Reducing waste in manufacturing operations: bi-objective scheduling on a single-machine with coupled-tasks

Corentin Le Hesran, Aayush Agarwal, Anne-Laure Ladier, Valerie Botta-Genoulaz, Valérie Laforest

Reducing waste in manufacturing operations: bi-objective scheduling on a single-machine with coupled-tasks

Corentin Le Hesran[a], Aayush Agarwal[a], Anne-Laure Ladier[a], Valérie Botta-Genoulaz[a] and Valérie Laforest[b]

[a]Univ Lyon, INSA Lyon, DISP laboratory EA 4570, 69621 Villeurbanne cedex, France
[b]Univ Lyon, EMSE, CNRS, UMR 5600 EVS, 158 cours Fauriel, 42023 Saint-Etienne cedex 2, France

ARTICLE HISTORY
Compiled September 20, 2019

ABSTRACT

This study addresses a scheduling problem involving a single-machine with coupled-tasks and bi-objective optimisation considering simultaneously inventory and environmental waste. A Mixed Integer Linear Program (MILP) representing the problem is first developed. Subsequently, a Genetic Algorithm (GA) is presented, followed by numerical experiments on multiple instances. Pareto fronts are determined using the ε-constraint and weighted sum methods, and a trade-off point is selected according to a distance criterion. Numerical experiments on both small and large instances show near optimal results for small instances, and considerably reduced computing times for large ones when using the GA. The results show that a compromise can be found, with a decrease in setup-related waste up to 36% for an increase of inventory of 12%. This will help decision-makers to better consider the environmental aspect when designing schedules, as well as reduce their production environmental impact and waste management costs.

KEYWORDS
Coupled-tasks, bi-objective optimisation, scheduling, genetic algorithm, waste prevention, Mixed-integer linear programming
Word Count : 7191.

1. Introduction

Sustainable production is defined as "the creation of goods and services using processes and systems that are non-polluting; conserving of energy and natural resources; economically viable; safe and healthful for workers, communities, and consumers; and socially and creatively rewarding for all working people" (Lowell Center for Sustainable Production, 1998). In the recent years, more and more research has been devoted to it as a possible answer to the environmental issues affecting industrial companies, such as stricter regulations, highly volatile energy prices, shortage of raw materials and natural resources or customer demand for more environmentally responsible products (Giret et al., 2015). As a key factor in production efficiency, operations scheduling is one of several levers that can be used in order to address those problems. In a literature review of sustainability in manufacturing operations scheduling, Giret et al. (2015) show that concerns have been mostly focused on the reduction of energy consumption thus far; detailed reviews on energy efficient scheduling can be found in Gahm et al. (2016) and Biel and Glock (2016). Giret et al. (2015) also emphasise the need to address the consequences of scheduling implementation, notably waste, to design sustainable scheduling systems, since there are few works on this topic.

This project is supported by the Auvergne Rhône-Alpes region.
Corresponding author: C. Le Hesran, email: corentin.le-hesran@insa-lyon.fr

Our goal is to show how scheduling can help reduce waste generation, e.g. by reducing cleaning and setup operations. As an example, in a painting line if there are less painting cabins than types of paint, the painting nozzles need to be cleaned at every paint change. Cleaning the painting nozzles generates wastewater soiled with paint particles and solvent, which needs to be treated and then disposed of. This treatment is costly and its end product, paint sludge, is considered hazardous by the European Waste Code (European Commission, 2000). This entails production restriction, careful handling and storage, and the hiring of a special contractor to retrieve it. Thus, it might be beneficial both environmentally and economically to include these costs in production scheduling considerations and minimise the number of paint changes. This particular problem is a case of coupled-tasks scheduling problem, where a product needs to be processed multiple times by the same machine with mandatory lag-times between each operation (Shapiro, 1980). It has been solved using Mixed Integer Linear Programming (MILP) in Le Hesran et al. (2018), showing potential for compromises between inventory and paint changes but requiring large computation times. The present paper is an extended version of Le Hesran et al. (2018) proposing a solving method based on a Genetic Algorithm (GA) in order to reduce the computation time needed for large instances. The contribution of this paper is twofold. Firstly, it introduces an environmental criterion where none had been considered before, allowing for the implementation of greener schedules. Secondly, it provides both an exact and meta-heuristic methods for solving a bi-objective problem involving coupled-tasks, which had not previously been done, providing relevant information for decision-making.

The rest of the paper is organised as follows: a review of the relevant literature is provided in section 2. Section 3 introduces the positioning of our work, the mathematical model of the problem and the data, variables and constraints used. Section 4 explains the structure and different operators of the GA. Finally, section 5 presents the numerical experiments and results, followed by conclusions regarding both solving methods (GA and MILP) in section 6.

2. Literature review

In this section, a brief review of the existing literature relevant to our case is conducted. It includes some cases of waste-related sustainable production, a summary of research concerning the coupled-tasks scheduling problem and some studies making use of GAs in operations scheduling.

2.1. *Waste-conscious scheduling*

As previously stated, Giret et al. (2015) note that research on waste-reducing scheduling is scarce, and mainly originates from the chemical industry. In a recent literature review on waste-minimising scheduling problems, Le Hesran et al. (2018) identify 70 articles that address the issue of waste minimisation through scheduling. Among those, Adonyi et al. (2008) address the issue of waste generated during equipment cleaning in a paint producing plant. By adapting former research on S-Graphs by Sanmartí et al. (2002), they propose a new algorithm which accounts for the equipment cleaning cost, and offer several alternative solutions with varying makespans and total cleaning costs. Similarly, Zhang (2018) addresses the issue of minimising setup-related waste in painting lines by avoiding colour changes, respectively using MILP and Particle Swarm Optimisation, providing alternative schedules. Other examples of setup-induced waste minimising scheduling problems can be found in Nonas and Thorstenson (2008); Cui and Yang (2010); Wu et al. (2017) or Pulluru et al. (2017), none of them including coupled-tasks characteristics.

2.2. *Coupled-tasks scheduling problems*

A coupled-tasks scheduling problem is a problem where a set of n jobs comprising two operations has to be processed on the same single-machine (Shapiro, 1980), a job being a set of similar products with a defined size and due date. An exact amount of time, or time-lag L_i, needs to elapse between the end of the first operation of job i (processing time a_i) and the beginning of its second operation (processing time b_i). Operations from other jobs can be processed during this time-lag. It is noted as $\{1 \mid \text{Coup-task} \mid -\}$ in the extended Graham notation (Graham et al., 1979). Particular cases of interest can emerge when specifying the values of a_i, b_i or L_i. Blazewicz et al. (2012) provide a survey of research on coupled-tasks scheduling problems, as well as a list of important results for the most common variants and subproblems.

The complexity of the coupled-task scheduling problem was studied by Orman and Potts (1997). They prove the general problem to be NP-Hard, as well as several particular cases. Due to the complexity of the problem, heuristic-based methods are more frequent than exact methods. They usually

focus on the minimisation of makespan, such as Shapiro (1980) who develops three sub-optimal algorithms for a specific case of radar scheduling, while Gupta (1996) develops several heuristic algorithms. Lin and Haley (1993) solve the makespan minimisation problem with arbitrary lower-bound time delays using greedy and iterative heuristics as well as a branch and bound algorithm. Lin et al. (1995) consider the same problem, using threshold acceptance and simulated annealing. Ahr et al. (2004) study the Identical Coupled-Tasks Problem where all processing times and time-lags are equal between jobs (i.e. $a_i = a, b_i = b, L_i = L$), and define an exact algorithm to solve it. Their work is adapted by Brauner et al. (2009) to fit a one-machine robotic cell problem, both with an exact and bounded delay L. Potts and Whitehead (2007) study the makespan minimisation problem with upper and lower bounds for the time-lags and compare seven different heuristics. Condotta and Shakhlevich (2012) propose a tabu-search algorithm for the exact time-lag problem, and compare it with the join-and-decompose heuristic defined by Potts and Whitehead (2007) for the flexible time-lag problem. A tabu-search metaheuristic for solving the general case is also developed by Li and Zhao (2007), showing good results when compared with a theoretical lower bound, as well as some algorithms for NP-Hard special cases of the problem. Finally, Amrouche and Boudhar (2016) and Amrouche et al. (2017) consider the problem of the two-machine chain re-entrant with identical time lags. This problem, noted $\{F2|ChR, l_j = L|C_{max}\}$ considers a two machine flowshop with exact time-lags where each task needs to be processed twice on the first machine. They develop nine heuristics, with numerical experiments marking two of those, $IH_{L_{RP}}$ and IH_{L6}, as more efficient. Courtad et al. (2017) study the single machine flowtime minimisation problem with paired-tasks, in which a minimum delay must occur between two tasks of a same job. They first use a MILP approach, then propose an insertion heuristic providing near-optimal results. Finally, Meziani et al. (2018) propose to minimise the makespan in a two-machine flowshop with coupled tasks problem ($F2|a_j, b_j, L_j, c_j|C_{\max}$). They first propose a lower bound as well as four heuristics. A hybrid PSO and Simulated Annealing (SA) metaheuristic is developed and compared with the PSO and SA-only approaches, outperforming them both. The coupled-task problem is a particular case of re-entrant problems, in which jobs are allowed to be processed by the same machine more than once. Re-entrant problems are mainly solved using heuristic approaches. Exact models are proposed by Chen and Chao-Hsien Pan (2006), who develop eight integer programming models for the re-entrant job-shop and flow-shop scheduling problem based on formulations by Wagner (1959), Manne (1960), Wilson (1989) and You and Chii-Tsuen (1992). Those initial models are not re-entrant, therefore Chen and Chao-Hsien Pan (2006) relax their assumption that every machine may only be visited once, in order to obtain new formulations for the re-entrant shop problem. Since re-entrant problems state that no two consecutive operations of a job can be processed on the same machine (Chen and Chao-Hsien Pan, 2006), this assumption must be relaxed for the coupled-task case.

Other works concern different objective functions or extensions of the problem. Focusing on radar control, Winter and Baptiste (2007) develop two heuristics and a local-search algorithm for a problem with lower and upper bounded time-lags, the objective function being the total cost minimisation of the delay between an operation's ideal starting time versus its real starting time. Simonin et al. (2011) study the acquisition and treatment of data by torpedoes, and propose an algorithm for minimising the makespan in a coupled-tasks problem with precedence constraints on treatment tasks $(1|\text{prec}, (a_i, L_i, b_i)(T_i, pmtn), G_c|C_{\max})$. Sequence dependence in the coupled-tasks scheduling problem is introduced by Blazewicz (2010) who studies the cases of general and in-out precedence constraints tree.

To the best of our knowledge, none of the papers on coupled-tasks scheduling address the issue of waste generation, as most consider the issue of makespan minimisation, nor do they optimise the number of setups required.

Additionally, no multi-objective problems involving coupled tasks have been addressed. Although the issue of reentrance was tackled using for example genetic algorithms (Dugardin et al., 2010; Cho et al., 2011; Zhang et al., 2012) or large neighborhood search (Rifai et al., 2016), none of them consider an environmental criterion in their objectives.

3. Problem modeling

In this section, the problem is defined and the particularities of our approach compared to the previous works reviewed in Section 2 are highlighted. The MILP is then detailed and the different constraints explained.

3.1. *Problem definition*

Drawing inspiration from the example of hubcap factory described in Section 1, our problem is defined as follows. In the shop-floor, only one painting line is available for the processing of all products, making this a single-machine scheduling problem. A passage into the painting line is referred to as an operation, while the set of operations required for completion of an order is called a job.

The products can be painted several times to apply coatings, which can be of different colours, meaning that a job can require to go through the painting line more than once. Additionally, a minimum drying time must be left between two coatings of a same product. Based on the definition provided in Section 2.2, this is a coupled-tasks scheduling problem. Since there is a need to clean the painting line at each colour change, another of its characteristics is a sequence-dependent waste treatment cost. A hard due date constraint forbids lateness on any order, and earliness is not desirable either since keeping inventory represents a holding cost for the company: indeed, most manufacturing companies nowadays operate on a just-in-time basis.

The objective function is therefore twofold. An environmental objective consists in minimising the waste induced by setups, which is proportional to the number of colour changes. The economic objective is to minimise the inventory associated with earliness, which is defined by the number of products held in the inventory while they await processing or shipping.

Based on this previous description, and using the extended three-fields notation from Blazewicz et al. (2012), this problem can be written as $\{1 \mid (a_i, L, b_i)$, dependent cost, $D_i \mid \sum E_i$, Number of setups$\}$

The proposed mathematical model is based on the extension of Manne's model (Manne, 1960) by Chen and Chao-Hsien Pan (2006) which assumes that the jobs to be scheduled are composed of different numbers of operations. The assumption that no machine can process two tasks of a same job consecutively was relaxed to allow for a single machine setting.

3.2. *Problem data*

The next paragraphs detail the different sets, data and decision variables necessary for the modeling of the problem as a MILP. While the studied case of hubcap manufacturing only considers two operations per job, this model also works for problems with more than two operations per job.

3.2.1. Data sets

- $\mathcal{I}$: set of the different jobs to be scheduled;
- $\mathcal{J}$: set of the different operations composing a job.

3.2.2. Data

- P_{ij}: processing time for operation j of job i;
- L: minimum drying time between two consecutive operations of a job;
- Q_i: number of products in job i;
- D_i: due date for job i;
- $Y_{ijkl} = 1$ if switching from operation j of job i to operation l of job k implies a setup, 0 otherwise;
- N_i: number of operations for job i;
- M: maximum length of the planning horizon, i.e. $M = \max_{i \in \mathcal{I}} D_i$.

3.2.3. Decision variables

- y_{ijkl}: 1 if operation j of job i takes place just before operation l of job k, 0 otherwise;
- s_{ij}: starting time of operation j of job i;
- t_{ij}: drying time after operation j of job i;
- e_i: earliness of job i (time between the end of the last operation and the due date of job i);
- g_{ij}: machine idle-time between the end of operation j of job i and the next scheduled operation.

The objective function is composed of two elements:

- $z_{\text{inventory}}$: the total inventory, which represents all products finished early, that therefore must be stored until their due date. This includes semi-finished products that stay in the drying inventory longer than the minimum required amount of time. Minimising the inventory equates

to minimising the holding cost, since it is proportional to the number of products in stock multiplied by the time spent before expedition;

- z_{setup}: the number of setups needed. Since a fixed quantity of waste is generated each time a colour change occurs, waste treatment and disposal cost is proportional to the number of setups.

3.3. *Mathematical model*

The complete MILP is detailed in Figure 1.

$$\min \quad z_{\text{inventory}} = \sum_{i \in \mathcal{I}} Q_i \times e_i + \sum_{i \in \mathcal{I} | N_i > 1} \sum_{j=1}^{N_i} Q_i (t_{ij} - L)$$

$$\min \quad z_{\text{setup}} = \sum_{i \in \mathcal{I}} \sum_{j=1}^{N_i} \sum_{k \in \mathcal{I}} \sum_{l=1}^{N_k} Y_{ijkl} \; y_{ijkl}$$

$$\text{s.t.} \quad s_{ij} + P_{ij} + t_{ij} \leq s_{i,j+1} \qquad \forall i \in \mathcal{I}, j \in \{1, \ldots, N_i - 1\} \qquad (1)$$

$$y_{ijil} = 0 \qquad \forall i \in \mathcal{I}, j \in \{1, \ldots, N_i\}, \forall l \in \{1, \ldots, N_i \mid l \leq j\} \qquad (2)$$

$$e_i = D_i - s_{iN_i} - P_{iN_i} \qquad \forall i \in \mathcal{I} \qquad (3)$$

$$s_{ij} + P_{ij} + g_{ij} + M(1 - y_{ijkl}) \geq s_{kl} \qquad \forall (i,k) \in \{\mathcal{I}^2 | k \neq i\}, j \in \{1, \ldots, N_i\}, l \in \{1, \ldots, N_k\} \qquad (4)$$

$$s_{ij} + P_{ij} + g_{ij} \leq M(1 - y_{ijkl}) + s_{kl} \qquad \forall (i,k) \in \{\mathcal{I}^2 | k \neq i\}, j \in \{1, \ldots, N_i\}, l \in \{1, \ldots, N_k\} \qquad (5)$$

$$\sum_{i \in \mathcal{I}} \sum_{j=1}^{N_i} \sum_{k \in \mathcal{I}} \sum_{l=1}^{N_k} y_{ijkl} = \sum_{i \in \mathcal{I}} N_i - 1 \qquad (6)$$

$$\sum_{k \in \mathcal{I}} \sum_{l=1}^{N_k} y_{ijkl} \leq 1 \qquad \forall i \in \mathcal{I}, j \in \{1, \ldots, N_i\} \qquad (7)$$

$$\sum_{k \in \mathcal{I}} \sum_{l=1}^{N_k} y_{klij} \leq 1 \qquad \forall i \in \mathcal{I}, j \in \{1, \ldots, N_i\} \qquad (8)$$

$$t_{ij} \geq L \qquad \forall i \in \{\mathcal{I} \mid N_i > 1\}, \forall j \in \mathcal{J} \qquad (9)$$

$$s_{ij}, t_{ij}, e_i, g_{ij} \geq 0 \qquad \forall i \in \mathcal{I}, j \in \mathcal{J} \qquad (10)$$

$$y_{ijkl} \in \{0, 1\} \qquad \forall i \in \mathcal{I}, j \in \mathcal{J}, k \in \mathcal{I}, l \in \mathcal{J} \qquad (11)$$

Figure 1. Mixed Integer Linear Program modeling the scheduling problem

Constraint set (1) ensures that all operations (but the first one) start only after the previous one on the same job is done and the drying time has ended. Constraints (2) ensure that no operation l of a job can be placed before operation j of a same job in the y_{ijkl} variables. Constraint set (3) defines job earliness as the difference between the due date and the completion date of the last operation on this job. The positivity constraint on e_i (10 ensures that no job can end after its due date.

Constraint sets (4) and (5) guarantee that only the operation (k, l) consecutive to (i, j) can be started after (i, j) (including some possible lag-time). They result from the linearisation of the following expression:

$$y_{ijkl} = 1 \Rightarrow s_{ij} + P_{ij} + g_{ij} = s_{kl}$$

Since y_{ijkl} is equal to one if and only if operation j of job i is directly followed by operation l of job k, each operation but the first one can have exactly one predecessor. Constraint (6) therefore makes sure that the number of possible successors is equal to the total number of operations minus one.

Constraint set (7) and (8) are used to make sure that a given operation (i, j) has no more than one successor or predecessor respectively. Constraint set (9) defines the minimum drying time between two operations of a same job. Finally, the non-negativity constraints and the binarity of y are given by constraint sets (10) and (11).

3.4. *Bi-objective approach*

Since we work on a multi-objective scheduling problem, it is advisable to provide the decision-maker with alternative solutions that represent the variety of possible results. In the case of bi-objective optimization, this can be achieved using a Pareto front. A Pareto front represents the set of non-dominated solutions for multiobjective optimization, i.e. solutions that cannot be improved without degrading at least one of the other objectives (more details are available in Blasco et al. (2008)). We generate this Pareto front using the ε-constraint method, that turns the multi-objective problem into a single-objective one by transforming other objective functions into constraints (Mavrotas, 2009). The fact that we only need to minimise two objectives, and that one takes integer values (namely the

number of setups z_{setup}) makes this method especially convenient. The steps required for the obtention of the Pareto front are detailed in Algorithm 1. The initialization phase computes a mono-objective MILP, using α and β as parameters in the weighted sum objective function. The chosen weights α=1 and β=0.005 ensure that the inventory criterion takes precedence over the number of setups, therefore giving the leftmost point of the Pareto front, i.e. the schedule with the lowest possible inventory and for which the number of setups cannot be reduced.

Algorithm 1 Pareto front generation

1: **Input:** Instance data
2: **Output:** Pareto front
3: Compute the $(z^{\text{min}}_{\text{inventory}}, z^{0}_{\text{setup}})$ point by solving the model with the following objective function: $z_{\text{weighted}} = \alpha z_{\text{inventory}} + \beta z_{\text{setup}}$
4: Set $\varepsilon = z^{0}_{\text{setup}} - 1$
5: **while** problem is feasible **do**
6: Solve the ε-constraint problem with $z_{\text{setup}} \leq \varepsilon$ as a constraint and $z_{\text{inventory}}$ as the objective function. Add the objective function value $(z^{\text{it}}_{\text{inventory}}, z^{\text{it}}_{\text{setup}})$ to the set of Pareto front points
7: Set $\varepsilon = z^{\text{it}}_{\text{setup}} - 1$
8: it = it + 1
9: **end while**

While obtaining this Pareto front provides us with all non-dominated schedules, all of them might not be suited to a practical use. Since the size of the front is limited by the maximum number of possible colour changes, which is equal to the total number of operations minus one, it might contain too many points to be easily understandable by the decision-makers. As such, providing the whole front might be counterproductive since it potentially contains a lot of unnecessary information. To alleviate this issue, key points are extracted for each instance, which each have different characteristics. Those points, and the way they are obtained, are detailed below:

Two extreme points $(z^{\text{min}}_{\text{inventory}}, z^{0}_{\text{setup}})$ and $(z^{0}_{\text{inventory}}, z^{\text{min}}_{\text{setup}})$, which represent the cases where the decision-maker wishes to minimise one objective in priority, either the inventory or the number of setups respectively.

The ideal point $(z^{\text{min}}_{\text{inventory}}, z^{\text{min}}_{\text{setup}})$, which is defined using the two optimum values of these points, i.e. the minimum inventory and minimum number of setups achievable.

The coordinates of each point z^{it} are normalised using the formula $z^{\text{normal}} = \frac{z^{\text{it}} - z^{\text{min}}}{z^{0} - z^{\text{min}}}$ for both $z_{\text{inventory}}$ and z_{setup}. This norm provides new values between 0 and 1 ; scaling both objective functions enables us to compare values of different nature and order of magnitude. This is especially useful in our case where a holding cost and a number of setups cannot be compared directly. In case the Pareto front consists of only one point, i.e. $z^{0} = z^{\text{min}}$ for inventory and setups, no normalization occurs and this single point is returned. Using these normalised values, the euclidean distance of each point to the ideal point is calculated, which is used to define our next key point:

The trade-off point $z^{\text{trade-off}}$, which represents the best compromise in terms of number of setups reduction versus increase in inventory according to our distance criterion. The euclidean distance provides an accurate evaluation of the geometrical distance to the ideal point, and corresponds more closely to the shape of the Pareto front.

Finally, one last key point is presented in order to provide another interesting alternative for decision-makers:

The percent point $z_{percent}$ is chosen as the point with the highest difference between waste percentage reduction and inventory percentage increase. This point aims at providing an attractive option for decision-makers that wish to improve their environmental impact without affecting their costs negatively.

An example of Pareto front with its important points is shown in Figure 2.

Figure 3 presents the Gantt chart for a 10 jobs instance with a drying-time L=4, couple (i, j) being operation j of job i, with due dates of each job appearing as red lines. Table 1 contains the processing times P_{ij}, operation type and due date D_i associated with each operation j of job i of the featured Gantt chart.

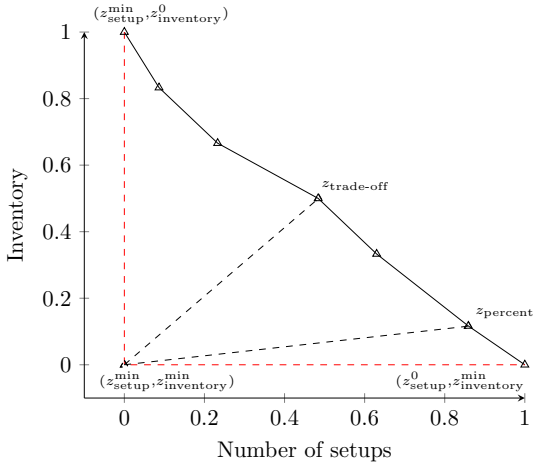

Figure 2. Example of Pareto front for a 10 jobs instances, 20%-80% distribution

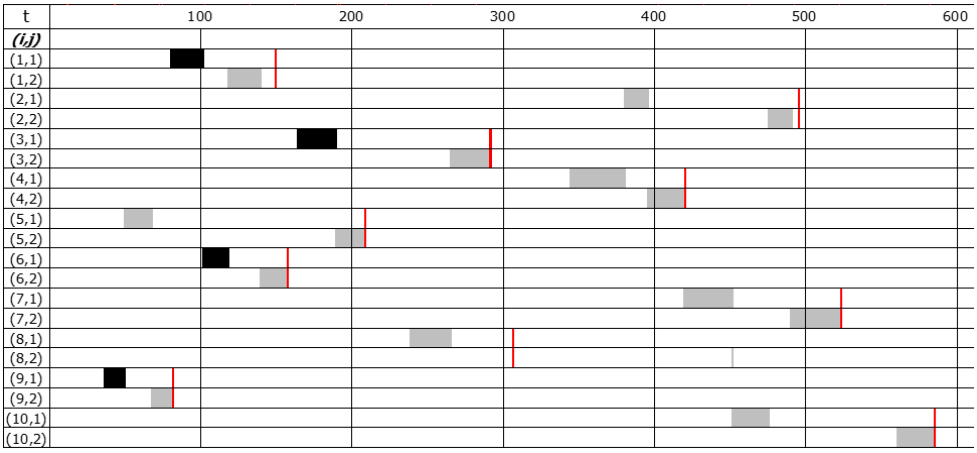

Figure 3. Gantt chart of a schedule with ten jobs

Table 1. Example chromosome instance data

i	1		2		3		4		5		6		7		8		9		10	
j	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
Type	2	1	1	1	1	2	1	2	2	1	1	2	1	1	2	0	1	1	2	1
P_{ij}	21	21	15	15	25	25	24	24	18	18	17	17	32	32	27	0	13	13	24	24
D_i	148	148	494	494	290	290	419	419	207	207	156	156	522	522	305	0	80	80	584	584

4. Genetic algorithm

4.1. *Genetic algorithms*

The principle of GAs was first introduced by J. Holland in the 1960s, and later formalised in Holland (1992). They are based on the theory of evolution, and the improvement of solutions through repeated modification and natural selection. Their basic idea is to maintain a population of candidate solutions that evolves under a selective pressure that favours better solutions. In the application of production scheduling, a GA is an iterative procedure that operates on a finite population of solutions called chromosomes. Each chromosome represents a fixed schedule of jobs and machine assignation, and can be evaluated according to a fitness function, similar to an objective function. The members of the population are then interbred using various genetic operators like crossover (to select useful traits)

and mutations (to introduce variety) to produce offspring. These offspring are evaluated based on the fitness function, and can replace the weaker chromosomes currently present in the population according to a defined reproduction strategy. This creates a new population of which new offspring can be created, and so on until a stopping mechanism is activated and the best solution is returned. GAs can be applied in many fields, including scheduling problems such as flowshops (Reeves, 1995), job-shops (Croce, 1995), flexible job-shops (Pezzella et al., 2008) and hybrid flowshops (Ruiz and Maroto, 2006). A good introduction to the use of GAs in scheduling can be found in Reeves (1996). Multiobjective optimisation is also an important feature of GAs (Konak et al., 2006; Deb et al., 2000), with studies including environmental concerns (Arbiza et al., 2008; Vaklieva-Bancheva and Kirilova, 2010; El Amraoui and Mesghouni, 2014; Araujo et al., 2014; Golfeto et al., 2009; Malik et al., 2009), although mostly energy-related (Giret et al., 2015; Dugardin et al., 2010; Liu et al., 2016; Zhang and Chiong, 2016). While GAs are metaheuristics, and offer no guarantee of optimality, they have the advantage of requiring less computation time than exact methods, which is particularly relevant when dealing with large instances.

Le Hesran et al. (2018) reported unpractical computation times for solving instances of our problem with more than ten jobs using MILP. To address this issue and enable the solving of industrial-size instances, the development of a meta-heuristic is necessary.

Some meta-heuristics such as PSO and SA (Meziani et al., 2018), tabu-search (Condotta and Shakhlevich, 2012; Li and Zhao, 2007), as well as various heuristics (Courtad et al., 2017; Amrouche et al., 2017) have been used to solve coupled-tasks scheduling problems. GAs also have been extensively used to solve scheduling problems, including problems involving reentrance characteristics which are similar to the coupled-tasks problem. Additionally, single-objective GAs can be efficiently adapted to bi-objective solving, either through the use of objective function aggregation or by implementing a Pareto dominance relationship in the fitness function. For these reasons, and due to their applicability to both scheduling and multiobjective optimisation as well as effectiveness for solving large instances, a GA was developed ; its features are detailed in the next section.

4.2. *Chromosome representation*

A sequence coding was adopted for chromosome representation. A chromosome represents a sequence of operations, its size being equal to the number of jobs times the maximum number of operations per job. Since not all jobs have the same number of operations, dummy operations with processing time zero are added to keep the chromosome size constant. This chromosome is constituted of genes, where a gene's position corresponds to the job it belongs to and its order within this job. The value of a gene represents its rank in the global operations sequence. Figure 3 features a Gantt chart of a problem and Table 1 its associated instance data. Table 2 is the corresponding chromosome, C giving the operations sequence, and the starting times s_{ij} in increasing order. As an example, operation 1 of job 9 is processed first, while operation 2 of job 1 is processed sixth, and operation 2 of job 8 is a dummy operation.

Table 2. Example chromosome sequence

i	1		2		3		4		5		6		7		8		9		10	
j	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2	1	2
C	4	6	13	17	8	10	12	14	2	9	5	7	15	18	11	20	1	3	16	19
s_{ij}	36	49	67	80	104	118	139	164	189	189	207	278	344	380	395	419	451	475	560	584

Once a sequence is known, the corresponding starting times are obtained using Algorithm 2. Here, L is the minimum drying time, s_k is the starting time of the k^{th} operation in the sequence, D_k its due date, and P_k its processing time. s_{last} refers to the starting time of the last operation to be scheduled, while $s_{\text{next}(k)}$ is the starting time of the operation scheduled right after operation k.

The fitness of a chromosome corresponds to the objective value ($\alpha \times z_{\text{inventory}} + \beta \times z_{\text{setup}}$) of its associated schedule, making it a single-objective GA. It can be shown (see Appendix A for proof) that this algorithm returns minimal inventory for a given operations order.

4.3. *Initialisation*

The initialisation step refers to the creation of the initial population. While randomisation is a common method for generating initial solutions, it is usually used for binary encoding and when constraints are not so severe as to generate many unfeasible solutions. In our particular case, the due date and

Algorithm 2 Chromosome to starting times conversion

Input: chromosome giving a sequence of operations
Output: starting time s_k for each operation k
$s_{\text{last}} = D_{\text{last}} - P_{\text{last}}$ (schedule the last operation in a "just-in-time" policy)
for each operation k to be scheduled, in decreasing sequence order, **do**
    if k is the only operation or the last operation of its job **then**
        $s_k = \min(D_k - P_k; s_{\text{next}(k)} - P_k)$
    else k is an operation followed by k' in its job, with drying time L in between:
        $s_k = \min(s_{k'} - P_k - L; \text{snext}(k) - P_k)$
    end if
end for

operations order constraints as well as the encoding used would make the use of a randomisation method unefficient. For this reason, we use the following method. Based on the instance data, a single initial solution is created. An algorithm sorts the jobs by increasing due date. The operations of jobs with the lowest due dates are scheduled first, and operations of other jobs can be introduced whenever the job with the lowest due date is in the drying inventory. Once the initial solution is created, two mutation operators are applied in order to generate a sufficient number of new offspring. These constitute the initial population introduced into the GA.

4.4. *Mutation operators*

Two different mutation operators are considered, namely the swap (Sevaux and Dauzère-Pérès, 2003) and insertion operators. The swap picks two random genes within the chromosome and exchanges them. The insertion picks a random gene and inserts it somewhere else in the chromosome. Since the chromosome size may vary depending on the number of jobs, the mutation scales accordingly by applying a number of swaps or large swaps equal to the number of jobs $|I|$ divided by ten. Figure 4 shows an example of how both operators work on a ten-jobs chromosome.

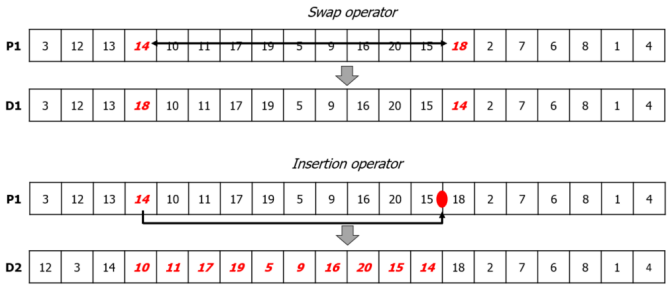

Figure 4. Swap (creates offspring D1) and insertion (creates offspring D2) operators and generated offspring

4.5. *Crossover operators*

Two types of crossovers are tested, namely the standard two-point crossover (Sevaux and Dauzère-Pérès, 2003) and the Linear Order Crossover (LOX) (Portmann, 1996).

The first parents are chosen using fitness proportionate selection, also known as roulette wheel selection, where the population of parent chromosomes is sorted by increasing fitness values (ascending order of objective function), meaning the chromosome with the lowest objective value is ranked first. Each parent is then assigned a probability from Reeves (1995) equal to

$$p(k) = \frac{2(W + 1 - k)}{W(W + 1)}$$

where k is the rank of the chromosome and W is the population size. The first parent is chosen using this discrete probability distribution. The fittest chromosomes thus have a higher probability of being

chosen as parents. The second parent is then selected randomly from the remaining population, thus maintaining diversity.

The standard two-point crossover chooses two random genes in the first parent and swaps them with the corresponding genes of the second parent, as represented in Figure 5 where offspring D1 and D2 are created from parents P1 and P2. Similarly to the way mutation operators scale, this operator is designed to execute this swapping manoeuvre a number of time equal to the number of jobs divided by ten.

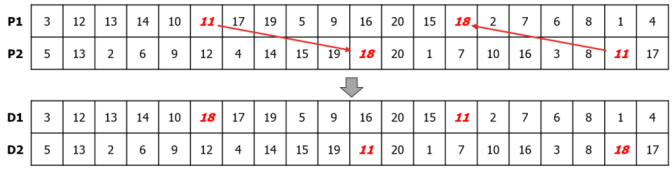

Figure 5. Two point standard crossover and generated offspring

The other crossover operator used is the LOX operator, which also chooses two random genes as crossover points. The partial sequence contained between those two points is transmitted to the offspring. The rest of the offspring is then filled with the missing genes from the other parent starting from the beginning of the chromosome, as shown in Figure 6. This operator has the merit of keeping

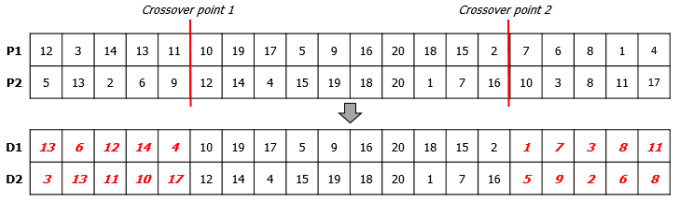

Figure 6. LOX operator and generated offspring

a part of the first parent intact, as well as the relative order from the second one, which is important in a problem where due dates severely constrain the ordering possibilities.

4.6. *Unfeasible solutions*

It is often useful in a GA to allow for unfeasible solutions to be part of the chromosome pool. Since optimal solutions are oftentimes found near the border of the search space, allowing for solutions located outside (but not too far from) these borders is a reasonable strategy.

To this end, a penalty is applied to the objective value of solutions that have one or more late job. This penalty is calculated with the following formula: $penalty = \max_{i \in \mathcal{I}}(D_i) \times \max_{i \in \mathcal{I}, j \in \mathcal{J}}(P_{ij})$, and is large enough to always favour feasible solutions over unfeasible ones. This makes sure that unfeasible solutions will be replaced over time by fitter feasible solutions. Additionally, a parameter `tolerance` defines the maximum total lateness allowed and removes the solutions that are too far from the search space. The value of `tolerance` decreases over time and unfeasible solutions are ultimately not allowed.

4.7. *Reproductive strategy*

Sevaux and Dauzère-Pérès (2003) refer to two specific methods for the reproductive strategy, i.e. how new populations are created during the application of a GA:

- Incremental replacement: each time a new offspring is generated, its fitness is evaluated and compared to the one of the current chromosomes in the population. If its fitness value is better than one or several chromosomes of the current population, it is inserted and the chromosome with the lowest fitness removed.
- Population replacement: a new generation of offspring is created which replaces the previous population, at the risk of losing more efficient chromosomes in the process.

Our reproductive strategy is hybrid, i.e. 20% of the best solutions are kept in the new population regardless of its status as offspring or parent. The rest of the population is then filled with random offspring. This helps maintain a pool of high fitness solutions in the population (which improves the quality of the offspring) while still allowing for diversity through the introduction of new offspring.

4.8. *Stopping mechanism*

The stopping mechanism is twofold. The algorithm stops if the best objective function value has not improved after `threshold` generations. This provides a good compromise between objective function improvement and computation time. Additionally, the algorithm stops when `Iteration number` pairs of parents have been selected, without creating any new population. This avoids spending too much time generating new solutions when it is too computationally demanding. It is especially relevant for large population sizes where producing enough feasible solutions can be difficult.

4.9. *Genetic algorithm structure*

As all components of this GA have been defined in the previous paragraphs, its overall structure, represented in Figure 7, is now explained.

After the initialisation phase in which the initial population is created, the GA iterations start. A pair of chromosomes is selected, and has a probability p_1 of being subjected to the swap operator (each chromosome is mutated independently). The insertion operator is then applied with a probability p_2 (meaning that any given pair of chromosome can be subjected to either zero, one or two mutations). The resulting chromosomes then have a probability p_3 of being subjected to a standard two-point crossover (as parents), followed by a probability p_4 of being subjected to the LOX operator.

If those new chromosomes are considered feasible (which depends on the `tolerance` allowed), they are kept in the offspring generation. If more offspring need to be generated to complete the population, the iteration counter `NbIterations` is incremented and a new pair of parents is selected and submitted to the operators. If the iteration counter reaches `Iteration number` before a new population has been created, the algorithm stops and the best current solution is returned.

Once a number of offspring equal to the population size have been accepted, the reproductive strategy is applied and the new population is created. The iteration counter is reset, and the best objective value of the new population is compared to the previous one. If it has improved, the generation counter is reset and a new loop begins. If not, it is incremented and a new loop begins, until the generation counter reaches the threshold described in subsection 4.8 and the best solution is returned.

5. Numerical experiments and results

5.1. *Instances generation*

An instance generator has been coded in C++. The data required to generate an instance consists of:

- the number of jobs n;
- the maximum number of operations per job m;
- the number of different types of operations $|\mathcal{J}|$.
- the distribution from which the number of operations N_i of each job i are drawn.
- the minimum drying time L;

The rest of the instance data is generated as follow:

- Using the chosen distribution, each job between 1 and n is assigned a number of operations N_i between 1 and m. Each operation is then assigned an operation type represented by an integer between 1 and $|\mathcal{J}|$ using a second discrete distribution. The result is a matrix of size $n \times m$ containing the details of each job.
- Q_i is drawn following a normal distribution $\mathcal{N}(20, 5)$.
- Processing time P_{ij} of operation j of job i is assumed to be a linear function of lot size Q_i, and all the operations of one job are assumed to have the same length : $P_{ij} = \gamma_i Q_i$ for all $j \in \{1, ..., N_i\}$. We set $\gamma_i = 1$ for all jobs i without a loss of generality.
- A lower and upper bound are then calculated for the determination of the due dates. The lower bound lb_i of job i is defined as $lb_i = 2 \times \sum_{j \in \mathcal{J}} P_{ij} + (N_i - 1) \times L$ for all i in $\mathcal{I}$, which is twice the sum of the processing and drying times necessary for job i. A time horizon for the

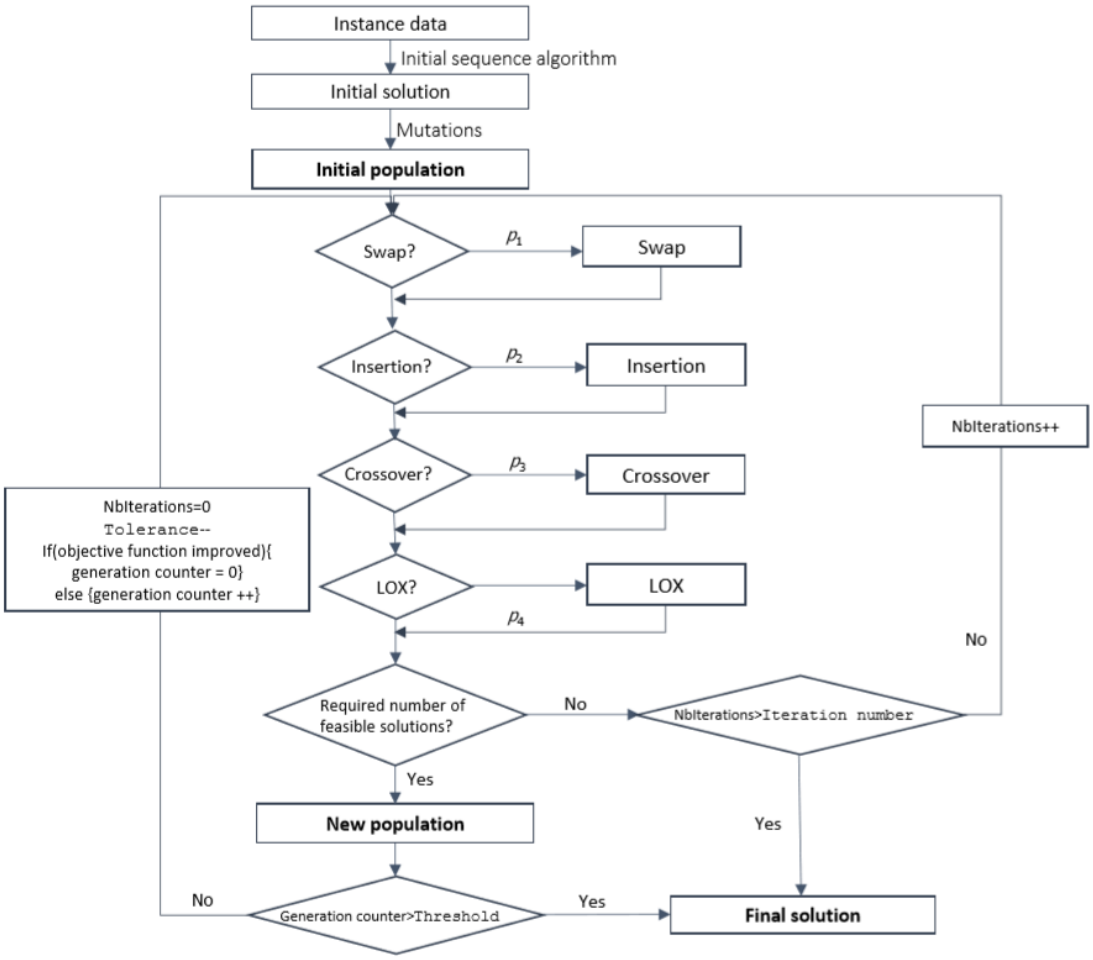

Figure 7. Genetic algorithm structure representation

problem is then set as: $M = 1.5\left(\sum_{i\in\mathcal{I}}\sum_{j=1}^{N_i} P_{ij} + L \times \sum_{i\in\mathcal{I}} N_i\right)$. This value was chosen big enough to ensure that a sufficient number of instances would be solvable, but would remain sufficiently constrained. The due dates D_i are then generated using a uniform distribution $\mathcal{U}(lb_i, M)$.

While this ensures that the first operation of a schedule is always feasible, note that it does not guarantee that every generated instance can be solved. A screening is done to remove unsuitable instances until the targeted number of solvable ones has been reached.

Sets of instances were generated with a maximum of two operations per job. Parameters were set at $n \in 10, 30$ and m=2 with an even repartition between both types (meaning that each operation has a fifty percent chance of being of type 1 or type 2). Different combinations for the distribution of variables N_i led to different configurations detailed in Table 3. For example, an 80%-20% distribution for N_i means that 80% of the jobs will consist of only one operation, while 20% will have two.

Table 3. Instance configurations

$\lvert\mathcal{I}\rvert$	n	m	L	Distribution of N_i
2	10	2	4	80%-20%
				50%-50%
	30			20%-80%

5.2. *MILP model experiments*

First experiments are carried out on the MILP model using the IBM ILOG CPLEX solver 12.6.2.0 version, in order to verify and validate it. A total of 110 instances are solved, namely thirty instances with ten jobs for each configuration and twenty instances of 30 jobs with the 80-20 configuration. Maximum solving time for a point is set to thirty minutes; in case the optimum is not reached after this time, a lower bound is returned by CPLEX. This experiment is carried-out on a bi-objective model and thus returns a Pareto front. Experiments results are shown in Table 4 for the case of the $z_{percent}$ point.

Table 4. MILP experiments results - average values for the $z_{percent}$ point

n	Distrib of N_i	Waste reduc. (%)	Inventory inc. (%)	Pareto size	CPU time (s)
10	80-20	14.4%	6.6%	3.34	0.1
10	50-50	37.4%	11.1%	4.55	118
10	20-80	36.2%	12.1%	5.6	595
30	80-20	25.9%	12.3%	9.05	1680

Table 4 shows that it is possible to significantly reduce waste generation with a relatively low increase in inventory. However, computation times increase exponentially with the number of operations, resulting in impractical computation times for instances of 30 or more jobs, where it can take more than a half hour to get a point of the Pareto front.

5.3. *GA parameters definition*

These experiments were designed in order to find the parameter values that ensure that the algorithm is efficient in terms of computation time and solution quality. From the seven main parameters affecting the algorithm performance, a Taguchi table (Roy, 2001) is constructed.

These parameters were considered in 8 experiments sets, switching between two extreme values and applied to instances of each configuration of 10 jobs. Table 5 detail these experiments values and the obtained results. Since decision-makers tend to prioritise the economic criterion when adjusting the schedule, the parameters were adjusted based on the resultant inventory values only.

Experimental values for each parameter were chosen after initial experiments and in order to consider a large possible range. The effect of each parameters on algorithm results is supposed to be linear, and only interactions of the first order between parameters are considered. The resulting parameters values are the one kept for the following experiments.

Table 5. Taguchi table parameter values

	State 1	State 2	Results
Population size	10	30	10
Swap rate	0.5	0.8	0.8
Insertion rate	0.5	0.8	0.8
Crossover rate	0.1	0.5	0.1
LOX rate	0.1	0.5	0.1
Threshold	1000	3000	3000
Iteration number	1000	3000	3000

5.4. *Results*

30 problems of each configuration with ten jobs were solved with both MILP and GA, using a computer with an Intel i5 6200 2.3 GHz processor and 8 GB of RAM. Additionally, ten instances with thirty jobs of each configuration were solved with the GA, resulting in a total of 120 instances tested. Each instance was solved three consecutive times, and the average value of these three rounds was kept as the result for the instance. Results for a single objective problem (with inventory only) are compared in Table 6. The gap for any given instance is $\text{gap} = \frac{z^{\text{GA}}_{\text{inventory}} - z^{\text{MILP}}_{\text{inventory}}}{z^{\text{MILP}}_{\text{inventory}}}$ and the CPU time gain is calculated similarly. Table 6 gives the average results over 30 instances; the last column indicates for how many instances (over the 30 considered) the GA found the optimal solution. For 30 job instances where no MILP result is available for comparison, the GA average CPU time and standard deviation are presented.

Results show that the GA reaches the optimal solution a majority of the time, with an average gap from the optimal not exceeding 2%. The GA is outperformed by the MILP regarding computation time for small instances, but becomes more efficient timewise as the number of jobs and operations increases.

Table 6. Single objective value and CPU time comparison for thirty instances

n	Distrib of N_i	Average gap (%)	Average CPU time gain (%)	Nb opt/nb total
10	80-20	1%	10000%	27/30
10	50-50	1,8%	800%	24/30
10	20-80	1%	-161%	20/30
n	Distrib of N_i	Average CPU time (s)	Standard deviation	
30	80-20	48,2	40,5	
30	50-50	58,5	34,5	
30	20-80	50,0	47,2	

Regarding the biobjective problem, the same set of 10 job instances is solved by both MILP and GA, as well as twenty 30 jobs instances with the 80%-20% configuration. The MILP is tested using the ε-constraint approach described in algorithm 1, and the GA with a weighted sum. Both methods allow to use a single-objective optimization method to solve a bi-objective one through successive algorithm runs. Each instance is solved twenty consecutive times in which an increasing setup cost is applied, i.e. the fitness of a chromosome becomes not only based on the inventory, but also on the number of setups. The increasing setup cost allows for the GA to find solutions with gradually decreasing numbers of setups, thus forming a Pareto front. Note that the reduced computing time of the GA is even more impactful when determining Pareto fronts, since multiple solvings need to be carried out for a same instance. When the total number of operations is greater than 40, obtaining a Pareto front using the MILP can result in computing times of several hours per instance, which is no longer suitable for any practical application. The GA is able to solve instances of a hundred jobs, which is the size of an industrial instance for a day of production, in less than an hour.

Table 7 contains the mean values and standard deviation for the trade-off points that were obtained from each instance using both MILP (ε-constraint method) and GA (weighted sum). Column "distance" contains the average distance to the ideal point previously defined, and column "CPU time" shows the average CPU time in seconds consumed for obtaining this particular point. Both the number of setups and inventory increase when the number of jobs with two operations increases (i.e. when the distribution switches from 80% - 20% towards 20% - 80%), which is a result of an increased number of operations, regardless of the solving method used. Similarly, table 8 shows the same results for the $z_{percent}$ point. The results are consistent over the different configurations tested, and the obtained trade-off points all provide effective alternative schedules for both MILP and GA. The lower inventory observed for the GA trade-off and $z^{\text{min}}_{\text{inventory}}$ points are due to its shorter Pareto front. Since they are located farther to the right than the points obtained with the exact methods, a larger number of setups occurs in the GA solutions, which allows for a better solution in terms of inventory.

Table 7. Characteristics of the trade-off point (standard deviation in parenthesis)

	n	Distrib of N_i	$z^{\text{trade-off}}_{\text{setup}}$	$z^{\text{trade-off}}_{\text{inventory}}$	Setup % reduc.	Inventory % inc.	CPU time (s)	Pareto size
MILP	10	80-20	3.13	3056	27.1 (22.7)	69.9 (95.5)	0.45 (1.2)	3.34
		50-50	3.82	5560	42.5 (19)	51.7 (62.1)	215 (539)	4.55
		20-80	4.38	7150	46.1 (19.2)	106.8 (286)	638 (737)	5.6
	30	80-20	8.9	16764	38.5 (16.1)	54.8 (73.2)	1714 (384)	9.05
GA	10	80-20	4	2360	12.27 (17.2)	25.3 (59.2)	16.1 (8.5)	2.55
		50-50	4.7	4052	20.6 (20.9)	12.2 (20.8)	24 (12)	3.34
		20-80	5.75	5981	28.5 (19.48)	41.4 (136)	28.3 (12.18)	3.85
	30	80-20	11.5	18518	22.4 (16)	25.5 (35.6)	95 (55.5)	5.35

Table 8. Characteristics of the $z_{percent}$ point (standard deviation in parenthesis)

	n	Distrib of N_i	$z^{\text{percent}}_{\text{setup}}$	$z^{\text{percent}}_{\text{inventory}}$	Setup % reduc.	Inventory % inc.	CPU time (s)	Pareto size
MILP	10	80-20	3.86	2215	14.4 (20.66)	6.6 (11.4)	0.10 (0.67)	3.34
		50-50	4.3	4385	37.4 (16.8)	11.1 (9)	118 (371)	4.55
		20-80	5.4	5852	36.2 (22.9)	12.1 (11.4)	595 (762)	5.6
	30	80-20	11	13179	25.9 (13.7)	12.3 (8.9)	1680 (392)	9.05
GA	10	80-20	3.9	2194	16.58 (7.9)	6.6 (10.5)	15.3 (7.9)	2.55
		50-50	4.2	4063	31.9 (19.1)	10.45 (9.65)	24 (19)	3.34
		20-80	5.2	6095	36.4 (23.8)	11.9 (11.9)	26.9 (12.2)	3.85
	30	80-20	12.8	16537	15.2 (17.1)	5.9 (9.34)	88.4 (58)	5.35

Figure 8 shows the average values of the four points $(z^{\text{min}}_{\text{inventory}}, z^{0}_{\text{setup}})$, $(z^{0}_{\text{inventory}}, z^{\text{min}}_{\text{setup}})$, $(z^{\text{trade-off}}_{\text{inventory}}, z^{\text{trade-off}}_{\text{setup}})$ and $(z^{\text{percent}}_{\text{inventory}}, z^{\text{percent}}_{\text{setup}})$ for the MILP (full line) and GA (dashed line) and for each configuration. As can be seen, the number of setups can be reduced to a certain extent with a relatively low increase in inventory. When reaching the trade-off point, any additional reduction of

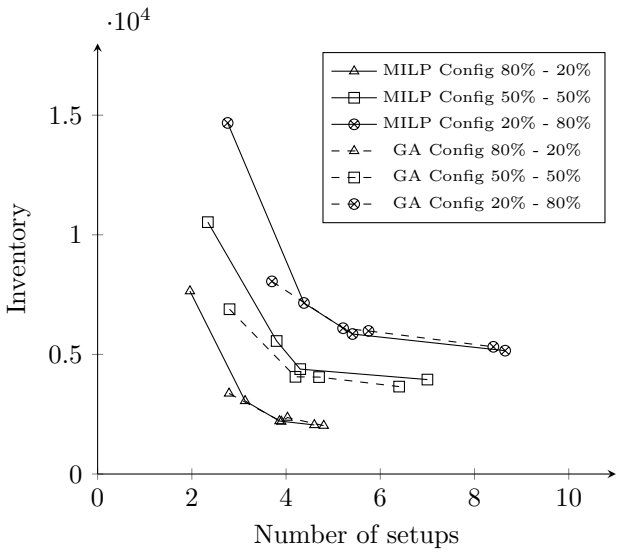

Figure 8. Average trade-off and extreme points for 30 instances of ten jobs

setups results in a greater increase. This is visible in the rising steepness of the curve when reaching the trade-off point. Tracing the same curves for larger instances or different drying times yields similar results. The additional z_{percent} point also provides an efficient schedule for decision-makers since it is based on direct waste reduction and inventory increase percentages. Thus, it might be beneficial for the decision-maker to consider this trade-off schedule (or other solutions located between this one and the $(z^{\min}_{\text{inventory}}, z^{0}_{\text{setup}})$ point on the Pareto front, such as the z_{percent} point) when planning the production, in accordance with the respective prices of inventory-keeping and waste treatment.

5.5. *Managerial implications*

As this paper deals with industrial manufacturing and is based on a realistic production plant, it is important to discuss the implication of the results for the decision-maker. Besides the two extreme points minimising inventory or setup waste, two points of interest are proposed. When using a distance to the ideal point criteria, the $z_{\text{trade-off}}$ point is the most efficient. However, when looking at the actual percentage increase and decrease regarding inventory and waste, the z_{percent} point is more suited to managerial needs. While the $z_{trade-off}$ point tends to produce less waste (10% more setups reduction than the $z_{percent}$ point on average), it also substantially increases inventory (a 71% inventory increase on average using MILP data), which might be unacceptable for decision-makers. On the other hand, the z_{percent} point increases inventory by only 10% on average for a 28% waste reduction, making it more suited to real-life decisions. If waste-management costs are accounted for, using a schedule based on the z_{percent} point could actually reduce overall costs. Instances of 100 jobs, which is the size of a daily schedule in real cases, have been solved using the GA, with a possible decrease of waste of 5% for an increase in inventory of 1.5%.

6. Conclusion

This paper investigates a single-machine scheduling problem with coupled-tasks aiming at reducing waste generation due to setups and costs induced by inventory, under the constraint of due dates. This problem has been shown to be NP-Hard, and the MILP model present excessive computation times for industrial-sized instances. Therefore, a GA is proposed and tested, and performances from both GA and MILP are compared. The multiobjective aspect of the problem is accounted for through the ε-constraint method as well as a weighted sum method, and the proposition of trade-off solutions to the decision-maker. Results show that the GA performs well on large instances that the MILP cannot solve in reasonable time, as well as in providing trade-off points out of a Pareto front. Regardless of the solving method used, we show that taking the waste criteria into account when designing the production schedule is an effective lever to improve the environmental performance of the production

system without compromising too much on economical criteria such as inventory costs. Generated waste can be decreased by a significant percentage at the expense of higher inventory costs via the proposed trade-off and percent point. Alternatively, all points of the Pareto front can be considered depending on the decision-makers priorities. To assess the actual cost of waste management, alternatives schedules reducing waste generation can be economically beneficial to companies.

For future work, several developments can be considered. Developing a fully fledged multiobjective GA will be the topic of future research to make full use of the GA potential for multiobjective optimisation, without using the ε-constraint or weighted sum methods. Calculating accurate lower bounds for large instances, using e.g. a lagrangian relaxation, would also allow for a better knowledge about the performance of the GA compared to the MILP when the number of jobs increases. Additionally, the proposed model and GA can easily be adapted to tackle other cases of coupled-tasks scheduling problems. while the experiments considered in this paper are limited to a maximum of two operations per job and two operation types, the developed model and GA can be used for higher values of m and $|\mathcal{J}|$. Similarly, new distributions regarding the operation type could be experimented on, as well as higher drying times compared to the mean processing time, or specific minimum drying times L_{ij} to better represent the reality and variety of industrial production plants. An extension to a multi-machine environment could also be useful for cases with multiple painting lines sharing common operation types.

References

Adonyi, R., G. Biros, T. Holczinger, and F. Friedler (2008). Effective scheduling of a large-scale paint production system. *Journal of Cleaner Production 16*(2), 225–232.

Ahr, D., J. Békési, G. Galambos, M. Oswald, and G. Reinelt (2004). An exact algorithm for scheduling identical coupled tasks. *Mathematical Methods of Operations Research 59*(2), 193–203.

Amrouche, K. and M. Boudhar (2016). Two machines flow shop with reentrance and exact time lag. *RAIRO - Operations Research 50*(2), 223–232.

Amrouche, K., M. Boudhar, M. Bendraouche, and F. Yalaoui (2017). Chain-reentrant shop with an exact time lag: new results. *International Journal of Production Research 55*(1), 285–295.

Araujo, S. A. d., K. C. Poldi, and J. Smith (2014). a Genetic Algorithm for the One-Dimensional Cutting Stock Problem With Setups. *Pesquisa Operacional 34*(2), 165–187.

Arbiza, M. J., A. Bonfill, G. Guillén, F. D. Mele, A. Espuña, and L. Puigjaner (2008). Metaheuristic multiobjective optimisation approach for the scheduling of multiproduct batch chemical plants. *Journal of Cleaner Production 16*(2), 233–244.

Biel, K. and C. H. Glock (2016). Systematic literature review of decision support models for energy-efficient production planning. *Computers and Industrial Engineering 101*, 243–259.

Blasco, X., J. M. Herrero, J. Sanchis, and M. Martínez (2008). A new graphical visualization of n-dimensional Pareto front for decision-making in multiobjective optimization. *Information Sciences 178*(20), 3908–3924.

Blazewicz, J. (2010). Scheduling of coupled tasks with unit processing times. *Journal of Scheduling 13*(5), 453–461.

Blazewicz, J., G. Pawlak, M. Tanas, and W. Wojciechowicz (2012). New algorithms for coupled tasks scheduling a survey. *RAIRO - Operations Research 46*(4), 335–353.

Brauner, N., G. Finke, V. Lehoux-Lebacque, C. Potts, and J. Whitehead (2009). Scheduling of coupled tasks and one-machine no-wait robotic cells. *Computers and Operations Research 36*(2), 301–307.

Chen, J.-S. and J. Chao-Hsien Pan (2006). Integer programming models for the re-entrant shop scheduling problems. *Engineering Optimization 38*(5), 577–592.

Cho, H.-M., S.-J. Bae, J. Kim, and I.-J. Jeong (2011, 10). Bi-objective scheduling for reentrant hybrid flow shop using Pareto genetic algorithm. *Computers & Industrial Engineering 61*(3), 529–541.

Condotta, A. and N. V. Shakhlevich (2012). Scheduling coupled-operation jobs with exact time-lags. *Discrete Applied Mathematics 160*(16-17), 2370–2388.

Courtad, B., K. Baker, M. Magazine, and G. Polak (2017). Minimizing flowtime for paired tasks. *European Journal of Operational Research 259*(3), 818–828.

Croce, F. D. (1995). A genetic algorithm for the Job-shop problem. *Science 22*(1), 15–24.

Cui, Y. and Y. Yang (2010). A heuristic for the one-dimensional cutting stock problem with usable leftover. *European Journal of Operational Research 204*(2), 245–250.

Deb, K., S. Agrawal, A. Pratap, and T. Meyarivan (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. *International Conference on Parallel Problem Solving From Nature*, 849–858.

Dugardin, F., F. Yalaoui, and L. Amodeo (2010). New multi-objective method to solve reentrant hybrid flow shop scheduling problem. *European Journal of Operational Research 203*(1), 22–31.

El Amraoui, A. and K. Mesghouni (2014). An evolutionary approach for multi-objective optimization

in Cyclic Hoist Scheduling Problem. *Proceedings - 2014 International Conference on Control, Decision and Information Technologies, CoDIT 2014*, 201–206.

European Commission (2000). Commission Decision on the European List of Waste (COM 2000/532/EC). *Official Journal of the European Communities 2000D0532*(01.01.2002), 1–31.

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

Giret, A., D. Trentesaux, and V. Prabhu (2015). Sustainability in manufacturing operations scheduling: A state of the art review. *Journal of Manufacturing Systems 37*, 126–140.

Golfeto, R. R., A. Moretti, and L. S. Neto (2009). A genetic symbiotic algorithm applied to the one-dimensional cutting stock problem. *Pesquisa Operacional 29*(January 2009), 365–382.

Graham, R. L., E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan (1979). Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey.

Gupta, J. N. D. (1996). Comparative evaluation of heuristic algorithms for the single machine scheduling problem with two operations per job and time-lags. *Journal of Global Optimization 9*(3), 239–253.

Holland, J. H. (1992). Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. *MIT Press 1*(1), 211.

Konak, A., D. W. Coit, and A. E. Smith (2006). Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering and System Safety 91*(9), 992–1007.

Le Hesran, C., A.-l. Ladier, V. Botta-Genoulaz, and V. Laforest (2018). Operations scheduling for waste minimization : a review. *Journal of Cleaner Production 206*, 211–226.

Le Hesran, C., A.-L. Ladier, V. Botta-genoulaz, and V. Laforest (2018). Reducing waste in manufacturing operations : a mixed integer linear program for bi-objective scheduling on a single-machine with coupled-tasks. In *16th IFAC Symposium on Information Control Problems in Manufacturing*.

Li, H. and H. Zhao (2007). Scheduling coupled-tasks on a single machine. *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Scheduling, CI-Sched 2007*, 137–142.

Lin, C., K. B. Haley, and C. Sparks (1995). A comparative study of both standard and adaptive versions of threshold accepting and simulated annealing algorithms in three scheduling problems. *European Journal of Operational Research 83*(2), 330–346.

Lin, C. K. and K. B. Haley (1993). Scheduling two-phase jobs with arbitrary time lags in a single-server system. *IMA Journal of Management Mathematics 5*(1), 143–161.

Liu, Y., H. Dong, N. Lohse, and S. Petrovic (2016). A multi-objective genetic algorithm for optimisation of energy consumption and shop floor production performance. *International Journal of Production Economics 179*, 259–272.

Lowell Center for Sustainable Production (1998). Sustainable production: A working definition. Informal meeting of the committee members.

Malik, M. M., M. Qiu, and J. Taplin (2009). An integrated approach to the lot sizing and cutting stock problems. *IEEM 2009 - IEEE International Conference on Industrial Engineering and Engineering Management*, 1111–1115.

Manne, A. S. (1960). On the Job-Shop Scheduling Problem. *Operations Research 8*(2), 219–223.

Mavrotas, G. (2009). Effective implementation of the ϵ-constraint method in Multi-Objective Mathematical Programming problems. *Applied Mathematics and Computation 213*(2), 455–465.

Meziani, N., M. Boudhar, and A. Oulamara (2018). PSO and simulated annealing for the two-machine flowshop scheduling problem with coupled-operations. *European Journal of Industrial Engineering 12*(1), 43–66.

Nonas, S. L. and A. Thorstenson (2008). Solving a combined cutting-stock and lot-sizing problem with a column generating procedure. *Computers and Operations Research 35*(10), 3371–3392.

Orman, A. J. and C. N. Potts (1997). On the complexity of coupled-task scheduling. *Discrete Applied Mathematics 72*(96), 141–154.

Pezzella, F., G. Morganti, and G. Ciaschetti (2008). A genetic algorithm for the Flexible Job-shop Scheduling Problem. *Computers and Operations Research 35*(10), 3202–3212.

Portmann, M. C. (1996). Genetic algorithms and scheduling: a state of the art and some propositions. *Proceedings of the Workshop on Production Planning and Control 9*(11), 1–24.

Potts, C. N. and J. D. Whitehead (2007). Heuristics for a coupled-operation scheduling problem. *Journal of the Operational Research Society 58*(10), 1375–1388.

Pulluru, S. J., R. Akkerman, and A. Hottenrott (2017). *Integrated production planning and water management in the food industry: A cheese production case study*, Volume 40. Elsevier Masson SAS.

Reeves, C. R. (1995). A genetic algorithm for flowshop sequencing. *Computers and Operations Research 22*(1), 5–13.

Reeves, R. C. (1996). Feature Article-Genetic Algorithms for the Operations Researcher. *INFORMS journal on computing 9*(January 2017), 231–250.

Rifai, A. P., H.-T. Nguyen, and S. Z. M. Dawal (2016, 3). Multi-objective adaptive large neighborhood

search for distributed reentrant permutation flow shop scheduling. *Applied Soft Computing 40*, 42–57.

Roy, R. K. (2001). *Design of Experiments Using the Taguchi Approach: 16 Steps to Product and Process Improvement*.

Ruiz, R. and C. Maroto (2006). A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. *European Journal of Operational Research 169*(3), 781–800.

Sanmartí, E., L. Puigjaner, T. Holczinger, and F. Friedler (2002). Combinatorial framework for effective scheduling of multipurpose batch plants. *American Institute of Chemical Engineers Journal 48*(11), 2557–2570.

Sevaux, M. and S. Dauzère-Pérès (2003). Genetic algorithms to minimize the weighted number of late jobs on a single machine. *European Journal of Operational Research 151*(2), 296–306.

Shapiro, R. D. (1980). Scheduling coupled tasks. *Naval Research Logistics Quarterly 27*(3), 489–498.

Simonin, G., B. Darties, R. Giroudeau, and J. C. König (2011). Isomorphic coupled-task scheduling problem with compatibility constraints on a single processor. *Journal of Scheduling 14*(5), 501–509.

Vaklieva-Bancheva, N. G. and E. G. Kirilova (2010). Cleaner manufacture of multipurpose batch chemical and biochemical plants. Scheduling and optimal choice of production recipes. *Journal of Cleaner Production 18*(13), 1300–1310.

Wagner, H. M. (1959). An Integer Linear Programming Model for Machine Scheduling. *Naval Research Logistics Quarterly 6*(2), 131–140.

Wilson, J. M. (1989). Alternative Formulations of a Flow-shop Scheduling Problem. *Journal of the Operational Research Society 40*(4), 395–399.

Winter, . and P. Baptiste (2007). On scheduling a multifunction radar. *Aerospace Science and Technology 11*(4), 289–294.

Wu, T., K. Akartunali, R. Jans, and Z. Liang (2017). Progressive selection method for the coupled lot-sizing and cutting-stock problem. *INFORMS Journal on Computing 29*(3), 523–543.

You, C.-J. L. and Chii-Tsuen (1992). An Improved Formulation for the Job-Shop Scheduling Problem. *The Journal of the Operational Research Society 43*(11), 1047–1054.

Zhang, Q., H. Manier, and M.-A. Manier (2012, 7). A genetic algorithm with tabu search procedure for flexible job shop scheduling with transportation constraints and bounded processing times. *Computers & Operations Research 39*(7), 1713–1723.

Zhang, R. (2018). Environment-aware production scheduling for paint shops in automobile manufacturing: A multi-objective optimization approach. *International Journal of Environmental Research and Public Health 15*(1).

Zhang, R. and R. Chiong (2016). Solving the energy-efficient job shop scheduling problem: A multi-objective genetic algorithm with enhanced local search for minimizing the total weighted tardiness and total energy consumption. *Journal of Cleaner Production 112*, 3361–3375.

Appendix A. Starting time definition algorithm optimality proof

Our aim is to prove that algorithm 2 returns starting times s_i that minimise the inventory objective function $z_{inventory}$ for a given sequence of operations input.

Let us define two subsets $\mathcal{K}^1$ and $\mathcal{K}^2$ from the set $\mathcal{K}$ of all operations, such that:

- $\mathcal{K}^1$ is the set of operations that are the only ones in their job and operations that are the last of their job;
- $\mathcal{K}^2$ is the set of operations after which a drying time is needed.

We have $\mathcal{K}^1 \cup \mathcal{K}^2 = \mathcal{K}$ and $\mathcal{K}^1 \cap \mathcal{K}^2 = \emptyset$

The inventory objective function

$$z_{inventory} = \sum_{i \in \mathcal{I}} Q_i * e_i + \sum_{i \in \mathcal{I} | N_i > 1} \sum_{j=1}^{N_i} Q_i (t_{ij} - L)$$

can be rewritten as:

$$z_{inventory} = \sum_{k \in \mathcal{K}^1} Q_k * e_k + \sum_{k \in \mathcal{K}^2} Q_k (t_k - L)$$

with $e_k = D_k - s_k - D_k$ and $t_k = s_{k'} - s_k - P_k - L$.

Similarly, constraints (1), (3) and constraint set (4) and (5) can be rewritten as:

$$\begin{aligned} (1) &\Rightarrow \quad s_{k'} - s_k - P_k \geq L \quad \forall k \in \mathcal{K}^2 \quad (12) \\ (3) &\Rightarrow \quad s_k + P_k \leq D_k \quad \forall k \in \mathcal{K}^1 \quad (13) \\ (4) + (5) &\Rightarrow \quad s_k \leq s_{next} - P_k \quad \forall k \in \mathcal{K} \quad (14) \end{aligned}$$

where k' is the next operation of the same job.

We wish to prove that for a given operations sequence, assigning the starting times s_k so that:

$$s_k = \begin{cases} D_{last} - P_{last} \text{ if k is the last operation} \\ \min\left(D_k - P_k; s_{next(k)} - P_k\right) \text{ if } k \in \mathcal{K}^1 \\ \min\left(s_{k'} - P_k - L; s_{next(k)} - P_k\right) \text{ if } k \in \mathcal{K}^2 \end{cases}$$

minimizes $z_{inventory}$. Since Q_k is always positive, minimising $z_{inventory}$ means minimising e_k and $(t_k - L)$

For each option, we have:

- If it is the last operation, we have $e_{last} = D_{last} - s_{last} - P_{last}$. If $s_{last} = D_{last} - P_{last}$, $e_k = 0$, which is the minimum possible value (since e_k is always positive).
- If $k \in \mathcal{K}^1$, constraints (13) and (14) apply. The value assigned to s_k is $\min(D_k - P_k; s_{next(k)} - P_k)$:
  - If $D_k - P_k \leq s_{next(k)} - P_k$, then $s_k = D_k - P_k$ and $e_k = 0$, which is the minimum value.
  - If $D_k - P_k \geq s_{next(k)} - P_k$, then $s_k = s_{next(k)} - P_k$. Since $e_k = D_k - s_k - P_k$ and D_k and P_k are fixed, minimising e_k is equivalent to maximising s_k. Since $s_k \leq s_{next(k)} - P_k$ due to constraint (14), e_k is at its minimum possible value.
- If $k \in \mathcal{K}^2$, constraints (12) and (14) apply. The value assigned to s_k is $\min(s_{next(k)} - P_k; s_{k'} - P_k - L)$:
  - If $s_{next(k)} - P_k \leq s_{k'} - P_k - L$, then $s_k = s_{next(k)} - P_k$ and $t_k = L$, which is the minimum value.
  - If $s_{next(k)} - P_k \geq s_{k'} - P_k - L$, then $s_k = s_{k'} - P_k - L$. Since $t_k = s_{k'} - s_k - P_k - L$, and P_k, $s_{k'}$ and L are fixed, minimising t_k is equivalent to maximising s_k. Since $s_k \leq s_{k'} - P_k - L$ due to constraint (12), t_k is at its minimum possible value.

Additionally, since by definition any previously scheduled operation $next(k)$ is scheduled with the maximum possible $s_{next(k)}$, the starting time s_k can also be set as high as possible (since $s_k \leq s_{next} - P_k$)