INTERNATIONAL TRANSACTIONS IN OPERATIONAL RESEARCH

Intl. Trans. in Op. Res. XX (20XX) 1–28
DOI: xx.xxxx/itor.xxxxx

Energy-aware flexible job shop scheduling problem with nonlinear routes and position-based learning effect

Ernesto G. Birgin[a,*], José Angel Riveaux[a] and Débora P. Ronconi[b]

[a]*Department of Computer Science, Institute of Mathematics and Statistics, University of São Paulo, Rua do Matão, 1010, Cidade Universitária, 05508-090, São Paulo, SP, Brazil.*
[b]*Department of Production Engineering, Polytechnic School, University of São Paulo, Av. Prof. Luciano Gualberto, 1380, Cidade Universitária, 05508-010 São Paulo, SP, Brazil*
E-mail: egbirgin@ime.us.br [Ernesto G. Birgin]; jangel.riveaux@usp.br [José Angel Riveaux]; dronconi@usp.br [Débora P. Ronconi]

Received DD MMMM YYYY; received in revised form DD MMMM YYYY; accepted DD MMMM YYYY

Abstract

Sustainability has become one of the main objectives in all human activities and, in particular, in manufacturing environments. In this paper we consider the flexible job shop scheduling problem with the objective of minimizing energy consumption. As it is known that a considerable part of the energy consumption occurs when the machines are on and idle, the addressed problem includes the possibility of turning the machines off and on between processing operations. To bring the problem closer to the large variety of real-world problems it encompasses, we include two relevant factors: nonlinear routes and position-based learning effect. The treated problem is formally described through a mixed integer linear programming model. We propose constructive heuristics, two types of neighborhood with which we construct local search schemes and three metaheuristics, namely, general variable neighborhood search, greedy randomized adaptive search procedure and simulated annealing. We conduct a large number of experiments to evaluate the performance of the introduced methods, on small-sized and large-sized instances. In the large-sized instances, the general variable neighborhood search, that combines the two neighborhoods into a single method, is particularly effective. In the small-sized instances with known optimal solution, the greedy randomized adaptive search procedure finds solutions that, on average, are within 0.22% of the optimal solution.

Keywords: energy-aware scheduling; flexible job shop; nonlinear routes; arbitrary precedence constraints; learning effect; constructive heuristics; local search; metaheuristics

1. Introduction

The flexible job shop (FJS) is a scheduling problem at the core of manufacturing environments that is notable for its number of practical applications. The problem is NP-hard because it includes the job shop (JS) scheduling problem, known to be NP-hard (Garey et al., 1976), as a particular case. Because of its

*Author to whom all correspondence should be addressed (e-mail: egbirgin@ime.us.br).

relevance and difficulty of solution, a wide variety of heuristic and metaheuristic methods have been developed in the recent literature for its solution, see Dauzère-Pérès et al. (2024); Xie et al. (2019). At the same time, due to the large number of real-world problems that fall within its scope, various practical aspects have been included in its formulation. In this work, we consider the FJS scheduling problem with nonlinear routes and position-based learning effect. By learning effect we mean that the processing times of the operations in the machines depend on the position that the operations occupy within the machines, i.e. we consider a position-based learning effect. We refer to Biskup (1999); Cheng and Wang (2000); Gupta and Gupta (1988) as the first applications of the learning effect idea in scheduling problems. By nonlinear routes we refer to the fact that the operations that constitute a job do not have to follow a linear order for their execution, but their precedence relations are given by an arbitrary directed acyclic graph. In particular, this may allow different operations of the same job to be processed in parallel. See Birgin et al. (2014) for details. It is worth noting that the inclusion of nonlinear routes in the FJS makes it possible to tackle the online printing shop (OPS) scheduling problem, a real and challenging problem in today's printing industry (Araujo et al., 2024a,b; Birgin et al., 2015; Lunardi et al., 2020, 2021). As described in Lunardi et al. (2020), in the OPS scheduling problem, orders of products to be manufactured, such as books, brochures, flyers, photo albums, and many others, are received online. Each type of product has a different production plan, but they all involve a printing operation. When a significant number of orders is reached, in order to save raw material (paper), a cutting stock problem is solved to merge the printing operations of the different orders placed. The orders whose printing operations are combined form a single job. Thus, the jobs in the OPS scheduling problem, which consist of a heterogeneous set of operations with arbitrary precedence constraints, are extremely diverse. In the study of the OPS problem carried out in Lunardi et al. (2020, 2021), several complicating features such as periods of unavailability of the machines, resumable operations, sequence-dependent setup times, partial overlapping of operations with precedence constraints, release times, and fixed operations were addressed. However, a complicating factor of this real-world problem was neglected: several operations are performed by human operators. These tasks include computer-aided layout of materials to be printed, assembling the various parts of a book and collating the covers, handling the cutting tools, packaging the finished products, and others. These tasks performed by human operators are subject to the learning effect. Assuming that a human operator learns by repeatedly performing the same operation, it is reasonable to say that, within certain limits, the ith execution will be faster than the $(i-1)$th. While there are other alternatives, this gives rise to the idea of a learning effect model based on the position of the operation within the list of operations to be performed by the same operator.

In the present work, we recognize that sustainability has gained paramount importance over the past few decades, becoming a top global objective. In a simple way, sustainability means meeting the needs of the present without affecting future generations. Therefore, recent literature has referred as green scheduling to scheduling problems that take into account workers' safety (Gong et al., 2019), well-being of workers (Destouet et al., 2024), machinery preservation (Wu and Sun, 2018), carbon emissions (Li and Chen, 2023; Zhu et al., 2020), noise emissions and energy consumption and/or cost (Gahm et al., 2016), among others. Energy, in particular, has been a focal point in The 2030 Agenda for Sustainable Development (Assembly, 2015) adopted at the United Nations Sustainable Development Summit in 2015. For this reason, in the present work we consider the energy-aware goal of minimizing energy consumption. As it is known that a considerable part of the energy consumption occurs when the machines are on and idle, the problem considered includes the possibility of turning the machines off and on between processing operations. As most of the time the energy consumed comes from non-renewable sources, there is a direct relationship between energy consumption and carbon emission, which intensifies the warming effect.

The energy consumption scheme in place takes into account the cost of turning machines on and off, the cost of each machine in processing each operation, the cost of keeping a machine on and idle, and a cost related to keeping the facility running. It should be noted that the possibility of switching a machine off and on between the processing of two successive operations is considered if this results in a lower cost than the cost of keeping the machine on and idle. However, at the same time that it may be less costly, turning a machine off and on may take longer, increasing the completion time of one or more jobs. Thus, the objective function of the problem is not regular. The starting point of this work is the modeling of the problem under consideration with mixed integer linear programming. The modeling is twofold. On the one hand, it aims to describe the problem exactly. On the other hand, it is used to solve small instances of the problem with an exact solver in order to check the effectiveness of the proposed methods. In the sequel, we develop a constructive list scheduling heuristic and two different neighborhoods: one based on removing and reinserting a single operation and another based on removing a single operation, destroying, reinserting, and reconstructing. On the basis of the neighborhoods, two local search algorithms and three metaheuristics are developed. The metaheuristics considered are simulating annealing (SA), greedy randomized adaptive search procedure (GRASP) and generalized variable neighborhood search (GVNS).

The rest of this work is organized as follows. A literature review is presented in Section 2. In Section 3, we formally describe the problem and formulate it as a mixed integer linear programming (MILP) problem. In Sections 4 and 5, we introduce a constructive heuristic and two local search strategies, respectively. In Section 6, we describe the metaheuristics considered. Extensive numerical experiments are presented in Section 7. Section 8 includes conclusions and directions for future work.

Notation. The symbol e represents the mathematical constant whose value is approximately 2.71828, $\ln(\cdot)$ is the natural logarithm, $\mathbb{R}_{>0} = \{x \in \mathbb{R} \mid x > 0\}$, and $\mathbb{Z}_{>0} = \{x \in \mathbb{Z} \mid x > 0\}$.

2. Literature review

In the following, we present a literature review of papers dealing with energy consumption in the FJS environment. It should be noted that, while a few of them take into account a learning effect, none of them consider nonlinear routes. The design of models for the FJS problem with the minimization of energy consumption has been the subject of a few recent publications. In Mouzon et al. (2007) it is highlighted that, in scheduling problems, a significant part of the energy consumption corresponds to non-bottleneck machines that remain on and idle. Based on this premise, Meng et al. (2019); Zhang et al. (2017a,b,c) propose mathematical models for the FJS scheduling problem, with the objective of minimizing energy consumption and allowing machines to be turned on and off between processing operations. (A constraint programming model and a minor modification to the MILP model proposed in Meng et al. (2019) are presented in Ham et al. (2021).) In Meng et al. (2019) a comparison with the models previously proposed in Zhang et al. (2017a,b,c) is presented, showing that the model proposed in Meng et al. (2019) is more effective/efficient when trying to solve small instances with an exact method. The model introduced in the present work, which uses the same binary variables as model 2.2 proposed in Meng et al. (2019), is based on the model proposed in Araujo et al. (2024b). The choice for binary variables indicating whether an operation i is attributed to position r of a machine k was driven by the need to model the learning effect that depends on the position that an operation takes in the machine (the higher the position the shorter the processing time). When compared to the model in Meng et al. (2019), it additionally includes the precedence relations between operations of the same job given by an arbitrary directed acyclic graph (nonlinear routes) and the effect of learning on processing time. When compared to the model presented in Araujo et al. (2024b), it differs in the objective function, which implies in considering, for example, the possibility of turning machines off and on between processing operations. Besides, it is worth mentioning that the presence of the model in the present work serves to clearly describe the problem under consideration.

In Li et al. (2020) the FJS environment with dual resources and the minimization of energy consumption is considered. The problem is described through a MILP model. For its solution, different neighborhoods, a local search, a restarting mechanism and an optimization method based on migrating birds are proposed. In Lu et al. (2019), the problem of minimizing the energy consumption combined with the completion time in an FJS environment is considered. As the makespan is multiplied by the energy consumption per time unit, this component of the objective function corresponds to consider an energy consumption relative to keeping the plant running. This means that the objective can be seen as minimizing energy consumption only. For this problem, a water wave optimization algorithm is considered.

In Lei et al. (2016), the conflict between minimizing energy consumption and balancing between the working lines is studied. The problem with the two objectives is modeled as a bi-objective problem and a shuffled frog-leaping algorithm (SFLA) is proposed. In Ren et al. (2020) it is considered an FJS environment with a particular type of nonlinear routes: some operations are standard operations that must be processed on machines while others are assembly operations that must be processed on assembly stations and require a set of operations to be previously completed. The objective of minimizing the makespan and energy consumption. For this bi-objective problem, a hybrid metaheuristic combining genetic algorithms with particle swarm optimization is proposed. In Wu and Sun (2018), turning machines off and on and controlling the speed at which machines operate are considered as ways to reduce energy consumption. The considered problem simultaneously optimizes the makespan, the energy consumption and the number of times the machines need to be turned off and on. For this problem, a non-dominated sorted genetic algorithm (NSGA-II) that integrates a green scheduling heuristic is proposed. In Gong et al. (2019), it is considered a multi-objective problem with five objectives, among them, the total energy cost. In an environment with dynamic electricity prices, it may be interesting to process operations in the night period, which would increase the cost with labor. Therefore, another cost considered is the labor cost. The other three objectives are the maximum load of a machine, the sum of all machines load and the makespan. For this problem, a NSGA-III method is designed. In Wu et al. (2019), the problem under consideration is a manufacturing problem of aerospace and military products, in which, due to the long processing cycle of the components, tool wear affects the processing of the work. The problem fits into an FJS environment and the goal is to simultaneously minimize makespan and energy consumption, taking into consideration the deterioration effect of processing times. The deterioration model is time-dependent and the energy consumption model follows a very specific energy consumption profile for operations that are all cutting operations. For this problem, a bi-objective hybrid pigeon-inspired optimization and simulated annealing algorithm is developed.

In Li and Chen (2023), a bi-objective problem in which makespan and carbon emissions are minimized is considered. The processing times are affected by Dejong's learning effect (De Jong, 1957), but the carbon emission from the processing of each operation is considered to be fixed and does not depend on its processing time. Therefore, even if there were a direct relationship between energy consumption and carbon emissions, minimization of one would not be equivalent to minimization of the other, since energy consumption is related to processing time. For this problem, a multiobjective sparrow search algorithm is proposed. For an overview of carbon emission as a performance measure in the manufacturing industry, see Laurent et al. (2010). More recently, Gong et al. (2024) dealt with the simultaneous minimization of makespan and energy consumption in an FJSP environment. In the considered scenario, some operations have a linear route, while others are independent and have no precedence relationship linking them to any other operation. The calculation of energy consumption does not take into account the possibility of turning off and on the machines. The authors proposed an algorithm based on a combination of the Memetic Algorithm (MA) and the Non-Dominated Sorting Genetic Algorithm II (NSGA-II).

3. Problem definition and formulation

The FJS scheduling problem is an extension of the JS scheduling problem. In the JS there is a set $\mathcal{O}$ of operations and a set $\mathcal{F}$ of machines. For each operation $i \in \mathcal{O}$, a machine $f_i \in \mathcal{F}$ is given that must process operation i. The operations are divided into jobs $J_1, J_2, \dots, J_n$ such that $\mathcal{O} = \cup_{k=1}^{n} J_k$ and $J_{k_1} \cap J_{k_2} = \emptyset$ whenever $k_1 \neq k_2$. The operations of the same job must be executed in a predefined linear order. The "F" in the FJS stands for "flexible" and refers to the fact that instead of there being only one machine f_i capable of processing operation i, for each operation i there is a subset of machines $\mathcal{F}_i \subseteq \mathcal{F}$ that can process it. This feature is known as routing flexibility. The objective is to allocate each operation to a machine and decide in which order the machine should execute the operations allocated to it, so that the precedences between operations are honored and some predefined objective is minimized.

The FJS with nonlinear routes is an extension of the FJS scheduling problem. (See (Dauzère-Pérès et al., 2024, §6.1) for a discussion of the different designations given in the literature for this problem.) The extension consists in relaxing the precedence constraints of the operations of the same job. Instead of a linear order, the relationships can be given by an arbitrary directed acyclic graph (DAG). This relaxation corresponds to important practical cases in the modern printing industry. For example, a job may be to produce a book and the operations may, simplistically, include a layout operation preceding all others, the printing (in parallel and without precedence between them) of different blocks of sheets, and, finally, gathering all the sheets blocks and gluing them together with the covers. Clearly, lots of other real-world problems fit into the same description.

The FJS with nonlinear routes and position-based learning effect adds a further real-world ingredient to the problem. In classical scheduling problems, given an operation $i \in \mathcal{O}$ and a machine $k \in \mathcal{F}_i$, the processing time p_{ik} that machine k needs to process operation i is part of the problem data. However, in the real world a machine (human operator) learns through the repetitive execution of operations. The first time it does something it takes some time, the second time it does it faster and so on. That is why we consider in the present work that the actual processing time is a function that depends on a standard time p_{ik} and on the position that operation i occupies in the list of operations to be executed by machine k. If we call this function ψ_α, then we say that the effective processing time of operation i, on machine k, if it occupies the position r in the list of machine k, is given by $\psi_\alpha(p_{ik}, r)$. In this work, we consider $\psi_\alpha(p, r) = \lfloor p/r^\alpha + 1/2 \rfloor$, where $\alpha > 0$ is a given learning rate. Adding $1/2$ and taking the floor has the purpose of rounding the potentially non-integer value p/r^α.

It now remains to mention the goal to be minimized. In general, the makespan is considered. In this work we consider the energy consumption. The data we have for this purpose, related to each machine $k \in \mathcal{F}$, are: (a) how much the machine consumes, per unit of time, when it is processing an operation (named γ_k^{proc}), (b) how much the machine consumes, per unit of time, when it is on and idle (named γ_k^{idle}), (c) how long it takes for the machine to be turned on and what is the consumption of turning it on (named τ_k^{on} and γ_k^{on}, respectively), (d) how long it takes for the machine to be turned off and what is the consumption of turning it off (named τ_k^{off} and γ_k^{off}, respectively) and (e) what is the maximum time the machine can be on and idle (named τ_k^{idle}). In addition, we also know the energy cost, per unit of time, of having the plant running (named γ^{extra}). We consider that all machines start off and must be shut down at the end. Of course, a machine must be turned on before processing its first operation. The plant should start running the instant the first machine is turned on and stop running the instant the last machine completes its shutdown process. With this data, for each machine and each pair of operations that are processed on it consecutively, we must decide whether the machine should be turned off and on again or whether it should remain on and idle. Naturally, if the decision is to be turned off and on, there must be, between the completion of one operation and the start of the next, enough time to turn the machine off and on. An interval greater than the minimum imposed by the precedence relations between two successive operations may allow the machine to be turned off and on. This can be advantageous from the point of view of energy consumption, while increasing the completion time of one or more jobs. Therefore, the objective function considered in this work is non-regular.

In some sense, considering that there is a cost, per unit of time, for having the plant running, one might think that minimizing energy consumption is nearly the same thing as minimizing makespan. The following example shows that this is not the case. Let us consider the instance with 16 operations divided into 4 jobs whose precedence DAG is shown in Figure 1. In this instance, we have $\mathcal{O} = \{1, 2, \dots, 16\}$ and $\mathcal{F} = \{1, 2, \dots, 7\}$. The $\mathcal{F}_i$ sets for $i \in \mathcal{O}$ and the standard processing times p_{ik} for $i \in \mathcal{O}$ and $k \in \mathcal{F}_i$ are represented in Table 1. The data from (a) to (e) specified in the previous paragraph and describing the machines' energy consumption are shown in Table 2. The cost per unit of time to operate the plant is $\gamma^{\text{extra}} = 422$. We solved this instance by considering two different objectives. In one case, we minimized the energy consumption. In the other case, we solved a problem whose solution is the minimum energy solution among those that minimize the makespan. The solutions to these two problems are shown in Figures 2a and 2b, respectively. The optimal solution of the problem corresponding to minimizing energy consumption has energy consumption $E = 200{,}793$ and makespan $C_{\max} = 270$. The solution of the second problem has energy consumption $E = 200{,}955$ and makespan $C_{\max} = 267$, i.e., higher consumption and lower makespan. This clearly shows that the problems are different.

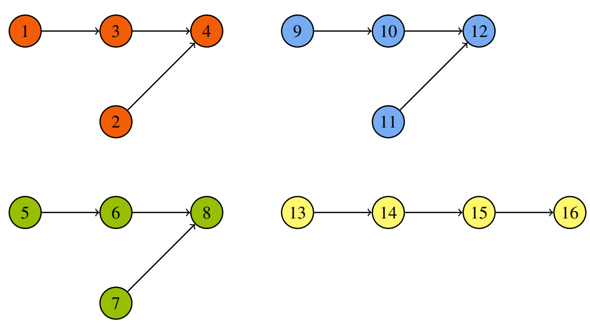


Fig. 1. DAG representing the precedence relationships of an instance with 16 operations divided into 4 jobs. The number of jobs corresponds to the number of connected components of the DAG.

We now introduce the mathematical MILP formulation of the FJS scheduling problem with nonlinear routes and position-based learning effect, in order to minimize the energy consumption. We first define the data of an instance of the problem, most of which were already mentioned. Subsequently, we describe the decision variables of the model and the model itself.

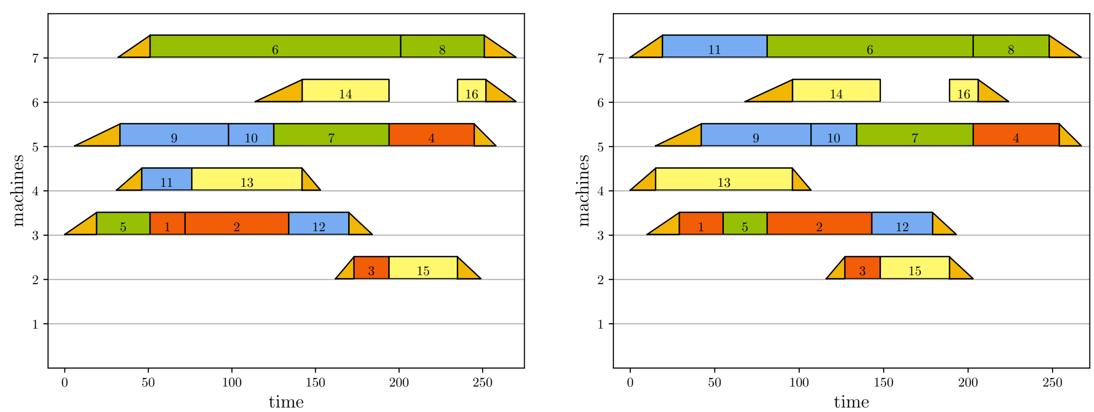
Instance data:

		$\mathcal{O}$															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$\mathcal{F}$	1	–	52	155	–	–	–	–	–	59	–	–	41	–	–	189	–
	2	185	90	21	142	–	–	99	–	–	–	–	–	–	179	50	–
	3	26	86	–	–	32	–	–	199	–	–	159	55	–	–	–	–
	4	–	–	144	195	–	–	–	–	129	–	30	195	81	132	95	163
	5	–	121	65	77	185	–	96	199	65	33	–	–	–	–	91	–
	6	126	–	–	146	–	–	–	–	84	146	151	188	–	52	–	21
	7	144	55	101	125	76	150	197	62	–	–	62	177	–	103	–	–

Table 1
Standard processing times and representation of the sets $\mathcal{F}_i$ for all $i \in \mathcal{O}$ of the small illustrative instance with 16 operations and 7 machines whose precedence relations are given in the DAG of Figure 1.

k	γ_k^{proc}	γ_k^{idle}	τ_k^{on}	γ_k^{on}	τ_k^{off}	γ_k^{off}	τ_k^{idle}
1	87	8	15	750	11	550	162
2	86	5	11	638	14	812	290
3	81	8	19	1653	14	1218	358
4	85	8	15	930	11	682	201
5	93	9	27	2025	13	975	333
6	92	9	28	1960	18	1260	357
7	96	5	19	1672	19	1672	668

Table 2
Data describing the energy consumption of the machines of the small illustrative instance with 16 operations and 7 machines whose precedence relations are given in the DAG of Figure 1.



(a) (b)

Fig. 2. Graphical representation of optimal solutions to (a) the problem of minimizing energy consumption and (b) the problem that consists in choosing the solution of minimum energy consumption among the solutions that minimize the makespan. In the pictures, the triangles represent the process of turning the machines on and off.

$\mathcal{O}$ set of operations,
$\mathcal{F}$ set of machines,
$\mathcal{O}_k$ set of operations that can be processed by machine $k \in \mathcal{F}$,
$\mathcal{F}_i$ set of machines that can process operation $i \in \mathcal{O}$,

$\widehat{A}$ set of directed arcs in $\mathcal{O} \times \mathcal{O}$ that represent operations' precedence constraints (the precedence constraints DAG is given by $D = (\mathcal{O}, \widehat{A})$),
p_{ik} standard processing time of operation $i \in \mathcal{O}$ in machine $k \in \mathcal{F}_i$,
γ_k^{proc} energy consumption, per unit of time, of machine $k \in \mathcal{F}$ when it is processing an operation,
γ_k^{idle} energy consumption, per unit of time, of machine k when it is on and idle,
τ_k^{on} time required to turn on machine $k \in \mathcal{F}$,
γ_k^{on} fixed energy consumption of turning on machine $k \in \mathcal{F}$,
τ_k^{off} time required to turn off machine $k \in \mathcal{F}$,
γ_k^{off} fixed energy consumption of turning off machine $k \in \mathcal{F}$,
τ_k^{idle} time limit for machine $k \in \mathcal{F}$ to remain on and idle,
γ^{extra} energy consumption, per unit of time, of having the plant running.
Constants p_{ik}, γ_k^{proc}, γ_k^{idle}, τ_k^{on}, γ_k^{on}, τ_k^{off}, γ_k^{off}, τ_k^{idle}, and γ^{extra} are assumed to be non-negative.

Decision variables:

x_{ikr} is 1 if operation $i \in \mathcal{O}$ is the r-th operation in the list of operations to be processed by machine $k \in \mathcal{F}_i$ and 0 otherwise (here r varies from 1 to $|\mathcal{O}_k|$),
y_{kr} is 1 if machine $k \in \mathcal{F}$ is turned off and on after processing the operation that is in the r-th position in the list of operations that the machine processes and 0 if the machine remains on and idle during that period (here r varies from 1 to $|\mathcal{O}_k| - 1$),
s_i starting time of the processing of operation $i \in \mathcal{O}$,
h_{kr} starting time of the processing of the operation that is in the r-th position in the list of operations processed by machine $k \in \mathcal{F}$ (here r varies from 1 to $|\mathcal{O}_k|$),
p'_i actual processing time of operation $i \in \mathcal{O}$ (this is an auxiliary variable that simplifies the presentation of the model),
t_{kr}^{idle} time that the machine $k \in \mathcal{F}$ remains idle between operations at positions r and $r+1$ in the list of operations it processes (here r varies from 1 to $|\mathcal{O}_k| - 1$).

The proposed model uses decision variables that determine the position of each operation within each machine's list. This is the most natural way to model the problem, since it allows to compute the effective processing time of an operation on a machine, which depends on the position of the operation in the machine's list. The model is based on the models presented in Birgin et al. (2014) and Araujo et al. (2024b), but the entire part related to energy consumption and the decision whether a machine should remain on and idle or should be turned off and on between the processing of two consecutive operations is new. Position-based decision variables for scheduling problems were initially used in Wagner (1959) and were also considered in Wilson (1989) in the flowshop scheduling problem.

The proposed MILP model follows:

$$\text{Minimize} \quad \sum_{k \in \mathcal{F}} \left\{ \gamma_k^{\text{proc}} \left(\sum_{i \in \mathcal{O}_k} \sum_{r=1}^{|\mathcal{O}_k|} \phi(p_{ik}, r)\, x_{ikr} \right) + \left(\gamma_k^{\text{on}} + \gamma_k^{\text{off}} \right) \left(\sum_{i \in \mathcal{O}_k} x_{ik1} + \sum_{r=1}^{|\mathcal{O}_k|-1} y_{kr} \right) + \gamma_k^{\text{idle}} \sum_{r=1}^{|\mathcal{O}_k|-1} t_{kr}^{\text{idle}} \right\} + \gamma^{\text{extra}} C_{\max} \tag{1}$$

subject to

$$\sum_{k \in \mathcal{F}_i} \sum_{r=1}^{|\mathcal{O}_k|} x_{ikr} = 1, \qquad i \in \mathcal{O}, \tag{2}$$

$$\sum_{i\in\mathcal{O}_k} x_{ikr} \leq 1, \qquad k\in\mathcal{F}, r=1,\ldots,|\mathcal{O}_k|, \tag{3}$$

$$\sum_{i\in\mathcal{O}_k} x_{i,k,r+1} \leq \sum_{i\in\mathcal{O}_k} x_{ikr}, \qquad k\in\mathcal{F}, r=1,\ldots,|\mathcal{O}_k|-1, \tag{4}$$

$$p'_i = \sum_{k\in\mathcal{F}_i}\sum_{r=1}^{|\mathcal{O}_k|} \phi(p_{ik},r)x_{ikr}, \qquad i\in\mathcal{O}, \tag{5}$$

$$s_i + p'_i \leq s_j, \qquad (i,j)\in\widehat{A}, \tag{6}$$

$$s_i + p'_i - \left(2 - x_{ikr} - x_{j,k,r+1}\right)M \leq s_j, \qquad i\neq j\in\mathcal{O}, k\in\mathcal{F}_i\cap\mathcal{F}_j,\ r=1,\ldots,|\mathcal{O}_k|-1, \tag{7}$$

$$h_{kr} \leq s_i + M(1-x_{ikr}), \qquad i\in\mathcal{O}, k\in\mathcal{F}_i, r=1,\ldots,|\mathcal{O}_k|, \tag{8}$$

$$s_i - M(1-x_{ikr}) \leq h_{kr}, \qquad i\in\mathcal{O}, k\in\mathcal{F}_i, r=1,\ldots,|\mathcal{O}_k|, \tag{9}$$

$$h_{k,r+1} - \left(h_{kr} + \sum_{i\in\mathcal{O}_k}\phi(p_{ik},r)x_{ikr}\right) - My_{kr} \leq t^{\text{idle}}_{kr}, \qquad k\in\mathcal{F}, r=1,\ldots,|\mathcal{O}_k|-1, \tag{10}$$

$$t^{\text{idle}}_{kr} \leq \tau^{\text{idle}}_k(1-y_{kr}), \qquad k\in\mathcal{F}, r=1,\ldots,|\mathcal{O}_k|-1, \tag{11}$$

$$t^{\text{idle}}_{kr} \geq 0, \qquad k\in\mathcal{F}, r=1,\ldots,|\mathcal{O}_k|-1, \tag{12}$$

$$h_{k,r+1} - \left(h_{kr} + \sum_{i\in\mathcal{O}_k}\phi(p_{ik},r)x_{ikr}\right) + M(1-y_{kr}) \geq \tau^{\text{off}}_k + \tau^{\text{on}}_k, \qquad k\in\mathcal{F}, r=1,\ldots,|\mathcal{O}_k|-1, \tag{13}$$

$$h_{kr} + \left(\sum_{i\in\mathcal{O}_k}\phi(p_{ik},r)x_{ikr}\right) + \tau^{\text{off}}_k\sum_{i\in\mathcal{O}_k} x_{ikr} \leq C_{\max}, \qquad k\in\mathcal{F}, r=1,\ldots,|\mathcal{O}_k|, \tag{14}$$

$$h_{kr} \geq \tau^{\text{on}}_k \sum_{i\in\mathcal{O}_k} x_{ikr}, \qquad k\in\mathcal{F}, r=1, \tag{15}$$

$$s_i \geq 0, \qquad i\in\mathcal{O}, \tag{16}$$

$$x_{ikr}\in\{0,1\}, \qquad i\in\mathcal{O}, k\in\mathcal{F}_i, r=1,\ldots,|\mathcal{O}_k|, \tag{17}$$

$$y_{kr}\in\{0,1\}, \qquad k\in\mathcal{F}, r=1,\ldots,|\mathcal{O}_k|-1. \tag{18}$$

The objective function (1) represents the minimization of energy consumption. The objective function is composed by the sum of two terms. The first term is a sum over all machines, while the second term refers to the energy consumption related to keeping the plant running. The latter cost is simply the product of the energy consumption per unit of time multiplied by the time elapsed from the moment the first machine is turned on to the moment the last machine is turned off. The machines' term sums, for each machine, the energy consumption associated with turning it on and off, the energy consumption associated with processing operations, and the energy consumption of the periods when it is on and idle. The consumption associated with turning a machine on and off is the consumption of turning it on and off once multiplied by the number of times the machine must be turned on and off. The consumption associated with processing operations is the product of the consumption per unit of time multiplied by the time the machine spends processing operations. This time is influenced by the learning effect. The idle time of the machine corresponds to the sum of the intervals between the processing of consecutive operations in which it was decided not to turn off the machine.

Constraints (2) define that each operation must be processed by exactly one machine and occupy only one position. Constraints (3) impose that a machine position can only be associated with at most one operation. Constraints (4) say that a machine position can only be occupied by an operation if all previous positions are also occupied. Constraints (5) define the actual processing time of each operation, taking into account the learning effect, in order to simplify the presentation of the model. Constraints (6) enforce that the precedence constraints between operations in the DAG be respected. Constraints

(7) state that, if both operations i and j are assigned to the same machine k and operation i precedes operation j, i and j do not overlap. Constraints (8) and (9) associate the two types of variables that refer to the start time of operations. Variable s_i refers to the start time of operation i. Variable h_{kr} refers to the start time of the r-th operation of machine k. If $x_{ikr} = 1$, then these two variables must coincide. Constraints (10) say that, if between two operations processed consecutively on the same machine, the machine remains on and idle, then the variable defining the idle time must be not smaller than the difference between the completion time of the first operation and the starting time of the second operation. Constraints (11) and (12) say that, if between two operations processed consecutively on the same machine, the machine is turned off and on, then the variable defining the idle time in-between these two operations must be zero. When a machine remains on and idle between processing two consecutive operations, constraints (11) state that the machine's idle time cannot exceed its given upper limit. Constraints (13) ensure that if machine k is turned off and on after processing the r-th operation, then there is sufficient time to do so before starting the processing of the operation at position $r + 1$. Constraints (14) state that the makespan be greater than or equal to the completion time of each operation plus the machine (processing the operation) shutdown time. Combining these constraints with the minimization of (1), $C_{\max}$ is set to be the instant at which the last machine shuts down. (Note the abuse of notation here, as this is not the usual definition of makespan). Constraints (15) say that before the processing of the first operation of each machine, there must be enough time to turn on the machine. Constraints (16) to (18) refer to the domain of the decision variables.

In the model, M is a sufficiently large number. In practice, the value of M may be different in each constraint. In (7), M needs to be an upper bound on the completion time of any operation in an optimal solution. In (8) and (9), M needs to be an upper bound on the starting time of any operation in an optimal solution. In (10), M needs to be an upper bound on the interval between any pair of consecutive operations in any machine, in an optimal solution. In (13), we can have an M_k for each k and M_k can be equal to $\tau_k^{\text{off}} + \tau_k^{\text{on}}$. In (7), (8), (9) and (10), all necessary bounds are upper bounded by an upper bound on the optimal makespan, which can be given by $\Gamma_1 = \sum_{i \in \mathcal{O}} \max_{k \in \mathcal{F}_i}\{p_{ik}\}$.

As mentioned above, for every machine $k \in \mathcal{F}$ and every position $r \in \{1, \ldots, |\mathcal{O}_k| - 1\}$, the constraints (10) say that, if machine k remains on and idle between the operations processed in position r and $r + 1$, then the variable t_{kr}^{idle} must be greater than or equal to the difference between the completion time of the rth operation and the starting time of the operation at position $r + 1$. However, this variable appears multiplied by the positive constant γ_k^{idle} in the objective function in a minimization problem. Therefore, in an optimal solution, the constraint must be active. Thus, the valid constraints

$$h_{k,r+1} - \left(h_{kr} + \sum_{i \in \mathcal{O}_k} \phi(p_{ik}, r) x_{ikr} \right) \geq t_{kr}^{\text{idle}}, \quad k \in \mathcal{F}, r = 1, \ldots, |\mathcal{O}_k| - 1, \tag{19}$$

that force the equality to hold, reduce the feasible region of the model by cutting off non-optimal feasible solutions.

As a curiosity, the optimal solution illustrated in Figure 2b, which corresponds to calculating a solution with minimum energy consumption from among the solutions that minimize the makespan, was calculated by substituting (1) by the objective function given by

$$\begin{aligned} \text{Minimize} \quad & \Gamma_2\, C_{\max} + \left[\sum_{k \in \mathcal{F}} \left\{ \gamma_k^{\text{proc}} \left(\sum_{i \in \mathcal{O}_k} \sum_{r=1}^{|\mathcal{O}_k|} \phi(p_{ik}, r)\, x_{ikr} \right) + \right. \right. \\ & \left. \left. \left(\gamma_k^{\text{on}} + \gamma_k^{\text{off}} \right) \left(\sum_{i \in \mathcal{O}_k} x_{ik1} + \sum_{r=1}^{|\mathcal{O}_k|-1} y_{kr} \right) + \gamma_k^{\text{idle}} \sum_{r=1}^{|\mathcal{O}_k|-1} t_{kr}^{\text{idle}} \right\} + \gamma^{\text{extra}} C_{\max} \right]. \end{aligned} \tag{20}$$

The objective function (20) corresponds to summing (1) with $C_{\max}$ multiplied by Γ_2, where Γ_2 is an upper bound on the optimal value of the energy consumption. Since all the quantities involved in an instance definition are integers, the optimal value of the makespan is an integer value. Since $C_{\max}$ appears multiplied by Γ_2, reducing $C_{\max}$ by a single unit is more advantageous than any possible reduction related to energy consumption. For that reason, the minimization of (20) results in an optimal makespan solution. The term in (20) that coincides with (1) has the role of, from among optimal makespan solutions, finding one that minimizes energy consumption.

4. Constructive heuristic

In this section we describe a greedy constructive heuristic (GCH) that schedules one operation per iteration until a feasible solution is constructed. The proposed heuristic is of the list scheduling type. This means that a measure related to the insertion of a new operation in the partial solution built so far is defined. The measure of all the operations that can be scheduled is calculated and the operation that optimizes that measure is chosen to be included in the partial solution. As the measure is related to energy consumption, the selected operation is not necessarily programmed to start as soon as possible. For this reason, the constructed schedule is not necessarily semi-active. (As all the methods considered in the present work use this constructive heuristic in one way or another, this property of the constructed solutions is inherent to all of them). The method continues until all operations have been scheduled. In the present work, the measure is related to energy consumption. Heuristics of this type have already been successfully employed in the FJS environment. See for example Birgin et al. (2014, 2015).

The heuristic builds two types of structures: (a) structures that represent the instance and will later be used by other methods and (b) structures that represent the constructed solution. Both types of structure contain redundancies, which serve to simplify the description of the heuristic and other methods later described. The structures representing the instance are:

- A directed acyclic graph $G = (V, A)$. The set of vertices V is formed by the set of operations $\mathcal{O}$ and two fictitious operations s and t. The set of edges A is formed by all edges in $\widehat{A}$, edges that exit from s to every operation i that has no precedents (i.e., i such that $(\cdot, i) \notin \widehat{A}$) and edges that exit from every operation i with no successors (i.e., i such that $(i, \cdot) \notin \widehat{A}$) to t.
- Given the directed graph $G = (V, A)$, we assume that the sets $\overleftarrow{\mathcal{N}}_i(G) \subset V$ and $\overrightarrow{\mathcal{N}}_i(G) \subset V$ with the immediate predecessors and successors of any node $i \in V$, respectively, are provided.
- We also assume that, for each $i \in V$, the sets $\overrightarrow{\mathcal{R}}_i(G)$ and $\overleftarrow{\mathcal{R}}_i(G)$ with the nodes that can be reached from i and the nodes from which i can be reached in the graph G, respectively, are also available.

The structures that represent the constructed feasible solution are:

- For each machine $k \in \mathcal{F}$, a list $Q_k = i_1^k, i_2^k, \dots i_{|Q_k|}^k$ representing the operations attributed to machine k and their order.
- For a given set of machine lists $Q = \{Q_k\}_{k \in \mathcal{F}}$, we assume that a set of edges $A_M(Q)$, known as the set of machine edges, with edges that goes from each operation in Q_k to the operation following it in Q_k, for all $k \in \mathcal{F}$, is available.
- For a given set of machine lists $Q = \{Q_k\}_{k \in \mathcal{F}}$ and $i \in \mathcal{O}$, the information $f_i(Q)$ representing the machine to which operation i is assigned is assumed to be available at constant cost.
- For each operation $i \in \mathcal{O}$, information s_i and p'_i indicating its starting time and its effective processing time, respectively.
- The E and $C_{\max}$ values of the energy consumption and the makespan of the constructed solution,

respectively.

In the description of the parameters of the heuristic, and of the methods that will follow, the set of lists Q_k for all $k \in \mathcal{F}$ is denoted by Q. The same abuse of notation occurs with all other parameters and structures. When the set of machines $\mathcal{F}$ appears as a parameter, it also includes the sets $\mathcal{F}_i$ for all $i \in \mathcal{O}$.

At each iteration, the heuristic begins by determining the set of operations $\mathcal{C}$ that corresponds to the operations whose predecessors have already been scheduled. That is, the set of operations that could be scheduled in that iteration. For each operation $i \in \mathcal{C}$ and for each machine $k \in \mathcal{F}_i$, the heuristic determines the most economical starting time, considering the options of (i) turning the machine off and on or (ii) leaving the machine on and idle before processing i. The option in which operation i is the first operation of machine k is also considered separately when an empty machine $k \in \mathcal{F}_i$ exists. From among these possibilities and from among all possible pairs operation-machine, the heuristic chooses the option that represents the lowest energy consumption and schedules it. Scheduling involves updating the aforementioned structures. The heuristic terminates when all operations have been scheduled. The heuristic is described in Algorithms 1 and 2. Algorithm 1 constructs a partial solution with zero scheduled operations and calls Algorithm 2 which receives a partially constructed solution and completes it. In Algorithm 2, the set Π represents the set of operations already scheduled. The heuristic is presented in this form because Algorithm 2 will soon be used, in the context of a local search, to complete partially constructed solutions.

In Algorithm 2, the calculation of the initial set $\mathcal{C}$ corresponds to lines 1–3. Lines 4–7 calculate the instant when each machine is free. The main loop, from lines 8 to 27 is executed as long as there are unscheduled operations. Within the loop, each operation $i \in \mathcal{C}$ and each machine $k \in \mathcal{F}_i$ are considered. To schedule an operation i on a machine $k \in \mathcal{F}_i$, there can be one or two alternatives. The alternative is only one if the machine is empty. In this case, the energy consumption is associated with turning on the machine, processing the operation and turning off the machine. If the scheduling of that operation increases the makespan of the existing partial solution, there is the extra cost of keeping the plant running longer. This is the calculation made in lines 15–16. The alternatives are two when machine k already has operations allocated to it. The two options are to keep the machine on and idle before processing operation i or to turn it off when the previous operation is completed and turn it on before processing operation i. These two options correspond to the calculations in lines 18–19 and 20–21, respectively. When there are two options, line 22 chooses the better of the two. Line 23 compares the best option for the pair (i, k) with the best of all the pairs already considered, saving the best of them. When the best pair is determined, the solution structures are updated in line 24 and in lines 25–27 the set $\mathcal{C}$ is updated. In Algorithm 2 and hereafter, the expression $(\cdot)_+$ means $\max(0, \cdot)$, while the expression $L \oplus \ell$, where L is a list and ℓ is an element, corresponds to add ℓ to the end of L.

Algorithm 1: Greedy constructive heuristic.

Input: $\mathcal{O}$, $\mathcal{F}$, p, $\widehat{A}$
Output: $G = (V, A)$, Q, s, p', E, $C_{\max}$
Function: GCH($\mathcal{O}$, $\mathcal{F}$, p, $\widehat{A}$, G, Q, s, p', E, $C_{\max}$)

$A \leftarrow \widehat{A} \cup \{(s, j) \mid (\cdot, j) \notin \widehat{A}\} \cup \{(i, t) \mid (i, \cdot) \notin \widehat{A}\}$, $V := \mathcal{O} \cup \{s, t\}$, $G := (V, A)$
Q_k is an empty list for all $k \in \mathcal{F}$, $E \leftarrow 0$, $C_{\max} \leftarrow 0$, $\Pi \leftarrow \emptyset$
PartialGCH($\mathcal{O}$, $\mathcal{F}$, p, G, Π, Q, s, p', E, $C_{\max}$)

Consider a minimalist example with two machines and 3 operations. Assume that operation 2 must precede operation 3, i.e., $\mathcal{F} = \{1, 2\}$, $\mathcal{O} = \{1, 2, 3\}$, $\widehat{A} = \{(2, 3)\}$. (Note that the precendence constraints are linear). Assume that operations 1 and 3 can only be processed by machine 1 and operation 2

Algorithm 2: Completes a partial solution with the greedy constructive heuristic.

Input: $\mathcal{O}, \mathcal{F}, p, G = (V, A), \Pi, Q, s, p', E, C_{\max}$
Output: $Q, s, p', E, C_{\max}$
Function: PartialGCH($\mathcal{O}, \mathcal{F}, p, G, \Pi, Q, s, p', E, C_{\max}$)

$\mathcal{C} \leftarrow \emptyset$
for $v \in \mathcal{O} \setminus \Pi$ **do**
  if $\overleftarrow{\mathcal{N}}_v(G) \subseteq \Pi$ **then** $\mathcal{C} \leftarrow \mathcal{C} \cup \{v\}$
for $k \in \mathcal{F}$ **do**
  $r_k^{\text{mach}} \leftarrow 0$
  if $|Q_k| \neq 0$ **then**
    Let v be the last operation of Q_k then, $r_k^{\text{mach}} \leftarrow s_v + p'_v$
while $\mathcal{C} \neq \emptyset$ **do**
  $\tilde{\Gamma} \leftarrow +\infty$
  for $v \in \mathcal{C}$ **do**
    $\mu \leftarrow \max\left\{s_j + p'_j \mid j \in \overleftarrow{\mathcal{N}}_v(G)\right\}$
    for $k \in \mathcal{F}_v$ **do**
      $\rho \leftarrow \psi_\alpha(p_{v,k}, |Q_k| + 1)$
      if $|Q_k| = 0$ **then**
        $\zeta \leftarrow \max\left\{r_k^{\text{mach}} + t_k^{\text{on}}, \mu\right\}$
        $\Gamma \leftarrow \gamma_k^{\text{proc}}\rho + \gamma^{\text{extra}}\left(\zeta + \rho + t_k^{\text{off}} - C_{\max}\right)_+ + \gamma_k^{\text{off}} + \gamma_k^{\text{on}}$
      else
        $\zeta_1 \leftarrow \max\left\{r_k^{\text{mach}}, \mu\right\}$
        $\Gamma_1 \leftarrow \gamma_k^{\text{proc}}\rho + \gamma^{\text{extra}}\left(\zeta_1 + \rho + t_k^{\text{off}} - C_{\max}\right)_+ + \gamma_k^{\text{iddle}}(\zeta_1 - (s_{i_{|Q_k|}} + p'_{i_{|Q_k|}}))$, where $Q_k = i_1, \ldots, i_{|Q_k|}$
        $\zeta_2 \leftarrow \max\left\{r_k^{\text{mach}} + t_k^{\text{off}} + t_k^{\text{on}}, \mu\right\}$
        $\Gamma_2 \leftarrow \gamma_k^{\text{proc}}\rho + \gamma^{\text{extra}}\left(\zeta_2 + \rho + t_k^{\text{off}} - C_{\max}\right)_+ + \gamma_k^{\text{off}} + \gamma_k^{\text{on}}$
        if $\Gamma_1 \leq \Gamma_2$ **then** $\zeta, \Gamma \leftarrow \zeta_1, \Gamma_1$ **else** $\zeta, \Gamma \leftarrow \zeta_2, \Gamma_2$
      if $(\Gamma, \zeta) < (\tilde{\Gamma}, \tilde{\zeta})$ **then** $\tilde{v}, \tilde{k}, \tilde{\zeta}, \tilde{\rho}, \tilde{\Gamma} \leftarrow v, k, \zeta, \rho, \Gamma$
  $Q_{\tilde{k}} \leftarrow Q_{\tilde{k}} \oplus \tilde{v}$, $s_{\tilde{v}} := \tilde{\zeta}$, $p'_{\tilde{v}} := \tilde{\rho}$, $E \leftarrow E + \tilde{\Gamma}$, $C_{\max} \leftarrow \max\{C_{\max}, s_{\tilde{v}} + p'_{\tilde{v}} + t_{\tilde{k}}^{\text{off}}\}$
  $\mathcal{C} \leftarrow \mathcal{C} \setminus \{\tilde{v}\}$, $r_{\tilde{k}}^{\text{mach}} \leftarrow s_{\tilde{v}} + p'_{\tilde{v}}$, $\Pi \leftarrow \Pi \cup \{\tilde{v}\}$
  for $j \in \overrightarrow{\mathcal{N}}_{\tilde{v}}(G)$ **do**
    if $\overleftarrow{\mathcal{N}}_j(G) \subseteq \Pi$ **then**
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{j\}$

can only be processed by machine 2, i.e., $\mathcal{F}_1 = \mathcal{F}_3 = \{1\}$, $\mathcal{F}_2 = \{2\}$, $\mathcal{O}_1 = \{1, 3\}$, $\mathcal{O}_2 = \{2\}$. For the processing times, let us consider $p_{11} = p_{31} = 10$ and $p_{22} = 20$. To simplify the example, let us assume that there is no learning effect. Assume that both machines take 6 units of time to turn on and 6 units of time to turn off, i.e., $\tau_k^{\text{on}} = \tau_k^{\text{off}} = 6$ for $k = 1, 2$. The GCH heuristic starts with an empty solution and $\mathcal{C} = \{1, 2\}$. Since operation 1 has a shorter processing time, operation 1 is assigned to machine 1. Since it takes 6 time units to turn on the machine, operation 1 is scheduled to start at time 6. In the next iteration, we have $\mathcal{C} = \{2\}$ (operation 3 does not have all its predecessors scheduled yet). Then, operation 2 is assigned to machine 2 and scheduled to start processing at time 6. Figure 3(a) shows the partial solution with operations 1 and 2 already scheduled. At iteration 3, we have $\mathcal{C} = \{3\}$. Operation 3 can only be assigned to machine 1. But here we have 2 options (shown in Figure 3(b) and Figure 3(c), respectively):

Option 1: Schedule operation 3 to start at time 26. In this case, machine 1 would be idle between the end of the processing of operation 1 at instant 16 and instant 26. Since this is not enough time to turn the machine off and on, this option incurs an energy cost of $10\gamma_1^{\text{idle}}$. In addition to that, this scheduling of operation 3 increases the makespan by 10 time units, at an additional cost of $10\gamma^{\text{extra}}$.

Option 2: Schedule operation 3 to start at instant 28. By delaying the start of operation 3 by 2 time units, we construct a non-semi-active schedule. This delay allows machine 1 to be turned off and on between the end of the processing of operation 1 and the start of the processing of operation 3, with an associated cost of $\gamma_1^{\text{off}} + \gamma_1^{\text{on}}$. Still, this scheduling increases the makespan by 12 time units, incurring an additional cost of $12\gamma^{\text{extra}}$.

If the instance data is such that $\gamma_1^{\text{off}} + \gamma_1^{\text{on}} + 12\gamma^{\text{extra}} < 10\gamma_1^{\text{idle}} + 10\gamma^{\text{extra}}$, then the heuristic chooses option 2. Otherwise, it chooses option 1.

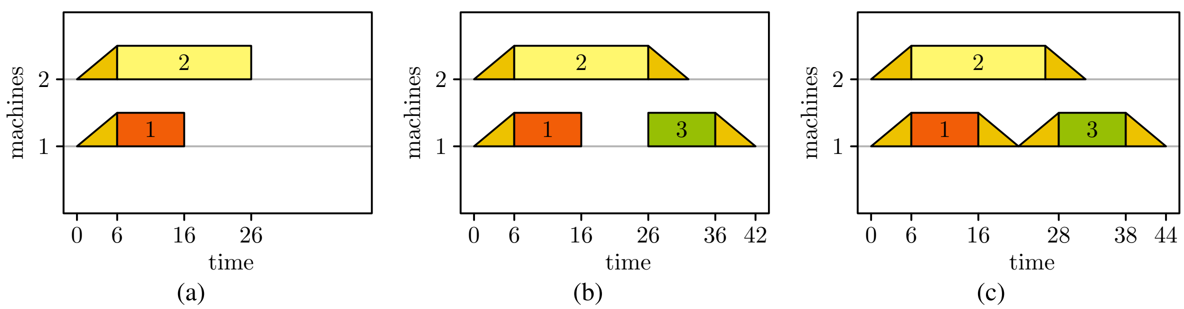


Fig. 3. Example of a non-semi-active solution computed by the constructive heuristic CGH. The graphic (a) shows the partial solution constructed after two iterations. The graphics (b) and (c) show options 1 and 2 in iteration 3.

5. Local search strategies

In this section we present two different local search strategies. Both generate neighbors by removing and reinserting an operation in the current solution, but the ways of removing and reinserting the operation are different. In the first local search, the neighborhood, called SRRN (single-operation removal and reinsertion neighborhood), is based on a move that removes and reinserts a single operation $v \in \mathcal{O}$ at a location where no cycle is formed. In the second local search, the neighborhood, called SRDRRN (single-operation removal, destruction, reinsertion, and reconstruction neighborhood), is based on a move such that if a $v \in \mathcal{O}$ operation is removed, then all its successors are also removed. Then, if v is reinserted in the place of another operation w, w and all successor operations of w are removed. The partially "destroyed" solution then needs to be reconstructed with the GCH constructive heuristic.

Let Q, s, p', E and $C_{\max}$ be the data of the current solution. In the first strategy, the neighborhood is constructed by considering one at a time, the operations $v \in \mathcal{O}$. For each operation, the operation is removed from the machine to which it is assigned, i.e., from the machine $f_v(Q)$. Let $\widehat{Q}$ be the set of machine lists representing the current solution with operation v removed. Then, an attempt is made to reinsert this same operation in all possible positions of all machines $k \in \mathcal{F}_v$, i.e., in all machines that can process operation v. Let $\widehat{Q}_k = h_1, \ldots, h_{|\widehat{Q}_k|}$ be the list of operations of a machine k in which we are trying to reinsert v. The possible positions are $r = 1, \ldots, |\widehat{Q}_k|+1$. For all possible values of r, we need to verify if the insertion is possible. What must be verified is if the insertion does not generate a cycle in the directed graph representing the solution. A cycle will be generated if any operation among $h_1, \dots, h_{r-1}$ is reachable from v in the directed graph $(V, A \cup A_M(\widehat{Q}))$. The set of operations reachable from v in this directed graph is given by $\overrightarrow{\mathcal{R}}_v((V, A \cup A_M(\widehat{Q})))$. Then, if $\{h_1, \dots, h_{r-1}\} \cap \overrightarrow{\mathcal{R}}_v((V, A \cup A_M(\widehat{Q}))) \neq \emptyset$, then the insertion of v in the r-th position of machine k will generate a cycle in the directed graph and that directed graph will not represent a feasible solution. Another way to form a cycle is when any of the operations that would remain after v reaches v, i.e., when $\{h_{r+1}, \dots, h_{|\widehat{Q}_k|}\} \cap \overleftarrow{\mathcal{R}}_v((V, A \cup A_M(\widehat{Q}))) \neq \emptyset$. Putting the two conditions together, the condition for v to be inserted at position r of the machine k is

$$\{h_1, \dots, h_{r-1}\} \cap \overrightarrow{\mathcal{R}}_v((V, A \cup A_M(\widehat{Q}))) = \emptyset \text{ and } \{h_{r+1}, \dots, h_{|\widehat{Q}_k|}\} \cap \overleftarrow{\mathcal{R}}_v((V, A \cup A_M(\widehat{Q}))) = \emptyset.$$

Let $\overline{Q}$ be the set of lists representing the current solution with the operation v removed and reinserted at position r of machine k. The processing time of v is given by $\psi_\alpha(p_{v,k}, r)$. The processing time of its successors in the list $\overline{Q}_k$ needs to be recalculated, since those operations are now one position further in the list and we are dealing with a position-based learning effect. With the allocations of operations to machines all defined and the processing times of all operations also defined, it is necessary to recalculate the starting time of each operation, the energy consumption, and the makespan of the newly constructed solution. This recalculation must be done from scratch by considering each operation in a topological order of the directed acyclic graph $(V, A \cup A_M(\overline{Q}))$. In addition, it must be decided, for each operation, whether before processing it the machine would remain on and idle or whether it would be turned off and on again. (If the operation is the first one of the machine, the machine must be simply turned on.) The construction of the whole neighborhood, including the choice of the best neighbor, is described in Algorithm 3. Algorithm 3 uses Algorithm 4 to do the reinsertion and recalculation of the structures defining the neighbor, its energy consumption, and its makespan.

Let Q, s, p', E and $C_{\max}$ be the data of the current solution. The neighborhood of the second strategy is also constructed by removing each operation and reinserting it at every possible position of each machine. The difference is that, when an operation v is removed, all operations reachable from v in the directed graph $(V, A \cup A_M(Q))$ are removed as well. Let us call $\overline{Q}$ the set of machine lists representing the current solution with v and all operations in $\overrightarrow{\mathcal{R}}_v((V, A \cup A_M(Q)))$ removed. When the operation is reinserted at position r of a machine $k \in \mathcal{F}_v$, there are two possibilities. If $r = |\overline{Q}_k| + 1$, then, since the position is empty, there is nothing else to be removed. Otherwise, if $\overline{Q}_k = i_1, \dots, i_{|\overline{Q}_k|}$, then i_r and all operations reachable from i_r in the directed graph $(V, A \cup A_M(\overline{Q}))$, i.e., all operations in $\overrightarrow{\mathcal{R}}_{i_r}((V, A \cup A_M(\overline{Q})))$, must be removed. These two mass removals are what is called "destruction". In order that the insertion of v in the r-th position of machine k generates a feasible solution, we cannot, in the removal of i_r and the nodes it reaches, remove any operation that reaches v. Otherwise v would not be ready to be reinserted because it would have unscheduled precedents. Therefore, the condition for v to be inserted in the r-th position of machine k is given by

$$r = |\overline{Q}_k| + 1 \text{ or } \left(\{i_r\} \cup \overrightarrow{\mathcal{R}}_{i_r}((V, A \cup A_M(\overline{Q})))\right) \cap \overleftarrow{\mathcal{R}}_v((V, A \cup A_M(\overline{Q}))) = \emptyset. \tag{21}$$

This condition is equivalent to

$$r = |\overline{Q}_k| + 1 \text{ or } i_r \notin \overleftarrow{\mathcal{R}}_v((V, A \cup A_M(\overline{Q}))). \tag{22}$$

The equivalence between (21) and (22) holds because if the intersection at (21) is empty then $i_r \notin \overleftarrow{\mathcal{R}}_v((V, A \cup A_M(\overline{Q})))$ and if i_r does not reach v then no operation reachable by i_r can reach v. Otherwise, by transitivity, i_r would reach v. If condition (22) is satisfied, v is reinserted. Its processing time is calculated, taking into consideration the learning effect, and it is decided as it was done before, what should be done with the machine (between leaving it on and idle or turning it off and on) to minimize energy consumption. This generates a partial solution that is then completed with the constructive heuristic of Algorithm 2. This is the phase called "reconstruction". Algorithm 5 describes the generation of all neighbors of the current solution, including the choice of the best of them. Algorithm 5 makes use of Algorithm 6 for the two destructions that precede the reconstruction.

Algorithm 7 describes the local search that, using exclusively one of the two neighborhoods, iterates until it finds a solution that is better than all its neighbors. Algorithm 7 already receives as input an initial feasible solution represented by Q, s, p', E, $C_{\max}$ and the instance data represented by $\mathcal{O}$, $\mathcal{F}$, p, G. For this reason, when the local search is used as a stand-alone method, we assume that before making a call to the local search, a call to the constructive heuristic GCH (Algorithm 1) is made giving as input the data $\mathcal{O}$, $\mathcal{F}$, p, $\widehat{A}$ of the instance to which the local search is to be applied. This call to GCH returns the graph G representing the instance and a feasible solution represented by Q, s, p', E, $C_{\max}$. For this reason, from now on, we name the method that consists of calculating an initial solution using the constructive heuristic GCH and applying the local search with the SRRN neighborhood as GCH-LS-SRRN. This method corresponds to the combination of Algorithms 1, 2, 3, 4, and 7. Analogously, we call GCH-LS-SRDRR the method using the local search with the SRDRR neighborhood, which corresponds to the combination of Algorithms 1, 2, 5, 6, and 7.

Algorithm 3: Given a current approximation of a solution, returns the best neighbor of a neighborhood based on removing and reinserting a single operation.

Input: $\mathcal{O}$, $\mathcal{F}$, p, $G = (V, A)$, Q, s, p', E, $C_{\max}$
Output: Q, s, p', E, $C_{\max}$
Function: SRRN($\mathcal{O}$, $\mathcal{F}$, p, G, Q, s, p', E, $C_{\max}$)

$\tilde{E} \leftarrow +\infty$
for $v \in \mathcal{O}$ **do**
    $\widehat{Q}, \widehat{s}, \widehat{p}', \widehat{E}, \widehat{C}_{\max} \leftarrow Q, s, p', E, C_{\max}$
    Let $\eta = f_v(\widehat{Q})$ be the machine to which v is assigned and ℓ the position of v in $\widehat{Q}_\eta$, i.e., $\widehat{Q} = j_1, \dots j_{|\widehat{Q}_\eta|}$ and $j_\ell = v$
    for $\lambda = \ell + 1, \dots, |\widehat{Q}_\eta|$ **do** $\widehat{p}'_{j_\lambda} \leftarrow \psi_\alpha(p_{j_\lambda,\eta}, \lambda - 1)$
    Remove v from $\widehat{Q}_\eta$
    for $k \in \mathcal{F}_v$ **do**
        Let $\widehat{Q}_k = h_1, \dots, h_{|\widehat{Q}_k|}$
        for $r \in \{1, \dots, |\widehat{Q}_k| + 1\}$ **do**
            if $\{h_1, \dots, h_{r-1}\} \cap \overrightarrow{\mathcal{R}}_v(V, A \cup A_M(\widehat{Q})) = \emptyset$ *and* $\{h_r, \dots, h_{|\widehat{Q}_k|}\} \cap \overleftarrow{\mathcal{R}}_v(V, A \cup A_M(\widehat{Q})) = \emptyset$ **then**
                $\overline{Q}, \overline{s}, \overline{p}', \overline{E}, \overline{C}_{\max} \leftarrow \widehat{Q}, \widehat{s}, \widehat{p}', \widehat{E}, \widehat{C}_{\max}$
                Reinsert($\mathcal{O}$, $\mathcal{F}$, p, G, v, k, r, $\overline{Q}$, $\overline{s}$, $\overline{p}'$, $\overline{E}$, $\overline{C}_{\max}$)
                if $\overline{E} < \tilde{E}$ **then** $\tilde{Q}, \tilde{s}, \tilde{p}', \tilde{E}, \tilde{C}_{\max} \leftarrow \overline{Q}, \overline{s}, \overline{p'}, \overline{E}, \overline{C}_{\max}$
if $\tilde{E} < E$ **then** $Q, s, p', E, C_{\max} \leftarrow \tilde{Q}, \tilde{s}, \tilde{p}', \tilde{E}, \tilde{C}_{\max}$

6. Metaheuristics

In this section, we describe the three metaheuristics considered, namely, greedy randomized adaptive search procedure (GRASP), simulated annealing (SA) and general variable neighborhood search

Algorithm 4: Constructs the neighbor of the current approximation of a solution that is obtained by reinserting the removed operation v at the r-th position of machine k.

Input: $\mathcal{O}, \mathcal{F}, p, G = (V, A), v, k, r, \overline{Q}, \overline{s}, \overline{p}', \overline{E}, \overline{C}_{\max}$
Output: $\overline{Q}, \overline{s}, \overline{p}', \overline{E}, \overline{C}_{\max}$
Function: Reinsert($\mathcal{O}, \mathcal{F}, p, G, v, k, r, \overline{Q}, \overline{s}, \overline{p}', \overline{E}, \overline{C}_{\max}$)

Let $\overline{Q}_k = i_1, \dots, i_{|\overline{Q}_k|}$
for $\lambda \in \{r, \dots, |\overline{Q}_k|\}$ **do** $\overline{p}'_{i_\lambda} \leftarrow \psi_\alpha(p_{i_\lambda,k}, \lambda + 1)$
Insert v in $\overline{Q}_k$ at position r, $\overline{p}'_v \leftarrow \psi_\alpha(p_{v,k}, r)$
for $u \in V$ **do** $d_u \leftarrow |\overleftarrow{\mathcal{N}}_u(G_v^+ = (V, A \cup A_M(\overline{Q})))|$
$\mathcal{U} \leftarrow \overrightarrow{\mathcal{N}}_s(G), E \leftarrow 0, C_{\max} \leftarrow 0$
while $\mathcal{U} \neq \emptyset$ **do**
  Select a vertex u from $\mathcal{U}$, let $\eta = f_u(\overline{Q})$ and $\mathcal{U} \leftarrow \mathcal{U} \setminus \{u\}$
  $\mu \leftarrow \max\left\{s_j + p'_j \mid j \in \overleftarrow{\mathcal{N}}_u(G_v^+ = (V, A \cup A_M(\overline{Q})))\right\}$
  if $(\cdot, u) \notin A_M(\overline{Q})$ **then**
    $s_u \leftarrow \max\left\{t_\eta^{\text{on}}, \mu\right\}$
    $\Gamma \leftarrow \gamma_\eta^{\text{proc}} p'_u + \gamma^{\text{extra}} \left(s_u + p'_u + t_\eta^{\text{off}} - C_{\max}\right)_+ + \gamma_\eta^{\text{off}} + \gamma_\eta^{\text{on}}$
  else
    Let w be such that $(w, u) \in A_M(\overline{Q})$
    $\zeta_1 \leftarrow \max\left\{s_w + p'_w, \mu\right\}$
    $\Gamma_1 \leftarrow \gamma_\eta^{\text{proc}} p'_u + \gamma^{\text{extra}} \left(\zeta_1 + p'_u + t_\eta^{\text{off}} - C_{\max}\right)_+ + \gamma_\eta^{\text{iddle}}(\zeta_1 - (s_w + p'_w))$
    $\zeta_2 \leftarrow \max\left\{s_w + p'_w + t_\eta^{\text{off}} + t_\eta^{\text{on}}, \mu\right\}$
    $\Gamma_2 \leftarrow \gamma_\eta^{\text{proc}} p'_u + \gamma^{\text{extra}} \left(\zeta_2 + p'_u + t_\eta^{\text{off}} - C_{\max}\right)_+ + \gamma_\eta^{\text{off}} + \gamma_\eta^{\text{on}}$
    if $\Gamma_1 \leq \Gamma_2$ **then** $s_u, \Gamma \leftarrow \zeta_1, \Gamma_1$ **else** $s_u, \Gamma \leftarrow \zeta_2, \Gamma_2$
  $E \leftarrow E + \Gamma$
  $C_{\max} \leftarrow \max\left\{C_{\max}, s_u + p'_u + t_\eta^{\text{off}}\right\}$
  for $w \in \overrightarrow{\mathcal{N}}_u(G_v^+ = (V, A \cup A_M(\overline{Q})))$ **do**
    $d_w \leftarrow d_w - 1$
    if $d_w = 0$ *and* $w \neq t$ **then** $\mathcal{U} \leftarrow \mathcal{U} \cup \{w\}$

(GVNS).

GRASP (Feo and Resende, 1995) is described in Algorithm 8 and follows the basic scheme. It starts by initializing the incumbent with the solution given by the GCH constructive heuristic (Algorithms 1–2). It then iterates by constructing an initial solution with a randomized version of the GCH constructive heuristic and performing a local search starting from the constructed initial solution. The local search corresponds to Algorithm 7 and can use either the SRRN neighborhood (Algorithms 3–4) or the SRDRR neighborhood (Algorithms 5–6), resulting in two versions of GRASP that we call GRASP-LS-SRRN and GRASP-LS-SRDRR, respectively. It remains to explain the randomization of the constructive heuristic GCH. The GCH heuristic schedules one operation per iteration until all operations are scheduled. At each iteration, it checks which operations can be scheduled (because all their precedents are already scheduled). This set of operations is called $\mathcal{C} \subseteq \mathcal{O}$. For each operation $v \in \mathcal{C}$ and for each machine $k \in \mathcal{F}_v$, it checks the best possible schedule and selects the pair (v, k) with the lowest energy consumption. In the randomized version, all (v, k) pairs with their respective energy consumption are stored in a list of candidates $\mathcal{L}$ and one pair is drawn from among those $\max\{1, \lfloor \alpha |\mathcal{L}| \rfloor\}$ pairs with the lowest energy consumption. The drawn pair is scheduled in that iteration. This randomization actually affects the PartialGCH routine described in Algorithm 2. We call RandomizedPartialGCH the randomized version of PartialGCH and call RandomizedGCH the GCH routine (Algorithm 1) that uses RandomizedPartialGCH instead of PartialGCH. The parameter $\alpha \in [0, 1] \subset \mathbb{R}$ is the only parameter of GRASP.

Algorithm 5: Given a current approximation of a solution, returns the best neighbor of a neighborhood based on removing a single operation, destroying, reinserting, and reconstructing.

Input: $\mathcal{O}, \mathcal{F}, p, G=(V,A), Q, s, p', E, C_{\max}$
Output: $Q, s, p', E, C_{\max}$
Function: SRDRR($\mathcal{O}, \mathcal{F}, p, G, Q, s, p', E, C_{\max}$)

$\tilde{E} \leftarrow \infty$
for $v \in \mathcal{O}$ **do**
  $\overline{Q}, \overline{s}, \overline{p}', \overline{E}, \overline{C}_{\max} \leftarrow Q, s, p', E, C_{\max}$
  $\overline{\Pi} \leftarrow \mathcal{O} \setminus (\overrightarrow{\mathcal{R}}_v((V, A \cup A_M(\overline{Q}))) \cup \{v\})$
  Unschedule($\mathcal{O}, \mathcal{F}, p, G, v, \overline{Q}, \overline{s}, \overline{p}', \overline{E}, \overline{C}_{\max}$)
  $\mu \leftarrow \max\{s_j + p'_j \mid j \in \overleftarrow{\mathcal{N}}_v((V, A \cup A_M(\overline{Q})))\}$
  for $k \in \mathcal{F}_v$, $r \in \{1, \dots, |\overline{Q}_k| + 1\}$ **do**
    Let $\overline{Q}_k = i_1, \dots, i_{|\overline{Q}_k|}$
    if $r = |\overline{Q}_k| + 1$ *or* $i_r \notin \overleftarrow{\mathcal{R}}_v((V, A \cup A_M(\overline{Q})))$ **then**
      $\widehat{Q}, \widehat{s}, \widehat{p}', \widehat{E}, \widehat{C}_{\max} \leftarrow \overline{Q}, \overline{s}, \overline{p}', \overline{E}, \overline{C}_{\max}$
      if $r \neq |\widehat{Q}_k| + 1$ **then**
        $\widehat{\Pi} \leftarrow \overline{\Pi} \setminus (\overrightarrow{\mathcal{R}}_{i_r}((V, A \cup A_M(\widehat{Q})) \cup \{i_r\})$
        Unschedule($\mathcal{O}, \mathcal{F}, p, G, i_r, \widehat{Q}, \widehat{s}, \widehat{p}', \widehat{E}, \widehat{C}_{\max}$)
      $\rho \leftarrow \psi_\alpha(p_{v,k}, r)$
      if $|\widehat{Q}_k| = 0$ **then**
        $\zeta \leftarrow \max\left\{t_k^{\text{on}}, \mu\right\}$
        $\Gamma \leftarrow \gamma_k^{\text{proc}} \rho + \gamma^{\text{extra}} \left(\zeta + \rho + t_k^{\text{off}} - \widehat{C}_{\max}\right)_+ + \gamma_k^{\text{off}} + \gamma_k^{\text{on}}$
      else
        $\zeta_1 \leftarrow \max\left\{\widehat{s}_{i_{r-1}} + \widehat{p}'_{i_{r-1}}, \mu\right\}$
        $\Gamma_1 \leftarrow \gamma_k^{\text{proc}} \rho + \gamma^{\text{extra}} \left(\zeta_1 + \rho + t_k^{\text{off}} - \widehat{C}_{\max}\right)_+ + \gamma_k^{\text{iddle}}(\zeta_1 - (s_{i_{r-1}} + p'_{i_{r-1}}))$
        $\zeta_2 \leftarrow \max\left\{\widehat{s}_{i_{r-1}} + \widehat{p}'_{i_{r-1}} + t_k^{\text{off}} + t_k^{\text{on}}, \mu\right\}$
        $\Gamma_2 \leftarrow \gamma_k^{\text{proc}} \rho + \gamma^{\text{extra}} \left(\zeta_2 + \rho + t_k^{\text{off}} - \widehat{C}_{\max}\right)_+ + \gamma_k^{\text{off}} + \gamma_k^{\text{on}}$
        if $(\Gamma_1, \zeta_1) \leq (\Gamma_2, \zeta_2)$ **then** $\zeta, \Gamma \leftarrow \zeta_1, \Gamma_1$ **else** $\zeta, \Gamma \leftarrow \zeta_2, \Gamma_2$
      $\widehat{Q}_k \leftarrow \widehat{Q}_k \oplus v$, $\widehat{s}_v := \zeta$, $\widehat{p}'_v := \rho$, $\widehat{E} \leftarrow \widehat{E} + \Gamma$, $\widehat{C}_{\max} \leftarrow \max\{\widehat{C}_{\max}, \widehat{s}_v + \widehat{p}'_v + t_k^{\text{off}}\}$
      $\widehat{\Pi} \leftarrow \widehat{\Pi} \cup \{v\}$
      PartialGCH($\mathcal{O}, \mathcal{F}, p, G, \widehat{\Pi}, \widehat{Q}, \widehat{s}, \widehat{p}', \widehat{E}, \widehat{C}_{\max}$)
      if $\widehat{E} < \tilde{E}$ **then** $\tilde{Q}, \tilde{s}, \tilde{p}', \tilde{E}, \tilde{C}_{\max} \leftarrow \widehat{Q}, \widehat{s}, \widehat{p}', \widehat{E}, \widehat{C}_{\max}$
if $\tilde{E} < E$ **then** $Q, s, p', E, C_{\max} \leftarrow \tilde{Q}, \tilde{s}, \tilde{p}', \tilde{E}, \tilde{C}_{\max}$

SA (Kirkpatrick et al., 1983) is described in Algorithm 10 and also follows its basic scheme. Each iteration consists of constructing a perturbation of the current solution, which is accepted or not with the usual test that depends on the current temperature. The temperature starts at $t_0 \in \mathbb{R}_{>0}$, goes to $t_f \leq t_0$ and is updated at every $j_{\max} \in \mathbb{Z}_{>0}$ iterations by multiplying it by $\delta \in (0,1) \subset \mathbb{R}$. The perturbation of the current solution is performed by the Shake routine. We consider two versions of the Shake routine, one based on the SRRN neighborhood and another based on the SRDRR neighborhood. In the SRRN neighborhood, each operation-machine pair (v,k) with $v \in \mathcal{O}$ and $k \in \mathcal{F}_v$ is considered. The operation v is removed from the machine to which it was assigned and reinserted in the positions $r = 1, \dots, |Q_k|+1$ of the list Q_k of machine k that do not generate cycles, i.e., that correspond to feasible solutions. Of all possible combinations of v, k and r, the one with the lowest energy consumption is chosen. The Shake based on the SRRN neighborhood consists of drawing an operation $v \in \mathcal{O}$ and then drawing a

Algorithm 6: Unschedule operation v and all its successors's in $(V, A \cup A_M(Q))$

Input: $\mathcal{O}$, $\mathcal{F}$, p, $G = (V, A)$, v, Q, s, p', E, $C_{\max}$
Output: Q, s, p', E, $C_{\max}$
Function: Unschedule($\mathcal{O}$, $\mathcal{F}$, p, G, v, Q, s, p', E, $C_{\max}$)

Let $\mathcal{W} = w_1, \ldots, w_{|V|}$ be a topological order of the operations in $V \setminus \{s, t\}$ according to the directed graph $(V, B \cup B_M(Q))$, where $B = \{(j, i) \mid (i, j) \in A\}$ and $B_M(Q) = \{(j, i) \mid (i, j) \in A_M(Q)\}$. Let $v = w_\ell$.
for $u = w_1, \ldots, w_\ell$ **do**
    if $u \in \overrightarrow{\mathcal{R}}_v((V, A \cup A_M(Q))) \cup \{v\}$ **then**
        Let $\eta = f_u(Q)$. Delete u from Q_η and $E \leftarrow E - \gamma_\eta^{\text{proc}} p'_u$.
        if $|Q_\eta| = 0$ **then** $E \leftarrow E - \gamma_\eta^{\text{on}} + \gamma_\eta^{\text{off}}$
        else $E \leftarrow E - \min\left\{\gamma_\eta^{\text{on}} + \gamma_\eta^{\text{off}}, \gamma_\eta^{\text{idle}}(s_u - (s_{i_{|Q_\eta|}} + p'_{i_{|Q_\eta|}}))\right\}$, where $Q_\eta = i_1, \ldots, i_{|Q_\eta|}$
        if $s_u + p'_u + t_\eta^{\text{off}} = C_{\max}$ **then**
            $C'_{\max} \leftarrow 0$
            for $k \in \mathcal{F}$ *such that* $|Q_k| > 0$ **do**
                $C'_{\max} \leftarrow \max\left\{C'_{\max}, s_{i_{|Q_k|}} + p'_{i_{|Q_k|}} + t_k^{\text{off}}\right\}$, where $Q_k = i_1, \ldots, i_{|Q_k|}$
            $E \leftarrow E - \gamma^{\text{extra}}(C_{\max} - C'_{\max})$
            $C_{\max} \leftarrow C'_{\max}$

Algorithm 7: Local search strategy based on the single reinsertion neighborhood.

Input: $\mathcal{O}$, $\mathcal{F}$, p, G
Output: Q, s, p', E, $C_{\max}$
Function: LocalSearch($\mathcal{O}$, $\mathcal{F}$, p, G, Q, s, p', E, $C_{\max}$)

do
    $\overline{Q}, \overline{s}, \overline{p}', \overline{E}, \overline{C}_{\max} \leftarrow Q, s, p', E, C_{\max}$
    SRRN($\mathcal{O}$, $\mathcal{F}$, p, G, $\overline{Q}$, $\overline{s}$, $\overline{p}'$, $\overline{E}$, $\overline{C}_{\max}$) (or SRDRR($\mathcal{O}$, $\mathcal{F}$, p, G, $\overline{Q}$, $\overline{s}$, $\overline{p}'$, $\overline{E}$, $\overline{C}_{\max}$))
    $\Delta E \leftarrow E - \overline{E}$
    if $\Delta E > 0$ **then**
        $Q, s, p', E, C_{\max} \leftarrow \overline{Q}, \overline{s}, \overline{p}', \overline{E}, \overline{C}_{\max}$
while $\Delta E > 0$

Algorithm 8: Greedy randomized adaptive search procedure

Input: $\mathcal{O}$, $\mathcal{F}$, p, $\widehat{A}$, α
Output: $Q^\star$, $s^\star$, $p'^\star$, $E^\star$, $C^\star_{\max}$
Function: GRASP($\mathcal{O}$, $\mathcal{F}$, p, $\widehat{A}$, α, $Q^\star$, $s^\star$, $p'^\star$, $E^\star$, $C^\star_{\max}$)

GCH($\mathcal{O}$, $\mathcal{F}$, p, $\widehat{A}$, G, $Q^\star$, $s^\star$, $p'^\star$, $E^\star$, $C^\star_{\max}$)
while *the stopping criterion is not satisfied* **do**
    RandomizedGCH($\mathcal{O}$, $\mathcal{F}$, p, G, α, Q, s, p', E, $C_{\max}$)
    LocalSearch($\mathcal{O}$, $\mathcal{F}$, p, G, Q, s, p', E, $C_{\max}$)
    if $E < E^\star$ **then**
        $Q^\star, s^\star, p'^\star, E^\star, C^\star_{\max} \leftarrow Q, s, p', E, C_{\max}$

machine $k \in \mathcal{F}_v$. For that pair, the positions $r = 1, \ldots, |Q_k| + 1$ corresponding to a feasible solution are determined and, among these, one is drawn at random. Let E be the energy of the current solution and $E(v, k, r)$ be the energy of the reinsertion of operation v at the r-th position of machine k. The first triple that satisfies the acceptance criterion $E(v, k, r) \leq E(1 + \epsilon)$, where $\epsilon \in \mathbb{R}_{>0}$ is a given parameter, is accepted. The number of draws is limited to $\sum_{i=1}^{|\mathcal{O}|} |\mathcal{F}_{v_i}| \leq |\mathcal{O}||\mathcal{F}|$, where v_i is the i-th drawn operation.

If no triple satisfies the acceptance criterion, the routine returns the current solution. The Shake routine was developed in this way to be used also in the context of other metaheuristics. In the particular case of SA, the acceptance criterion is an intrinsic part of the metaheuristic. Thus, we consider $\epsilon = +\infty$ and the first triple drawn is returned. The Shake based on the SRDRR neighborhood follows exactly the same idea. The only difference is that after the destruction, to introduce more randomness into the process, the partial solutions are reconstructed with the RandomizedPartialGCH routine instead of the PartialGCH routine. We call the SA using the Shake routine based on the SRRN neighborhood of SA-SRRN and the SA using the Shake routine based on the SRDRR neighborhood of SA-SRDRR. In addition to the aforementioned parameters t_0, t_f, δ, and $j_{\max}$, the SA-SRRN has the parameter ϵ for the Shake while the SA-SRDRR has the parameters ϵ and α for the Shake.

In fact, t_0 is not a parameter of the SA. Let $E > 0$ be the energy consumption of the current solution and $\overline{E}$ be the energy consumption of a candidate solution. The candidate solution is accepted as the new current solution if $e^{-\Delta E/t} \geq \rho$ where $\Delta E = (\overline{E} - E)/E$ and $\rho \in [0, 1]$ is a random number. If $\overline{E} \leq E$, then $\Delta E \leq 0$, $-\Delta E/t \geq 0$ for all $t > 0$ and $e^{-\Delta E/t} \geq 1 \geq \rho$ for any $\rho \in [0, 1]$. This means that if the candidate solution is better than or equal to the current solution, then it will meet the acceptance criterion. If $\overline{E} > E$ then $\Delta E > 0$ and the acceptance criterion is satisfied if $\Delta E \leq -t \ln(\rho)$. As a result, candidate solutions $\overline{E}$ satisfying $\Delta E = (\overline{E} - E)/E \leq \theta$, i.e. that are up to $100\%\,\theta$ worse than the current solution E, will satisfy the acceptance test if t is such that $\theta \leq -t \ln(\rho)$, where $\rho \in [0, 1]$ is a random number. To satisfy this with probability ν, all that is needed is that $t = -\theta/\ln(\nu)$. For this reason, we choose to consider the relative difference $\Delta E = (\overline{E} - E)/E$ instead of the absolute difference $\Delta E = \overline{E} - E$ in the SA implementation. Moreover, the choice of t_0 is given by $t_0 = -\theta/\ln(\nu)$, where $\theta \in \mathbb{R}_{>0}$ and $\nu \in (0, 1) \subset \mathbb{R}$ are dimensionless parameters to be determined. See Johnson et al. (1989) for ways to select the initial temperature in SA.

Algorithm 9: Simulated Annealing

Input: $\mathcal{O}, \mathcal{F}, p, \widehat{A}, t_0, t_f, \delta, j_{\max}, \alpha$
Output: $Q^\star, s^\star, p'^\star, E^\star, C^\star_{\max}$
Function: SA($\mathcal{O}, \mathcal{F}, p, \widehat{A}, t_0, t_f, \delta, j_{\max}, \alpha, \epsilon, Q^\star, s^\star, p'^\star, E^\star, C^\star_{\max}$)

GCH($\mathcal{O}, \mathcal{F}, p, \widehat{A}, G, Q, s, p', E, C_{\max}$)
$Q^\star, s^\star, p'^\star, E^\star, C^\star_{\max} \leftarrow Q, s, p', E, C_{\max}$
$t \leftarrow t_0$
while *the stopping criterion is not satisfied* **do**
  for $j = 1, \ldots, j_{\max}$ **do**
    $\overline{Q}, \overline{s}, \overline{p}', \overline{E}, \overline{C}_{\max} \leftarrow Q, s, p', E, C_{\max}$
    Shake($\mathcal{O}, \mathcal{F}, p, G, \alpha, \epsilon \equiv +\infty, \overline{Q}, \overline{s}, \overline{p}', \overline{E}, \overline{C}_{\max}$)
    $\Delta E \leftarrow (\overline{E} - E)/E$
    if $e^{-\Delta E/t} \geq \rho$, *where* $\rho \in [0, 1]$ *is a random number* **then**
      $Q, s, p', E, C_{\max} \leftarrow \overline{Q}, \overline{s}, \overline{p}', \overline{E}, \overline{C}_{\max}$
      if $\overline{E} < E^\star$ **then**
        $Q^\star, s^\star, p'^\star, E^\star, C^\star_{\max} \leftarrow \overline{Q}, \overline{s}, \overline{p}', \overline{E}, \overline{C}_{\max}$
  $t \leftarrow \max\{\delta\, t, t_f\}$

The GVNS (Hansen et al., 2018) is described in Algorithm 10 and corresponds exactly to (Hansen et al., 2018, Alg.8, p.64). It is a generalized version of VNS because it uses different neighborhoods in both Shake and local search. In classic VNS, different neighborhoods are used to generate initial points from which a local search (which always uses the same type of neighborhood) is launched. In GVNS, local searches with different neighborhoods are also considered. In the considered implementation, we have $j_{\max} = 2$, $j = 1$ corresponds to the Shake based on the SRRN neighborhood and $j = 2$ corresponds to the Shake based on the SRDRR neighborhood. Similarly, $k_{\max} = 2$ and, with $k = 1$, the NeighborhoodSearch routine corresponds to the SRRN routine, while, with $k = 2$, the NeighborhoodSearch routine corresponds to the SRDRR routine.

Algorithm 10: General variable neighborhood search

Input: $\mathcal{O}, \mathcal{F}, p, \widehat{A}, \alpha, \epsilon, k_{\max}, j_{\max}$
Output: $Q, s, p', E, C_{\max}$
Function: GVNS($\mathcal{O}, \mathcal{F}, p, \widehat{A}, \alpha, \epsilon, k_{\max}, j_{\max}, Q, s, p', E, C_{\max}$)
GCH($\mathcal{O}, \mathcal{F}, p, \widehat{A}, G, Q, s, p', E, C_{\max}$)
while *the stopping criterion is not satisfied* **do**
    $k \leftarrow 1$
    while $k \leq k_{\max}$ **do**
        $\overline{Q}, \overline{s}, \overline{p}', \overline{E}, \overline{C}_{\max} \leftarrow Q, s, p', E, C_{\max}$
        Shake($\mathcal{O}, \mathcal{F}, p, G, k, \alpha, \epsilon, \overline{Q}, \overline{s}, \overline{p}', \overline{E}, \overline{C}_{\max}$)
        $j \leftarrow 1$
        while $j \leq j_{\max}$ **do**
            $\widehat{Q}, \widehat{s}, \widehat{p}', \widehat{E}, \widehat{C}_{\max} \leftarrow \overline{Q}, \overline{s}, \overline{p}', \overline{E}, \overline{C}_{\max}$
            NeighborhoodSearch($\mathcal{O}, \mathcal{F}, p, G, j, \widehat{Q}, \widehat{s}, \widehat{p}', \widehat{E}, \widehat{C}_{\max}$)
            if $\widehat{E} < \overline{E}$ **then**
                $j \leftarrow 1$
                $\overline{Q}, \overline{s}, \overline{p}', \overline{E}, \overline{C}_{\max} \leftarrow \widehat{Q}, \widehat{s}, \widehat{p}', \widehat{E}, \widehat{C}_{\max}$
            else $j \leftarrow j + 1$
        if $\overline{E} < E$ **then**
            $k \leftarrow 1$
            $Q, s, p', E, C_{\max} \leftarrow \overline{Q}, \overline{s}, \overline{p}', \overline{E}, \overline{C}_{\max}$
        else $k \leftarrow k + 1$

7. Numerical experiments

In this section we present numerical experiments to evaluate the introduced local searches and the considered metaheuristics. In Section 7.1 we compare the local searches using the two introduced neighborhood variants. In Section 7.2 we calibrate the parameters of the metaheuristics and compare their performance. In Section 7.3 we consider the best performing metaheuristics and compare the quality of the solutions they achieve with an exact solver as a reference.

The local search and the metaheuristics were implemented in C++ programming language. The code was compiled using g++ 10.2.1. The experiments were carried out in an Intel i9-12900K (12th Gen) processor operating at 5.200GHz and 128 GB of RAM.

In the experiments of Sections 7.1 and 7.2, we consider the 50 large-sized instances of the FJS with nonlinear routes introduced in Birgin et al. (2014). To the instances, we must add the data related to energy consumption. For each instance, following Wu et al. (2019), we draw, with discrete uniform distribution $\gamma^{\text{extra}} \in [100, 2500]$ and $\gamma_k^{\text{proc}} \in [80, 100]$, $\gamma_k^{\text{idle}} \in [5, 20]$, $\tau_k^{\text{on}} \in [10, 30]$, $\gamma_k^{\text{on}} \in [50, 90]$, $\tau_k^{\text{off}} \in [10, 20]$, and $\gamma_k^{\text{off}} \in [50, 90]$ for all $k \in \mathcal{F}$. For each machine $k \in \mathcal{F}$, we set $\tau_k^{\text{idle}} = \max\{\tau_k^{\text{on}} + \tau_k^{\text{off}}, \lfloor(\gamma_k^{\text{on}} + \gamma_k^{\text{off}})/\gamma_k^{\text{idle}}\rfloor\}$. This means that a machine is not allowed to be idle for a longer time than is necessary to turn the machine off and on if this time consumes more than is consumed by turning the machine off and on. This is a condition naturally satisfied by an optimal solution, but this constraint helps in solving the model by an exact method. Following Araujo et al. (2024a,b), we consider learning rates $\alpha \in \{0.1, 0.2, 0.3\}$, totaling 150 instances. Details of instance characteristics can be found in (Birgin et al., 2024, Table S1). It is worth noting that the largest instance has almost 74,000 binary variables and almost 4 million constraints. The instances and solutions found are available at http://www.ime.usp.br/~egbirgin/ for future reference.

7.1. Experiments with the local search strategies

In this section we show the results of applying the local searches LS-SRRN and LS-SRDRR to the considered 150 large-sized instances. Details of the results obtained by applying each method to each instance can be found in (Birgin et al., 2024, Table S2). Table 3 shows a summary of the results. In the table we show the average energy consumption of the solutions found by the GCH constructive heuristic and, for each of the two local searches, the average energy consumption of the solutions found, the average number of iterations and the average CPU time in *miliseconds*. The number of best solutions found and the average gap to the solution found by the GCH constructive heuristic are also displayed. GCH spends, on average, 0.37 milliseconds per instance and in no instance it takes more than 2 milliseconds.

Instances		GCH	LS-SRRN					LS-SRDRR				
type	α	$\overline{E}$	#wins	$\overline{gap}(\%)$	$\overline{E}$	$\overline{\#it}$	$\overline{Time}$	#wins	$\overline{gap}(\%)$	$\overline{E}$	$\overline{\#it}$	$\overline{Time}$
DA	0.1	1,245,668.07	6	-6.21	1,157,385.23	20.23	849.92	24	-9.69	1,109,529.97	7.43	2,272.45
	0.2	1,043,387.03	6	-4.49	996,649.17	17.10	723.87	24	-8.29	950,481.00	6.60	2,045.01
	0.3	906,801.00	3	-4.65	864,704.50	15.27	540.29	27	-9.63	818,844.27	6.77	2,027.10
Y	0.1	1,622,068.05	6	-4.30	1,553,797.40	20.35	8,745.91	14	-9.11	1,487,848.90	6.35	6,333.98
	0.2	1,436,712.65	3	-4.97	1,370,043.95	20.60	9,280.43	17	-9.38	1,315,247.10	6.05	4,791.40
	0.3	1,280,518.20	2	-4.70	1,222,768.95	20.40	8,266.10	18	-9.25	1,174,502.65	6.95	7,043.83
All			26	-4.93				124	-9.22			

Table 3
Summary of the results obtained by applying the constructive heuristic GCH and the local searches LS-SRRN and LS-SRDRR to the 50 large-sized instances based on the instances introduced in Birgin et al. (2014), with learning rates $\alpha \in \{0.1, 0.2, 0.3\}$.

It is clear that LS-SRRN iterations (search in a neighborhood) are cheaper than LS-SRDRR iterations. At the same time, LS-SRRN is expected to do more iterations than LS-SRDRR. Experiments confirm that the former does, on average, about three times as many iterations as the latter. Yet, in DA-type instances, LS-SRRN takes three times less time than the latter, suggesting that the iterations of LS-SRRN are an order of magnitude faster. The same is not confirmed for Y-type instances. In those instances, the ratio between the number of iterations of the two local searches remains the same, but LS-SRDRR takes less time than LS-SRNN. The explanation for this is the level of flexibility and routes' nonlinearity of the two instance types. Instances of type DA have higher flexibility levels than instances of type Y and that justifies that generating all neighbors of a solution is more expensive. Overall, LS-SRRN improves the initial solution constructed by the GCH constructive heuristic by 4.93% while LS-SRDRR improves by 9.22%. LS-SRDRR is also superior to LS-SRRN in number of best solutions found. The CPU times used by the two local searches show that instances with more than 100 operations might be challenging. The application of local searches, which use the best neighbor technique and at each iteration inspect the complete neighborhood of the current solution, is slightly demanding from a computational cost point of view. Which of the two local searches, or their neighborhoods, will be better when embedded in the context of a metaheuristic is something to be analyzed in the next section.

7.2. *Experiments with the metaheuristics*

In this section we present results of applying GRASP-LS-SRRN, GRASP-LS-SRDRR, SA-SRRN, SA-SRDRR and GVNS to the 150 large-sized instances. We calibrated the five methods using the irace package (López-Ibáñez et al., 2016). Let $\Lambda = \{0.1, 0.2, \dots, 0.5\}$ and $\Theta = \{0, 0.05, 0.10, \dots, 0.95, 0.99\}$. For the two versions of GRASP we considered $\alpha \in \Lambda$. For the GVNS, we considered $\alpha \in \Lambda$ and $\epsilon \in \Theta$. For the two versions of the SA, we considered $\alpha \in \Lambda$, $t_0 = -\theta / \ln(\nu)$ with $\theta \in \Theta$ and $\nu \in \Theta$, $t_f \in \{1, 10^{-1}, 10^{-2}, \dots, 10^{-5}\}$, $\delta \in \{0.80, 0.85, 0.90, 0.95, 0.99, 0.999\}$, and $j_{\max} \in \{1, 10, 20, \dots, 100\}$. We ran irace with maxExperiments $= 2{,}000$ and all its other default parameters. We used 30 instances for training and 30 instances for testing (10 for each learning factor value $\alpha \in \{0.1, 0.2, 0.3\}$). These instances were generated with the generator introduced in Birgin et al. (2014), with the same parameters that were used in Birgin et al. (2014) to generate the large-sized instances. Energy consumption data was also added as described at the beginning of Section 7. In this parameter calibration phase, we used a CPU time limit of 5 minutes as the stopping criterion for the five methods. As all the considered methods have random components, we ran each instance 10 times. As a result, we selected $\alpha = 0.2$ for the two versions of the GRASP, $\alpha = 0.1$ and $\epsilon = 0.60$ for GVNS, $\theta = 0.25$, $\nu = 0.30$, $t_f = 10^{-3}$, $\delta = 0.90$ and $j_{\max} = 50$ for SA-SRRN, and $\alpha = 0.50$, $\theta = 0.10$, $\nu = 0.40$, $t_f = 10^{-2}$, $\delta = 0.95$ and $j_{\max} = 50$ for SA-SRDRR.

Details of the results obtained by applying each method to each instance can be found in (Birgin et al., 2024, Tables S3, S4, and S5). Table 4 shows a summary of the results. In the table, we show the average energy consumption when considered the average of the 10 runs per method/instance, the average energy consumption when considering the lowest value of the 10 runs per method/instance, the average CPU time (in seconds) considering for each pair method/instance the time of the run that found the lowest energy consumption. The number of best solutions found and the average gap to the solution found by the GCH constructive heuristic are also displayed. Out of the total of 150 instances, each of the methods GRASP-LS-SRRN, GRASP-LS-SRDRR, GVNS, SA-LS-SRRN, and SA-LS-SRDRR found the best solution in 38, 52, 64, 0 and 2 instances each. Each of the methods improved the initial solution given by the GCH constructive heuristic by 12.43%, 11.51%, 11.92%, 2.97% and 10.59%, respectively. Figure 4 shows the evolution of the energy consumption of the solutions constructed by each of the methods as a function of time. To strengthen the comparison between methods, we used the Wilcoxon test (Wilcoxon, 1945) for each pair of methods, with a significance level of $\bar{\alpha} = 0.05$, to accept or reject the null hypothesis that "the samples of the two methods come from the same distribution" or, equivalently, "the difference between the samples of the two methods follows a symmetric distribution around zero". Table 5 shows the results. This shows that GVNS and GRASP-LS-SRRN are equivalent. Furthermore, both are better than all other methods that are different from each other. GRASP-LS-SRDRR is the third best method, followed by SA-LS-SRDRR and finally SA-LS-SRRN.

Figure 4 shows that the comparison between the methods in the previous paragraph is valid when considering a CPU time limit of 5 minutes. For lower CPU time limits, the ranking between the methods, in terms of average power consumption, may vary. When we compare the two versions of GRASP using local searches with neighborhoods SRRN and SRDRR we observe that: (i) for small time budgets, they behave similarly, (ii) for intermediate values of time budget, the intensity of neighborhood SRDRR leads to better solutions on average, and (iii) for larger time budgets, neighborhood SRRN, which is cheaper, allows the method to make a higher diversification by performing more local searches of different initial solutions and that leads, in the end, to better solutions, on average. On the other hand, GVNS seems to make better use of the combination of the two existing neighborhoods and outperforms, when evaluating the average energy consumption, the two versions of GRASP. The SA, with either of the two

Instances		GRASP-LS-SRRN					GRASP-LS-SRDRR				
type	α	#wins	$\overline{gap}(\%)$	$\overline{E}$	$\overline{E_{\min}}$	$\overline{Time}$	#wins	$\overline{gap}(\%)$	$\overline{E}$	$\overline{E_{\min}}$	$\overline{Time}$
DA	0.1	7	-12.71	1,089,544.74	1,077,181.30	275.11	6	-11.84	1,095,389.50	1,087,000.47	205.18
	0.2	7	-11.40	935,288.63	924,574.67	276.96	6	-10.31	945,413.83	936,549.37	202.52
	0.3	8	-11.89	809,016.94	800,324.43	305.50	5	-10.89	815,644.53	809,252.77	163.57
Y	0.1	4	-13.34	1,445,083.76	1,427,460.25	310.04	11	-12.61	1,457,253.89	1,450,942.15	200.66
	0.2	4	-13.01	1,283,622.88	1,269,477.20	276.27	12	-12.15	1,297,485.71	1,291,717.60	212.16
	0.3	8	-12.85	1,143,387.66	1,131,684.50	265.87	12	-11.98	1,155,404.40	1,151,201.50	209.29
All		38	-12.43				52	-11.51			

Instances		GVNS				
type	α	#wins	$\overline{gap}(\%)$	$\overline{E}$	$\overline{E_{\min}}$	$\overline{Time}$
DA	0.1	18	-13.13	1,080,904.47	1,062,464.40	24.58
	0.2	17	-11.43	924,694.28	917,809.47	26.61
	0.3	16	-11.81	802,414.55	796,975.97	26.12
Y	0.1	6	-12.23	1,447,009.73	1,434,069.70	46.56
	0.2	4	-11.38	1,289,193.43	1,281,924.55	42.25
	0.3	3	-11.20	1,147,983.68	1,143,420.50	41.35
All		64	-11.92			

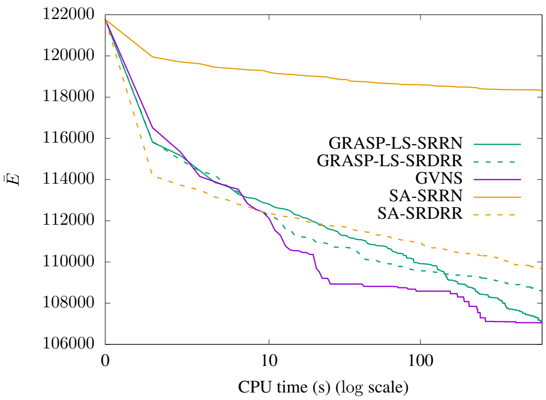
Instances		SA-LS-SRRN					SA-LS-SRDRR				
type	α	#wins	$\overline{gap}(\%)$	$\overline{E}$	$\overline{E_{\min}}$	$\overline{Time}$	#wins	$\overline{gap}(\%)$	$\overline{E}$	$\overline{E_{\min}}$	$\overline{Time}$
DA	0.1	0	-4.25	1,233,066.26	1,184,646.63	22.90	0	-11.61	1,107,158.51	1,089,523.03	190.43
	0.2	0	-2.96	1,037,538.97	1,011,472.83	16.49	1	-10.04	953,753.95	938,522.93	188.95
	0.3	0	-3.75	900,615.48	878,058.83	19.98	1	-10.53	826,519.04	812,194.93	163.53
Y	0.1	0	-1.63	1,610,423.48	1,599,004.00	2.19	0	-10.88	1,501,692.65	1,477,110.30	133.30
	0.2	0	-2.38	1,425,396.49	1,406,635.35	4.79	0	-10.45	1,338,861.23	1,312,209.35	107.78
	0.3	0	-1.84	1,274,202.92	1,258,199.90	4.74	0	-9.83	1,197,182.75	1,176,760.10	128.27
All		0	-2.97				2	-10.59			

Table 4
Summary of the results obtained by applying the metaheuristics GRASP-LS-SRRN, GRASP-LS-SRDRR, GVNS, SA-LS-SRRN, and SA-LS-SRDRR to the 50 large-sized instances based on the instances introduced in Birgin et al. (2014), with learning rates $\alpha \in \{0.1, 0.2, 0.3\}$.

neighborhoods, which does not use a local search strategy, does not present a competitive performance when compared to the other methods.

comparison			R^+	R^-	p-value
GRASP-LS-SRRN	versus	GRASP-LS-SRDRR	7,588	3,734	0.0003
GRASP-LS-SRRN	versus	GVNS	5,361	5,964	0.5716
GRASP-LS-SRRN	versus	SA-LS-SRRN	11,325	0	0.0000
GRASP-LS-SRRN	versus	SA-LS-SRDRR	11,275	50	0.0000
GRASP-LS-SRDRR	versus	GVNS	4,329	6,996	0.0124
GRASP-LS-SRDRR	versus	SA-LS-SRRN	11,325	0	0.0000
GRASP-LS-SRDRR	versus	SA-LS-SRDRR	10,540	785	0.0000
GVNS	versus	SA-LS-SRRN	11,325	0	0.0000
GVNS	versus	SA-LS-SRDRR	9,667	1,658	0.0000
SA-LS-SRRN	versus	SA-LS-SRDRR	0	11,324	0.0000

Table 5
Details of the Wilcoxon test comparing each pair of methods, when applied to the large-sized instances, to accept or reject the null hypothesis “the difference between the two methods follows a symmetrical distribution around zero”.



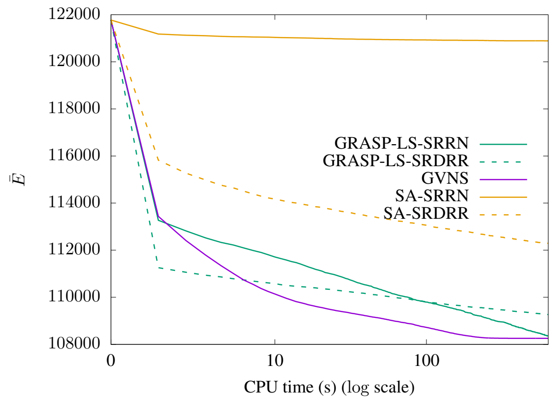


Fig. 4. This figure shows the average energy of the solutions found by each method as a function of CPU time. The average is calculated over the total of 150 large-sized instances. In the graphic on the left, the average considers, for each instance, the minimum over the 10 runs. In the graphic at the right, the average considers, for each instance, the average of the 10 runs.

7.3. Comparison with solutions from an exact solution method

In this section, we consider the two best performing metaheuristics (GVNS and GRASP-LS-SRRN) and analyze their performance considering solutions computed with an exact method. For these experiments, we considered the 50 large-sized instances from the previous section, plus 60 small-sized instances introduced in Araujo et al. (2024b), for which we included the energy consumption data in exactly the same way as for the large-sized instances (see the description at the beginning of Section 7). For details on the characteristics of small-sized instances, see (Birgin et al., 2024, Table S12). Since in this experiment we will also consider learning rates $\alpha \in \{0.1, 0.2, 0.3\}$, we will have a total of 150 large-sized instances plus 180 small-sized instances.

Models were solved using IBM ILOG CPLEX Optimization Studio version 22.1, using default parameters, with concert library and C++. The code was compiled using g++ 10.2.1. We provided as initial solution the solution calculated with the GCH constructive heuristic (Algorithm 1). A solution is reported as optimal by CPLEX when

$$\text{absolute gap} = \text{best feasible solution} - \text{best lower bound} \leq \epsilon_{\text{abs}} \tag{23}$$

or

$$\text{relative gap} = \frac{|\text{best feasible solution} - \text{best lower bound}|}{10^{-10} + |\text{best feasible solution}|} \leq \epsilon_{\text{rel}}, \tag{24}$$

where, by default, $\epsilon_{\text{abs}} = 10^{-6}$ and $\epsilon_{\text{rel}} = 10^{-4}$, and “best feasible solution” means the smallest value of the objective function related to a feasible solution generated by the method. Since the optimal value of the objective function of the instances considered in this paper is always an integer, we chose $\epsilon_{\text{abs}} = 1 - 10^{-6}$ and $\epsilon_{\text{rel}} = 0$. Choosing $\epsilon_{\text{rel}} = 0$ avoids premature stops in a solution that may not be optimal. The choice $\epsilon_{\text{abs}} = 1 - 10^{-6}$ allows stopping early when a relative gap less than 1 clearly indicates that the optimal solution has already been found. A CPU time limit of 1 hour was set. All other CPLEX parameters were used with their default values. Details of the solutions found by the exact method are available in (Birgin et al., 2024, Tables S6, S7, S8, S13, S14, and S15). Out of the 150 large-sized instances, CPLEX was able to find a single provably optimal solution. As for the small-sized instances, despite their relatively small size, CPLEX was able to find a provably optimal solution in only 137 out of 180 instances.

Details of the solutions found by the GVNS and GRASP-LS-SRRN metaheuristics when applied to the small-sized instances are available in (Birgin et al., 2024, Tables S16, S17, and S18). The heuristics were used with the parameters calibrated for the large-sized instances. That is, they were not recalibrated. Since these are random component methods, each method was applied 10 times to each instance. When comparing the solutions found by the metaheuristics with the solutions found by CPLEX, we consider (a) the mean of the 10 runs and (b) the minimum of the 10 runs. In each case, we calculate the relative gap with respect to the solution found by CPLEX. Let us first consider case (b).

If we consider just the 137 small-sized instances in which CPLEX found a provably optimal solution, GVNS finds solutions that are, on average, 2.02% away from the optimal solution, while this number is 0.22% for GRASP-LS-SRRN. If we consider all 180 small-sized instances, these values are 2.16% and 0.02%, respectively. In the only large-sized instance in which CPLEX found a proven optimal solution, the GVNS and GRASP-LS-SRRN metaheuristics found a solution with a gap of 3.98% and 0.09%, respectively. Considering all 150 large-sized instances, the GVNS and GRASP-LS-SRRN metaheuristics found solutions with average gaps of -8.70% and -9.32%, respectively. When we consider case (a), i.e., the average of the 10 runs for each method/instance, these same four values for the small-sized instances are 2.16%, 0.24%, 2.29% and 0.04%, respectively, while they are 4.50%, 0.09%, -8.00%, and -8.42% for the large-sized instances, i.e., little significant variation. The most relevant data from these experiments is that GRASP-LS-SRRN finds solutions at, on average, 0.22% of the 137 known optima and, when we include the 43 instances with non-guaranteed known optima, the average gap is 0.02%.

8. Concluding remarks

In this work we considered the flexible jobshop environment with two special features: nonlinear routes (or precedences between operations of the same job given by an arbitrary directed acyclic graph) and learning effect on the processing time. In alignment with contemporary sustainability concerns, we considered the minimization of energy consumption. We formulated the problem as a mixed-integer linear programming problem. We proposed a constructive heuristic, two neighborhoods, and three metaheuristics. We conducted comprehensive experiments to demonstrate the efficacy of the studied methods. The GVNS that uses the two neighborhoods concomitantly was the most effective in the large-sized instances. In the small-sized instances, the GRASP with the removing-and-reinserting neighborhood was the most effective. It found solutions that are, on average, 0.22% of the known optimal solutions.

As future work, we intend to consider the FJS environments with nonlinear routes and, instead of energy consumption, the total energy cost (TEC). This means considering energy costs that vary over time, including the consideration of peak times and seasonal tariffs or electricity tariffs by time of use (TOU). See Shen et al. (2023) and references therein for details. Another possibility that brings the problem under consideration closer to reality is to consider that a machine can operate at different speeds and that its energy consumption depends on its speed. See Wu and Sun (2018). Alternative learning models to the one considered in the present work, as well as deterioration models, are reviewed in Pei et al. (2022). Analyzing the different learning models as well as including the influence of deterioration in the context of the studied problem are possible tasks for future work.

Acknowledgments

This work was founded by the Brazilian agencies FAPESP (grants 2013/07375-0, 2022/05803-3, 2022/16743-1, and 2023/08706-1) and CNPq (grants 311536/2020-4 and 302073/2022-1).

References

Araujo, K.A.G., Birgin, E.G., Ronconi, D.P., 2024a. Local search and trajectory metaheuristics for the flexible job shop scheduling problem with nonlinear routes and position-based learning effect. Technical Report MCDO19032024, University of São Paulo, São Paulo, SP, Brazil.

Araujo, K.A.G., Birgin, E.G., Ronconi, D.P., 2024b. Models, constructive heuristics, and benchmark instances for the flexible job shop scheduling problem with nonlinear routes and position-based learning effect. Technical Report MCDO02022024, University of São Paulo, São Paulo, SP, Brazil.

Assembly, U.N.G., 2015. Transforming our world: The 2030 Agenda for Sustainable Development, A/RES/70/1, https://wedocs.unep.org/20.500.11822/11125.

Birgin, E.G., Feofiloff, P., Fernandes, C.G., de Melo, E.L., Oshiro, M.T.I., Ronconi, D.P., 2014. A MILP model for an extended version of the flexible job shop problem. *Optimization Letters* 8, 4, 1417–1431.

Birgin, E.G., Ferreira, J.E., Ronconi, D.P., 2015. List scheduling and beam search methods for the flexible job shop scheduling problem with sequencing flexibility. *European Journal of Operational Research* 247, 2, 421–440.

Birgin, E.G., Riveaux, J.A., Ronconi, D.P., 2024. Supplementary material to the article "Energy-aware flexible job shop scheduling problem with nonlinear routes and position-based learning effect". Technical Report MCDO27062024, University of São Paulo, São Paulo, SP, Brazil.

Biskup, D., 1999. Single-machine scheduling with learning considerations. *European Journal of Operational Research* 115, 1, 173–178.

Cheng, T.C.E., Wang, G., 2000. Single machine scheduling with learning effect considerations. *Annals of Operations Research* 98, 1/4, 273–290.

Dauzère-Pérès, S., Ding, J., Shen, L., Tamssaouet, K., 2024. The flexible job shop scheduling problem: A review. *European Journal of Operational Research* 314, 2, 409–432.

De Jong, J.R., 1957. The effects of increasing skill on cycle time and its consequences for time standars. *Ergonomics* 1, 1, 51–60.

Destouet, C., Tlahig, H., Bettayeb, B., Mazari, B., 2023. Flexible job shop scheduling problem under industry 5.0: A survey on human reintegration, environmental consideration and resilience improvement. *Journal of Manufacturing Systems* 67, 155–173.

Destouet, C., Tlahig, H., Bettayeb, B., Mazari, B., 2024. Multi-objective sustainable flexible job shop scheduling problem: Balancing economic, ecological, and social criteria. *Computers & Industrial Engineering* 195, 110419.

Feo, T.A., Resende, M.G.C., 1995. Greedy randomized adaptive search procedures. *Journal of Global Optimization* 6, 2, 109–133.

Gahm, C., Denz, F., Dirr, M., Tuma, A., 2016. Energy-efficient scheduling in manufacturing companies: A review and research framework. *European Journal of Operational Research* 248, 3, 744–757.

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

Gong, G., Tang, J., Huang, D., Luo, Q., Zhu, K., Peng, N., 2024. Energy-efficient flexible job shop scheduling problem considering discrete operation sequence flexibility. *Swarm and Evolutionary Computation* 84, 101421.

Gong, X., De Pessemier, T., Martens, L., Joseph, W., 2019. Energy- and labor-aware flexible job shop scheduling under dynamic electricity pricing: A many-objective optimization investigation. *Journal of Cleaner Production* 209, 1078–1094.

Gupta, J.N.D., Gupta, S.K., 1988. Single facility scheduling with nonlinear processing times. *Computers and Industrial Engineering* 14, 4, 387–393.

Ham, A., Park, M.J., Kim, K.M., 2021. Energy-aware flexible job shop scheduling using mixed integer programming and constraint programming. *Mathematical Problems in Engineering* 2021, 1–12.

Hansen, P., Mladenović, N., Brimberg, J., Pérez, J.A.M., 2018. Variable Neighborhood Search, Springer International Publishing. pp. 57–97.

Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C., 1989. Optimization by simulated annealing: An experimental

evaluation – Part I, graph partitioning. *Operations Research* 37, 6, 865–892.
Kahn, A.B., 1962. Topological sorting of large networks. *Communications of the ACM* 5, 11, 558–562.
Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., 1983. Optimization by simulated annealing. *Science* 220, 4598, 671–680.
Laurent, A., Olsen, S.I., Hauschild, M.Z., 2010. Carbon footprint as environmental performance indicator for the manufacturing industry. *CIRP Annals* 59, 1, 37–40.
Lei, D., Zheng, Y., Guo, X., 2016. A shuffled frog-leaping algorithm for flexible job shop scheduling with the consideration of energy consumption. *International Journal of Production Research* 55, 11, 3126–3140.
Li, H., Zhu, H., Jiang, T., 2020. Modified migrating birds optimization for energy-aware flexible job shop scheduling problem. *Algorithms* 13, 2, 44.
Li, Z., Chen, Y., 2023. Minimizing the makespan and carbon emissions in the green flexible job shop scheduling problem with learning effects. *Scientific Reports* 13, 6369.
Lu, Y., Lu, J., Jiang, T., 2019. Energy-conscious scheduling problem in a flexible job shop using a discrete water wave optimization algorithm. *IEEE Access* 7, 101561–101574.
Lunardi, W.T., Birgin, E.G., Laborie, P., Ronconi, D.P., Voos, H., 2020. Mixed integer linear programming and constraint programming models for the online printing shop scheduling problem. *Computers and Operations Research* 123, 105020, 105020.
Lunardi, W.T., Birgin, E.G., Ronconi, D.P., Voos, H., 2021. Metaheuristics for the online printing shop scheduling problem. *European Journal of Operational Research* 293, 2, 419–441.
López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Birattari, M., Stützle, T., 2016. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives* 3, 43–58.
Meng, L., Zhang, C., Shao, X., Ren, Y., 2019. MILP models for energy-aware flexible job shop scheduling problem. *Journal of Cleaner Production* 210, 710–723.
Mouzon, G., Yildirim, M.B., Twomey, J., 2007. Operational methods for minimization of energy consumption of manufacturing equipment. *International Journal of Production Research* 45, 18-19, 4247–4271.
Pei, J., Zhou, Y., Yan, P., Pardalos, P.M., 2022. A concise guide to scheduling with learning and deteriorating effects. *International Journal of Production Research* 61, 6, 2010–2031.
Ren, W., Wen, J., Yan, Y. Y.and Hu, Guan, Y., Li, J., 2020. Multi-objective optimisation for energy-aware flexible job-shop scheduling problem with assembly operations. *International Journal of Production Research* 59, 23, 7216–7231.
Shen, L., Dauzère-Pérès, S., Maecker, S., 2023. Energy cost efficient scheduling in flexible job-shop manufacturing systems. *European Journal of Operational Research* 310, 3, 992–1016.
Wagner, H.M., 1959. An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly* 6, 2, 131–140.
Wilcoxon, F., 1945. Individual comparisons by ranking methods. *Biometrics Bulletin* 1, 6, 80–83.
Wilson, J.M., 1989. Alternative formulations of a flow-shop scheduling problem. *Journal of the Operational Research Society* 40, 4, 395–399.
Wu, X., Shen, X., Li, C., 2019. The flexible job-shop scheduling problem considering deterioration effect and energy consumption simultaneously. *Computers and Industrial Engineering* 135, 1004–1024.
Wu, X., Sun, Y., 2018. A green scheduling algorithm for flexible job shop with energy-saving measures. *Journal of Cleaner Production* 172, 3249–3264.
Xie, J., Gao, L., Peng, K., Li, X., Li, H., 2019. Review on flexible job shop scheduling. *IET Collaborative Intelligent Manufacturing* 1, 3, 67–77.
Zhang, H., Deng, Z., Fu, Y., Lv, L., Yan, C., 2017a. A process parameters optimization method of multi-pass dry milling for high efficiency, low energy and low carbon emissions. *Journal of Cleaner Production* 148, 174–184.
Zhang, L., Tang, Q., Wu, Z., Wang, F., 2017b. Mathematical modeling and evolutionary generation of rule sets for energy-efficient flexible job shops. *Energy* 138, 210–227.
Zhang, Y., Liu, Q., Zhou, Y., Ying, B., 2017c. Integrated optimization of cutting parameters and scheduling for reducing carbon emissions. *Journal of Cleaner Production* 149, 886–895.
Zhu, H., Deng, Q., Zhang, L., Hu, X., Lin, W., 2020. Low carbon flexible job shop scheduling problem considering worker learning using a memetic algorithm. *Optimization and Engineering* 21, 4, 1691–1716.