure of the two generated datasets. J, M, W are the number of jobs, machines, and workers. J/M and J/W are the ratios between the number of jobs and the number of machines and workers. $avg(O)$ is the average number of operations. $avg(H)$ is the average upper bound Horizon. F, P stand for future and present datasets. $d(M)$ stands for $degree(machine)$, i.e., the number of workers assigned to it, and analogously $d(W)$. $avg(Vars)$ and $avg(Const)$ are the average number of variables and constraints.

							$d(M)$		$d(W)$		$avg(Vars)$		$avg(Const)$	
						dataset	P	F	P	F	P	F	P	F
J	M	W	J/M	J/W	$avg(O)$	$avg(H)$								
50	3	1	16.7	50.0	307	2610	1		3		1569		2395	
	4	1	12.5	50.0	290	2252	1		4		1730		2603	
	8	2	6.2	25.0	296	1478	1	2	4	8	1777	1974	2679	2728
	9	3	5.6	16.7	278	1314	1	3	3	9	1477	1848	2223	2309
	15	5	3.3	10.0	299	688	1	3	3	9	1562	2140	2370	2491
	16	4	3.1	12.5	280	790	1	2	4	8	1727	2143	2574	2646
200	12	4	16.7	50.0	1167	3690	1	3	3	9	6019	7985	9101	9523
	16	4	12.5	50.0	1165	2983	1	2	4	8	7017	8725	10544	10856
	32	8	6.2	25.0	1157	1649	1	2	4	8	7050	9475	10532	10890
	36	12	5.6	16.7	1158	1225	1	3	3	9	6000	9162	9051	9569
	60	20	3.3	10.0	1192	664	1	3	3	9	6217	9701	9406	9977
	64	16	3.1	12.5	1199	969	1	2	4	8	7232	10026	10837	11216
500	30	10	16.7	50.0	2896	3511	1	3	3	9	15043	22471	22651	23895
	40	10	12.5	50.0	2917	2994	1	2	4	8	17927	24399	26913	27925
	80	20	6.2	25.0	3031	1886	1	2	4	8	18240	25619	27464	28488
	90	30	5.6	16.7	2927	1323	1	3	3	9	15240	24341	22985	24395
	150	50	3.3	10.0	2904	914	1	3	3	9	15135	24364	22757	24096
	160	40	3.1	12.5	2875	1162	1	2	4	8	17395	24584	25962	26913

6.1.1. The Present Use Case Dataset (Decomposable)

We generated a dataset for the present use case by means of the following parameters:

(i) three different *numbers of jobs*, $|J| \in \{50, 200, 500\}$, corresponding respectively to medium-, large-, and very large-scale instances;

(ii) two different *workers' flexibility degrees* as the number of machines assigned to each worker, $d(W) = (|M|/|W|) \in \{3, 4\}$, in accordance with the real use case;

(iii) the fixed *machines' flexibility degree*, i.e., the number of assigned workers to each machine, $d(M) = 1$ (cf. Assumption 7);[8]

(iv) for each setting of $d(W)$, three different *jobs-to-machines ratios*, $|J|/|M|$, i.e., "few" (up to 4), "some" (up to 10), and "many" (more than 10), to represent periods of varying system load; and

(v) the fixed number of *machine modes* per cut operation, $k = 2$.

The numbers of machines $|M|$ and workers $|W|$ then follow from the flexibility degree $d(W)$, the jobs-to-machines ratio $|J|/|M|$, and the number of jobs $|J|$. Note that if the number of jobs is fixed but the number of resources increases, both makespan and tardiness are expected to decrease—since fewer jobs are assigned to each resource—while the number of decision variables handled by the solver may increase. Therefore, by varying the jobs-to-machines ratio, we can observe how changes in resource availability influence both throughput and solver performance.

Once all the parameters are set, the remaining data is sampled as follows. The jobs' due dates are defined in terms of shifts (of 8 hours) and follow a gamma distribution $Gamma(\alpha = 2, \theta = 1)$.[9] Each job $j \in J$ is randomly assigned to one of the workers $w^\star \in W$, who in turn is assigned to a set of 3 or 4 machines $M_{w^\star}$, according to the workers' flexibility degree $d(W)$. For the processing operation o_{ji}, its set of eligible resources is a random non-empty subset of $M_{w^\star}$, i.e., $R_{ji} = \{\langle m, \mathsf{mch}\rangle : m \in M_{ji}\}$

[8]The machines-workers assignments can be represented as a bipartite graph $G = (M, W, E)$, where $(w, m) \in E$ if worker $w \in W$ is assigned to machine $m \in M$. Thus, the node-degree $d(w)$ represents the number of assigned machines to worker w. Since it is constant for each $w \in W$, we denote it by $d(W)$ for brevity. The same applies to the machines.

[9]Each sample is then rounded to the nearest integer ≥ 1. The resulting distribution is similar to a positively skewed bell-shape with expected value $\mathbb{E}[X] = \alpha\theta = 2$ and variance $Var(X) = \alpha\theta^2 = 2$. That is, 80 % of the jobs have a due shift ≤ 3.

where $M_{ji} \subseteq M_{w^\star}$. Basically, since $|M_w| = d(W)$ for any $w \in W$, the number of eligible machines per operation follows a discrete uniform distribution $\mathcal{U}\{1, \ldots, d(W)\}$. Load and unload operations also require a worker, namely $w^\star$, the one assigned to job j. Therefore, the set of eligible resources for the setup operations can be defined as $R_{j(i-1)} = R_{j(i+1)} = R_{ji} \cup \{\langle w^\star, \mathsf{wrk}\rangle\}$.

The single operations are generated as follows. The number of processing operations per job follows a shifted exponential distribution $Exp(\lambda = 1) + 1$.[10] Each processing operation has an associated physical piece to be cut. The dimensions of each input piece (height, width, and depth) before the cut operation are randomly sampled from three normal distributions $\mathcal{N}(\mu = 100, \sigma = 50), \mathcal{N}(450, 400), \mathcal{N}(200, 80)$ $[mm]$.[11] Then, out of these three values, two of them are randomly selected, one to be the cut depth and the other one to be the cut channel; these values represent (i) how long the cut will be and (ii) the width of the cut on a single blade-pass. The weight is then computed by multiplying the density constant of the steel, 7.85 $[g/cm^3]$, with the volume of the piece, derived by multiplying together height, width, and length, i.e., by assuming that all pieces are cuboids. The weight also enables deriving the setup time and the transport time. The k different machine modes (processing speeds) are then computed by first obtaining the manufacturers' suggested values[12] and then by applying k different random perturbations. The duration of the processing operations depends on the chosen machine mode, and, for a given machine mode, can be computed as the ratio between the cut depth $[mm]$ and the feed rate V_f $[mm/min]$. Finally, we estimate the processing energy of each option, and then randomly perturb the prediction. To do so, we assume having a black-box model that estimates the energy of a cut given the machine mode, the cut channel, and the cut depth. We also compute a worst-case upper bound Horizon for the entire schedule.

The procedure presented here is exemplary of how a scheduling problem in the manufacturing sector could look. Note that each combination of the parameters $\langle |J|, d(W), |J|/|M|\rangle$—which in total are $18 = 3 \times 2 \times 3$—leads to a unique combination $\langle |J|, |M|, |W|\rangle$. For each of the latter, we generate 3 different random instances. Thus, the number of instances for the present dataset is $3 \times 18 = 54$.

6.1.2. The Future Use Case Dataset (General)

In the future use case scenario, multiple workers can be assigned to each machine. To evaluate the impact of such a generalization, we keep the two datasets as identical as possible while preventing decomposability. We modify the machines-workers assignments so that at least 2 workers are assigned to each machine, i.e., the machines' flexibility degree $d(M) \geq 2$. Starting from one worker per machine, we achieve this by randomly assigning additional workers to each machine while ensuring that the expected number of machines assigned to each worker is realistic, i.e., $\mathbb{E}[d(W)] < 10$. Apart from that, all settings are equal to the present use case (Sec. 6.1.1), including the use of the same interval variable upper bound Horizon to ensure a fair comparison.

Table 3 details parameters and properties of the generated datasets, where F stands for the future (general) dataset and P for the present (decomposable). The columns mentioning $avg(\cdot)$ are derived by averaging the values from the 3 random instances. Note that dataset F has more variables (as there are more choices) and more constraints (as there are more *prev* constraints) than dataset P. For dataset F, entries with $W = 1$ are left blank, as with only one worker, the instances for P and F are identical. Thus, the number of instances for the future dataset is $54 - 3 * 2 = 48$.

6.2. Experiments

To present the experiment design, we first need to introduce the notion of a tolerance level.

[10] A shifted random distribution adds a constant k to all values of the original distribution. Since $\forall k\ Exp(1) + k > k$, by setting $k = 1$ we define at least one cut operation for each job (discretization is done by rounding to the nearest integer).

[11] Note that there is no loss of generality in assigning random dimensions to *every* cut operation, even though it is inconsistent with the real case, since the dimensions at task $i + 1$ depend on the ones at task $i - 1$.

[12] In general, the values suggested by the manufacturer can be represented as a function that, given the material type to be cut, the machine, and the cut channel, retrieves the suggested cutting parameters. For the sake of simplicity, we only rely on the cut channel, while the machine and the material type are simply randomly selected.

6.2.1. Tolerance levels

In single-objective minimization, tolerance levels quantify the acceptable deviation from the (unknown) global optimum g^*. For a given tolerance level $|tol|$ (or, in relative terms, $tol\% < 1$) and a lower bound $g_{LB} \leq g^*$, a solution with value g is considered optimal if $gap = g - g_{LB} \leq |tol|$ (or if $gap\% = gap/g_{LB} \leq tol\%$). When this happens, the search stops. In contrast, in the multi-objective case (with fixed priorities), the solver continues optimizing the subsequent goal. CPO provides tolerance levels, as it is able to estimate lower bounds while solving. Combining tolerance levels and *staticLex*$(\cdot)$ (cf. Table 1) seems a promising approach, especially for finding initial solutions. By relaxing the priority order of the goals up to a tolerance level, the solver is prevented from stalling in proving the optimality of the current goal. Instead, it can advance to the following goals, which may offer a more straightforward margin for improvement (while still not worsening any of the previously optimized goals).

6.2.2. Design

The model was evaluated using CPO v22.1.2 with default settings, except for tolerance and timeout. Experiments were executed in parallel (up to four concurrently), each bound to a CPU (`taskset -c cpu`, cpu $\in \{0, 1, 2, 3\}$) on Ubuntu 24.04.2 LTS (x86_64) with an AMD EPYC 9354 (32 cores, 3.25 GHz) and 62 GB RAM. CPO runs were executed in parallel mode (`workers=4`). We employed a single timeout, $t_{out} = 10$ minutes, and two different tolerance levels, $\langle |tol|, tol\% \rangle \in \{\langle 0, 0\% \rangle, \langle 5, 10\% \rangle\}$, where $|tol|$ is the absolute tolerance and $tol\%$ is the relative tolerance.[13] For every instance and for each tolerance, we launched CPO three times, each with a different random seed. The tolerance levels are motivated as follows: with $\langle 0, 0\% \rangle$ we aim at the global optimum, while with a tolerance $\langle 5, 10\% \rangle$ we investigate the impact of such a tolerance level on solution quality (in theory, it should find solutions having higher tardiness but lower energy and makespan). A small constant $|tol|$ is needed when the lower bound can be zero, as it is for the tardiness, because, in such a case, the $gap\%$ is undefined, and thus any value for $tol\%$ would have no effect. Thus, we set $|tol| = 5$ to avoid the solver getting stuck in minimizing tardiness without minimizing the other goals as well.

We collect the following data: the number of variables $Vars$ and of constraints $Const$ (cf. Table 3), the solving time in $[s]$ and memory in $[MB]$, the values of the goals ($Tardiness$ $[\#shifts]$, $Energy$ $[kWh]$, C_{max} $[min]$), the number of optimized goals $\#OG$, and the Boolean flags $Solved$ and $Optimal$.

6.2.3. A Bug in CPO

Unfortunately, we found bugs and unexpected results in CPO when combining tolerance levels with *staticLex*. Initially, IBM indicated that, although negative gaps could appear, optimality reports would remain reliable. Later, however, we found cases where CPO declared optimality despite some goals being well beyond their acceptable values ($g \gg g^* + \epsilon$). Therefore, we used tolerance levels, but did not employ the *staticLex* function. Instead, we optimized one objective at a time by externally updating the goal function and its bounds. IBM has indicated that this bug will be resolved in the next CPO version.[14]

6.3. Results

Tables 4 and 5 summarize the results for the two datasets, the present and future use cases, for a total timeout of 10 minutes. The resource usage is not reported in the tables, as the timeout is basically always reached, while the memory usage is consistently reasonable (the peak virtual memory size of 2113 $[MB]$ was recorded with $J = 500$, $M = 40$ with dataset F). Each table cell per $\langle J, M, W \rangle$ combination reports an average (or a percentage) over 9 runs (3 instances, each run 3 times). The objectives' averages are based only on the solved runs. $Solved[\%]$ ($Optimal[\%]$) denotes the percentage of runs for which a solution (the optimal solution, up to the given tolerance) is found. Note that a

[13] If both absolute and relative tolerances are defined, a solution is considered acceptable iff $gap \leq |tol| \vee gap\% \leq tol\%$.

[14] For details, check the discussion thread on the IBM community: https://community.ibm.com/community/user/question/consistency-of-cpo-lower-bounds-combined-with-staticlex-and-tolerances.

Table 4
Results for dataset P solved with $t_{out} = 10$ minutes and $tol\% \in \{0\%, 10\%\}$. The values for $\#OG$, $Solved[\%]$, $Optimal[\%]$ are averaged from 9 runs (3 per instance). For the objectives, only the solved runs are considered; if none, then the symbol "–" is reported.

			$\#OG$		$Solved[\%]$		$Optimal[\%]$		$Tardiness$		$Energy$		$Makespan$	
		$tol[\%]$	0	10	0	10	0	10	0	10	0	10	0	10
J	M	W												
50	3	1	0.0	0.0	100	100	0	0	12.7	13.0	38.3	38.3	1551	1571
	4	1	0.3	0.7	100	100	0	0	7.0	7.3	31.1	31.3	1535	1512
	8	2	1.0	1.0	100	100	0	0	2.7	2.7	41.6	41.6	972	972
	9	3	1.0	1.0	67	67	0	0	2.0	2.0	32.2	32.2	876	876
	15	5	2.0	2.0	67	67	67	67	0.0	0.5	30.7	30.7	410	444
	16	4	3.0	3.0	100	100	100	100	0.7	1.0	36.1	36.1	437	466
200	12	4	0.0	0.0	0	0	0	0	–	–	–	–	–	–
	16	4	0.0	0.0	67	67	0	0	93.5	98.0	144.5	144.6	1876	1848
	32	8	0.0	0.3	33	33	0	0	38.0	5.0	101.8	101.5	977	957
	36	12	0.0	0.0	0	0	0	0	–	–	–	–	–	–
	60	20	0.3	0.3	33	33	0	0	0.0	0.0	105.6	105.6	474	474
	64	16	0.3	0.3	33	33	0	0	2.0	2.0	106.9	106.9	567	567
500	30	10	0.0	0.0	0	0	0	0	–	–	–	–	–	–
	40	10	0.0	0.0	33	33	0	0	424.0	415.0	258.6	257.4	2086	2098
	80	20	0.0	0.0	0	0	0	0	–	–	–	–	–	–
	90	30	0.0	0.0	0	0	0	0	–	–	–	–	–	–
	150	50	0.0	0.0	0	0	0	0	–	–	–	–	–	–
	160	40	0.0	0.0	0	0	0	0	–	–	–	–	–	–
	Average		0.4	0.5	41	41	9	9	53.0	49.7	84.3	84.2	1069	1071

solution is optimal iff *all* the objectives have been optimized. $\#OG \in [0.0, 3.0]$ indicates the average number of optimized goals. Thus, $\#OG = 3.0$ means $Optimal[\%] = 100\%$, but $\#OG = 0.0$ does *not* imply $Solved[\%] = 0\%$. Also, being an average, $\#OG = 1.0$ does *not* imply that in all runs the first goal (i.e., tardiness) has been optimized.

First, and surprisingly, we can see that CPO consistently finds a schedule only for the general dataset F, but not for the present dataset P, although the latter is decomposable and involves fewer variables, constraints, and choices. A plausible reason could be the following. Although the choice space in P is smaller—since workers are fixed and each operation has fewer eligible machines—the job-machine-worker eligibility graph is sparse and disconnected. This can make finding a valid time interval I for operation o on resource r more difficult. By contrast, the additional worker-machine assignments in F make its eligibility graph connected and denser. Therefore, the chances of finding a resource r for operation o available at interval I are higher. So, CPO can find an initial solution for F more easily. Despite that, F has a larger search space, and the global optimum with no tolerance is never reached.

Second, we observe that the problem becomes more complex not only as the number of jobs increases, but also as the number of resources decreases. In fact, as M and W increase (i.e., as the J/M and J/W ratios decrease), $\#OG$ grows, and the solutions achieve higher quality (e.g., lower tardiness). The simple reason for this is that a reduction in the number of available resources limits the opportunities to distribute jobs across them. As a result, the remaining resources experience long queues of operations, and identifying the optimal solution requires determining the most efficient execution order for the jobs on each resource—a combinatorial task whose complexity grows factorially. Thus, the instances sharing equal ratios of J/M and J/W, such as $\{\langle 50, 15, 5\rangle, \langle 200, 60, 20\rangle, \langle 500, 150, 50\rangle\}$ also reflect similar optimal values for tardiness and makespan (cf. the entries in the tables having $\#OG \geq 1$). However, since the solver often reaches the timeout, this statement can be experimentally observed only in a few cases. Clearly, a larger J yields a larger search space, which can negatively impact solution quality.

In general, CPO appears to be more effective at minimizing makespan than tardiness. Although makespan optimization—the third objective—often cannot even start before the timeout (cf. the global

Table 5
Results for dataset F solved with $t_{out} = 10$ minutes and $tol\% \in \{0\%, 10\%\}$. The values for $\#OG$, $Solved[\%]$, $Optimal[\%]$ are averaged from 9 runs (3 per instance).

			$\#OG$		$Solved[\%]$		$Optimal[\%]$		$Tardiness$		$Energy$		$Makespan$	
		$tol[\%]$	0	10	0	10	0	10	0	10	0	10	0	10
J	M	W												
50	3	1	0.0	0.0	100	100	0	0	12.7	13.0	38.3	38.3	1551	1571
	4	1	0.3	0.7	100	100	0	0	7.0	7.3	31.1	31.3	1535	1512
	8	2	2.0	2.3	100	100	0	33	1.0	1.0	41.2	41.2	798	814
	9	3	2.0	2.0	100	100	0	0	1.0	0.7	35.2	35.2	485	493
	15	5	2.0	2.0	100	100	0	0	0.0	0.0	32.6	32.6	299	300
	16	4	2.0	2.7	100	100	0	67	0.0	0.0	35.7	35.9	343	355
200	12	4	0.0	0.0	100	100	0	0	76.0	82.0	179.8	178.7	1747	1865
	16	4	0.0	0.0	100	100	0	0	62.0	76.3	142.2	142.4	1576	1574
	32	8	0.7	1.0	100	100	0	0	11.9	2.3	129.4	128.9	932	871
	36	12	1.7	1.7	100	100	0	0	2.7	2.7	143.3	145.8	592	561
	60	20	2.0	2.0	100	100	0	0	0.0	0.0	103.7	105.7	335	321
	64	16	2.0	2.0	100	100	0	0	0.0	0.0	150.8	156.0	435	410
500	30	10	0.0	0.0	100	100	0	0	327.3	379.7	401.1	398.9	1670	1628
	40	10	0.0	0.0	100	100	0	0	494.7	403.2	357.0	356.1	1659	1709
	80	20	0.0	0.0	100	100	0	0	139.7	129.0	412.2	410.5	992	929
	90	30	0.7	0.7	100	100	0	0	22.8	27.3	359.9	362.0	601	614
	150	50	1.0	2.0	100	100	0	0	0.0	0.0	346.8	354.6	403	372
	160	40	1.0	2.0	100	100	0	0	0.0	0.0	337.8	345.8	447	428
	Average		1.0	1.2	100	100	0	6	64.4	62.5	182.1	183.3	911	907

$avg(\#OG) \leq 1.2$ in F and P), the resulting makespan values still appear reasonable. In contrast, despite being the primary objective, tardiness values can become quite large.[15] Thus, CPO shows limited ability in minimizing tardiness and may lack effective heuristics for it (e.g., *Earliest-Deadline First*).

Tolerance levels prevent the solver from spending excessive time optimizing a single goal while neglecting others. Their impact on solution quality appears once the optimization runs long enough for at least one goal to be optimized. Unfortunately, in dataset P, in most cases, CPO cannot even optimize the first goal, while, in F, only the first is optimized on average (cf. column $\#OG$ in the tables). Thus, the effects of the tolerances are limited in our experiments, and in fact, the average values of the objectives have a low deviation from each other. This is especially true for energy consumption since *(i)* its optimization rarely begins due to timeouts (energy is the second objective), and *(ii)* the model considers only the processing energy, but disregards idle and makespan-related energy costs.

7. Conclusion

Based on a real-world industrial use case in the steel industry, we introduced a novel and relevant extension of the Energy-Aware Double-Flexible Job-Shop Scheduling Problem (E-DFJSP). While E-DFJSP already includes energy costs and both machines' and workers' flexibility, we incorporated alternative machine modes (with varying energy consumption and processing time), as well as setup and transport operations—aspects often overlooked in traditional scheduling models. To address this problem, we proposed and implemented a Constraint Programming model using IBM CP Optimizer (CPO). We optimized tardiness, energy, and makespan (in this order) using defined optimality tolerance levels.

Conducting extensive experiments on two datasets from a steel-cutting company reflecting different factory policies in terms of worker responsibilities, we found that: *(i)* finding the optimal solution is unfeasible in most cases, *(ii)* CPO is consistently better in minimizing makespan rather than tardiness,

[15] All instances should have an optimal tardiness close to 0 since *(i)* $avg(Tard) < 3$ in all combinations of $\langle J, M, W\rangle$ having $\#OG \geq 1$, and *(ii)* instances sharing equal J/M and J/W ratios also share similar optimal values for $Tard$ and C_{max}.

(iii) CPO can find solutions for harder instances (with many choices available) but failing for simpler instances (with few choices), *(iv)* increasing the number of resources yields lower tardiness and makespan and eases solving, and *(v)* CPO is not a memory-intensive solver. Finding *(iv)* is supported by the fact that the resource-rich instances are the ones with the highest number of optimized goals (one or two).

Future research could extend this study in various meaningful ways. First, it appears promising to devise ways of assisting CPO in tardiness optimization by supplying a valid initial solution (e.g., from a greedy algorithm), or by designing (or learning [29]) a custom heuristic. Second, the presented model could be integrated with machine learning techniques, such as *decision-focused learning* [30], to include the uncertainty of parameters such as energy consumption or induced tool wear, which must be predicted based on the collected data in the factory. Third, inspired by [16], we aim to estimate the total energy consumption, i.e., to extend our approach by accounting for idle and overall factory energy.

Declaration on Generative AI

GPT-4 assisted only with language editing; all content was verified by the authors.

Acknowledgements

This work was supported by FFG, contract FO999910235. This research was funded in part by the Austrian Science Fund (FWF) 10.55776/COE12. We thank Giulia Francescutto, Martin Hackhofer, and Giray Havur for their valuable contributions throughout the development of this work.

References

[1] A. M. Omer, Energy, environment and sustainable development, Renewable and Sustainable Energy Reviews 12 (2008) 2265–2300.

[2] S. Pervaiz, S. Kannan, I. Deiab, H. Kishawy, Role of energy consumption, cutting tool and workpiece materials towards environmentally conscious machining: a comprehensive review, Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture 234 (2020) 335–354.

[3] C. Li, X. Zhao, H. Cao, L. Li, X. Chen, A data and knowledge-driven cutting parameter adaptive optimization method considering dynamic tool wear, Robotics and Computer-Integrated Manufacturing 81 (2023) 102491.

[4] A. Bhattacharya, S. Das, P. Majumder, A. Batish, Estimating the effect of cutting parameters on surface finish and power consumption during high speed machining of AISI 1045 steel using Taguchi design and ANOVA, Production Engineering 3 (2009) 31–40.

[5] M. R. Garey, D. S. Johnson, Computers and intractability, volume 29, W. H. Freeman & Co., 2002.

[6] G. Gong, Q. Deng, X. Gong, W. Liu, Q. Ren, A new double flexible job-shop scheduling problem integrating processing time, green production, and human factor indicators, Journal of Cleaner Production 174 (2018) 560–576.

[7] M. Schlenkrich, S. N. Parragh, Solving large scale industrial production scheduling problems with complex constraints: an overview of the state-of-the-art, Procedia Computer Science 217 (2023) 1028–1037.

[8] M. M. El-Kholany, M. Gebser, K. Schekotihin, Problem decomposition and multi-shot ASP solving for job-shop scheduling, Theory and Practice of Logic Programming 22 (2022) 623–639.

[9] G. Francescutto, K. Schekotihin, M. M. El-Kholany, Solving a multi-resource partial-ordering flexible variant of the job-shop scheduling problem with hybrid ASP, in: European Conference on Logics in Artificial Intelligence, Springer, 2021, pp. 313–328.

[10] M. Gebser, R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, P. Wanko, Theory solving made easy with clingo 5, in: Technical Communications of the 32nd International Conference on Logic Programming, Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016, pp. 1–15.

[11] P. Laborie, J. Rogerie, P. Shaw, P. Vilím, IBM ILOG CP Optimizer for scheduling: 20+ years of scheduling with constraints at IBM/ILOG, Constraints 23 (2018) 210–250.
[12] J. M. Novas, Production scheduling and lot streaming at flexible job-shops environments using constraint programming, Computers & Industrial Engineering 136 (2019) 252–264.
[13] M.-J. Park, A. Ham, Energy-aware flexible job shop scheduling under time-of-use pricing, International Journal of Production Economics 248 (2022) 108507.
[14] G. Da Col, E. C. Teppan, Industrial-size job shop scheduling with constraint programming, Operations Research Perspectives 9 (2022) 100249.
[15] P. Yunusoglu, S. Topaloglu Yildiz, Solving the flexible job shop scheduling and lot streaming problem with setup and transport resource constraints, International Journal of Systems Science: Operations & Logistics 10 (2023) 2221072.
[16] H. Terbrack, T. Claus, The generalized energy-aware flexible job shop scheduling model: A constraint programming approach, Computers & Industrial Engineering 204 (2025) 111065.
[17] G. A. Kasapidis, D. C. Paraskevopoulos, I. Mourtos, P. P. Repoussis, A unified solution framework for flexible job shop scheduling problems with multiple resource constraints, European Journal of Operational Research 320 (2025) 479–495.
[18] I. A. Chaudhry, A. A. Khan, A research survey: review of flexible job shop scheduling techniques, International Transactions in Operational Research 23 (2016) 551–591.
[19] L. Yin, X. Li, L. Gao, C. Lu, Z. Zhang, A novel mathematical model and multi-objective method for the low-carbon flexible job shop scheduling problem, Sustainable Computing: Informatics and Systems 13 (2017) 15–30.
[20] Z. Liu, S. Guo, L. Wang, Integrated green scheduling optimization of flexible job shop and crane transportation considering comprehensive energy consumption, Journal of Cleaner Production 211 (2019) 765–786.
[21] M. A. Salido, J. Escamilla, A. Giret, F. Barber, A genetic algorithm for energy-efficiency in job-shop scheduling, The International Journal of Advanced Manufacturing Technology 85 (2016) 1303–1314.
[22] M. Dai, D. Tang, A. Giret, M. A. Salido, Multi-objective optimization for energy-efficient flexible job shop scheduling problem with transportation constraints, Robotics and Computer-Integrated Manufacturing 59 (2019) 143–157.
[23] J.-q. Li, J.-w. Deng, C.-y. Li, Y.-y. Han, J. Tian, B. Zhang, C.-g. Wang, An improved jaya algorithm for solving the flexible job shop scheduling problem with transportation and setup times, Knowledge-Based Systems 200 (2020) 106032.
[24] X. Chen, J. Li, Z. Wang, J. Li, K. Gao, A genetic programming based cooperative evolutionary algorithm for flexible job shop with crane transportation and setup times, Applied Soft Computing 169 (2025) 112614. doi:10.1016/j.asoc.2024.112614.
[25] L. Meng, C. Zhang, B. Zhang, Y. Ren, Mathematical modeling and optimization of energy-conscious flexible job shop scheduling problem with worker flexibility, IEEE Access 7 (2019) 68043–68059.
[26] J.-Y. Moon, J. Park, Smart production scheduling with time-dependent and machine-dependent electricity cost by considering distributed energy resources and energy storage, International Journal of Production Research 52 (2014) 3922–3939.
[27] J. M. Fernandes, S. M. Homayouni, D. B. Fontes, Energy-efficient scheduling in job shop manufacturing systems: a literature review, Sustainability 14 (2022) 6264.
[28] P. Brucker, R. Schlie, Job-shop scheduling with multi-purpose machines, Computing 45 (1990) 369–375.
[29] A. Novikov, N. Vũ, M. Eisenberger, E. Dupont, P.-S. Huang, A. Z. Wagner, S. Shirobokov, B. Kozlovskii, F. J. Ruiz, A. Mehrabian, et al., AlphaEvolve: A coding agent for scientific and algorithmic discovery, arXiv preprint arXiv:2506.13131 (2025).
[30] J. Mandi, E. Demirović, P. J. Stuckey, T. Guns, Smart predict-and-optimize for hard combinatorial optimization problems, in: Proceedings of the AAAI conference on artificial intelligence, volume 34, 2020, pp. 1603–1610.