Contents lists available at GrowingScience

Journal of Project Management

homepage: www.GrowingScience.com

Scheduling parallel extrusion lines

Fayez F. Boctor[a*], Dhiaeddine Zaatour[a] and Jacques Renaud[a]

[a]*Centre interuniversitaire de recherche sur les réseaux d'entreprises, la logistique et le transport, Faculté des sciences de l'administration, Université Laval, Canada*

C H R O N I C L E

Article history:
Received: September 1, 2023
Received in revised format: October 28, 2023
Accepted: November 17, 2023
Available online:
November 17, 2023

Keywords:
Production planning
Neighborhood search heuristics
Sequencing and scheduling

A B S T R A C T

This paper introduces the problem of scheduling jobs on parallel plastic extrusion lines where each line is composed of one or more than one extruder. Although there are some similarities between the introduced problem and the non-identical parallel machines scheduling problems with sequence-dependent setup times, limited additional resources and machine eligibility restrictions, the problem considered in this paper is a generalization of the parallel machine scheduling problem. This is because in parallel machines scheduling each job requires only one machine but in our case some jobs require more than one machine. Thus, our problem reduces to the parallel machine scheduling problem if all jobs require only one machine. This paper describes the problem of scheduling parallel extrusion lines, its industrial context, and develops a mixed-linear formulation to model the problem. This formulation allowed solving instances of up to 15 jobs. In addition, we developed four metaheuristics: a simulated annealing algorithm, a tabu search heuristic, a genetic algorithm, and a greedy randomized adaptive search procedure. These metaheuristics can be used to solve real-life instances of the problem. A numerical experiment shows that the proposed metaheuristics produce excellent solutions. Some of the proposed simulated annealing adaptations and of the tabu search heuristics obtained solutions with less than 2% deviation from the optimum.

1. Introduction

Plastic extrusion is a manufacturing process in which raw plastic is melted and shaped into a continuous profile. The product is then coiled or cut to a desired length. Extrusion products include pipes, tubes, wire insulation and plastic sheets. The raw material usually consists of thermoplastic beads. As it is shown in Figure 1, raw material is fed through a hopper and vacuumed to the extruder barrel and heated until it reaches its melting temperature. It is then pushed forward by a rotating screw and passes through a die, shaping it to the desired profile. A cooling phase ensues where the plastic passes through a sealed-water bath. Upon exiting the bath, the material is coiled or cut to its desired length. Secondary operations, such as printing characters or drilling a hole, can also take place following the cooling stage. Figure 2 shows a typical extrusion line with one extruder. As it is shown in this figure, an extrusion line contains, in addition to the extruder (or the extruders, if more than one is needed), a cooling bath, a haul-off machine, a coiler or a cutting machine. Also, it may include a printing machine or a drill if needed. It is to be noted that some products require an extrusion line containing more than one extruder. Thus, the product passes through the first extruder and then through the second one. One important problem in extrusion facilities is the scheduling of the execution of several jobs induced from customer orders. The usual practice is to prepare a production schedule for the following week. In our industrial setting, a week schedule contains from 25 to 35 jobs. This research work was motivated by the fact that there are no scheduling tools or methods that have been used by the plastic extrusion plants to efficiently schedule their operations. Also, we have been told by our industrial partner that, often, they are not able to meet the due dates required by their customers. Furthermore, extrusion is the most important process in the plastic industry. More than 114 million tons of plastics are processed at this industrial process worldwide every year. It is

* Corresponding author.
E-mail address: fayez.boctor@fsa.ulaval.ca (F. F. Boctor)

doi: 10.5267/j.jpm.2023.11.002

estimated that the demand for plastic extruded products will increase with a rate of about 3% yearly[1].

The production facility of our industrial partner is equipped with several extruders of varying diameters. There are four 3.5 inches extruders, two 2.5 inches, two 1.75 inches and three 1.5 inches. For most jobs, only one extruder is used, although some jobs require two extruders. There are about 17 types of raw materials and products can come out in 60 different colors. Each job requires a specific die. These dies are expensive and only one of each type is present in the plant. In total, the plant possesses about 200 different dies. However, only 20 of them are used for most jobs.

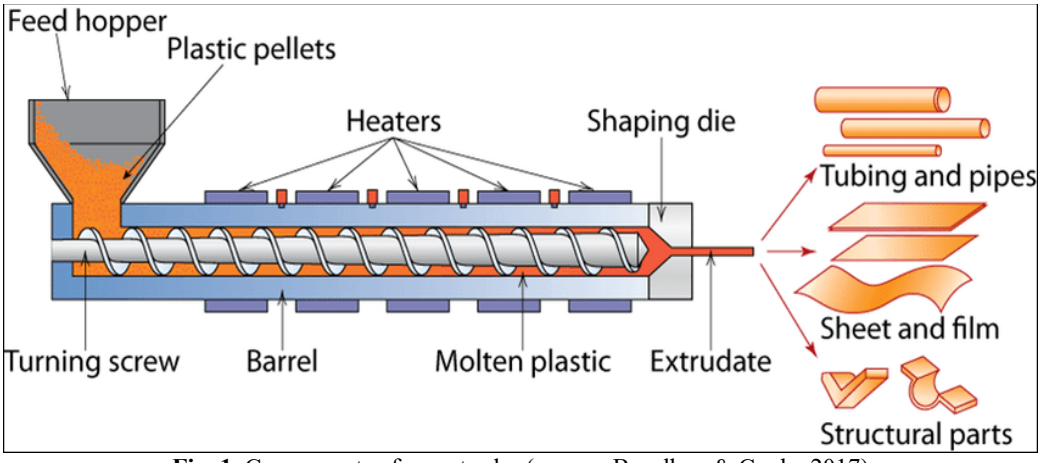

Fig. 1. Components of an extruder (source: Bacalhau & Cunha 2017)

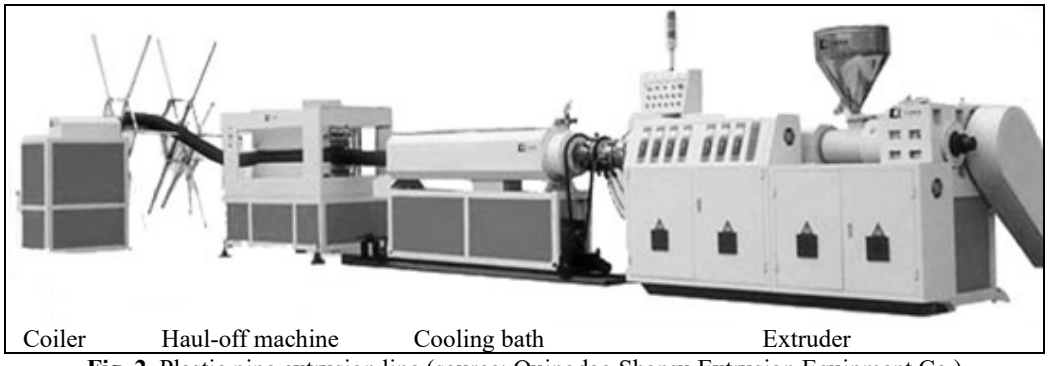

Fig. 2. Plastic pipe extrusion line (source: Quingdao Shansu Extrusion Equipment Co.)

Each job has a due date, is given some level of priority, and can be processed by some (but not all) extruders. Each of the possible extruders has a specific processing speed for that job. In the case of co-extrusion (two extruders are required for a single job), the primary extruder determines the speed of the process. It is important to note that if an extruder of a specific size (for example, 3.5 inches) might be able to process a job, not all extruders of that same size can process this same job. Also, extruders of the same size do not necessarily have the same processing speed. In addition, in some cases, extruders of different sizes can be used for the extrusion of the same job.

Changeover time between two jobs on a given extruder is sequence-dependent and symmetric, i.e., the changeover time if a job i follows another job j on the same extruder k is the same as if j follows i. Job changeover times are mainly influenced by the characteristics of consequent jobs. For simplicity, changeover times are estimated depending on five main aspects: the die or dies to be used, the color of the raw material, the raw material composition, the product shape option (coil or straight pieces) and necessary secondary operations like drilling or printing on the finished product. Changing a die on an extruder approximately requires 45 minutes. Raw material change requires cleaning the extrusion line and takes 120 minutes. Color changes while using the same raw material takes 30 minutes, and modifying the shape option can be done in 15 minutes. Installing or removing a drill at the end of the extrusion line is an operation that takes about 15 minutes. Notice that raw material change requires 120 minutes whether the new material is of the same color as the previous one or not. As an approximation, we estimate the time required to setup an extrusion line to perform a first job in the schedule to be about 75 minutes in case where only one extruder is required; if a co-extruder is needed for the line, we add extra 45

[1] See: https://www.kunststofenrubber.nl/download/Ceresana_Brochure_Market-Study_Plastic-Extrusion.pdf

minutes to this first setup time.

Usually, only one person is assigned to do a job changeover. So, the time for a given job changeover is the sum of the time to do all the required setup operations (die or dies change, material, or color change, etc.) for this job changeover. However, if we need to do two or more job changeovers on different extrusion lines at the same time, then a different person is assigned to do each line. For jobs requiring a co-extruder, we assume that only one person is assigned to execute the setup operations for the two extruders. The setup of the co-extruder consists only of replacing the extruder die if necessary.

The problem studied hereafter can be stated as follows. Given a number of jobs to schedule over a given planning horizon, the characteristics of each of them (i.e., its raw material, color, quantity to produce, final product shape, whether it requires one or two extruders, the extruders that can process the job, execution time on each of the possible extruders, its priority index, and its due date), the available extruders and the available set of dies; schedule these jobs in order to minimize the sum of tardiness weighted by the given priority index. Wisner and Siferd (1995) mentioned that this objective is the most common approach in practice. Framinan et al. (2014) indicated that "meeting due dates is probably the highest regarded indicator of customer's satisfaction (page 111)".

Based on what we observed in our industrial setting, we can assume that cooling baths, haul-off machines, coiling machines, drills and printers are available in sufficiently large numbers. Also, we can assume that there are enough people to do the required job changeovers. The closest problem to the one studied in this paper is the non-identical parallel machines scheduling problem with sequence-dependent setup times, limited additional resources (tools, labor, etc.) and machine eligibility restrictions (Afzalirad and Rezaeian, 2016). However, there is one important difference between this parallel machine scheduling problem and the problem studied in this paper. In our extrusion lines scheduling problem, a job may require a processing line of more than one processor (extruder) that operates in tandem. Thus, the problem studied hereafter can be seen as a generalization of the parallel machine scheduling problem, as it reduces to this parallel machine scheduling problem in the special case where all jobs require only one machine (extruder).

The paper is organized as follows. Section 2 provides the literature review of some problems related to the scheduling problem addressed in this paper. Section 3 proposes a mathematical formulation of this scheduling problem. Section 4 presents a numerical example to illustrate some of the characteristics of the studied problem while section 5 presents four metaheuristics as solution methods for this scheduling problem. Section 6 presents the numerical experiment and the results obtained by the proposed metaheuristics, and, finally, section 7 presents the conclusions of this research work.

2. Literature Review

To the best of our knowledge, this paper appears to be the first one to address our problem of scheduling jobs in a plastic extrusion facility. However, we can mention some publications that addressed other problems that share some but not all the characteristics of this problem. For example, Dastidar and Nagi (2005) addressed the production planning problem in an injection molding facility. In this industrial problem, the facility is composed of several parallel molding machines of different capabilities, manufacturing a number of different products, and some other resources are required to process these products. However, in their work the planning horizon is divided into several periods, there is a demand for each product at each period and the objective is to determine the quantities to be produced each period while minimizing the sum of the inventory holding costs and the backorder costs. The scheduling issues of these products are not addressed in this paper. Also, some publications addressed the problem of scheduling jobs on unrelated parallel machines. In its simplest version, the non-identical unrelated machine (UPM) problem is the problem of assigning n jobs to m machines where each job should be assigned to one out of the m non-identical machines with different processing speeds and a machine cannot process more than one job at a time. In this simple version of the problem, setup times are not taken into consideration. Fanjul and Ruiz (2011) proposed several metaheuristics to solve the UPM problem with the objective of minimizing the total completion time (make span). Bilyk and Mönch (2012) developed a variable neighborhood search heuristic to solve the problem and indicated that the proposed heuristic outperforms a genetic algorithm previously presented in Mönch et al. (2005). Rodriguez et al. (2013) presented an iterative greedy heuristic to solve large-scale problems with the objective of minimizing the total weighted completion times. Lee et al. (2018) considered the UPM problem with the objective of minimizing total tardiness while each of the machines needs preventive maintenance that should be started within a given cumulative working time limit after the previous maintenance. Wang and Alidaee (2019) proposed a sequential improvement local search algorithm using multiple-jump strategy embedded within tabu search (TS) and showed that the proposed algorithm outperformed the iterative greedy algorithm of Rodriguez at al. (2013).

Another version of the problem is the unrelated parallel machines scheduling with sequence dependent or machine-dependent setup times (UPMS). Allahverdi (2015) reviewed about 500 papers dealing with this problem. For example, Lee and Pinedo (1997) presented a three-phase heuristic to minimize the sum of the weighted tardiness of jobs on identical parallel machines. The first phase is a preprocessing procedure, the second phase is a solution construction heuristic, and the third phase uses the simulated annealing algorithm to improve the solution obtained in the second phase. Bilge et al. (2004) proposed a tabu search heuristic to solve the problem where the objective is to minimize total job tardiness. They

investigated several tabu search strategies and showed that they were able to improve the best-known results for some problems obtained from the literature. Logendran et al. (2007) proposed six tabu search algorithms to solve the UPMS with dynamic job release and dynamic machine availability. The objective of their solution methods is to minimize the total weighted tardiness. They recommended using search algorithms with short-term memory and fixed tabu list size for solving small size problems. They also recommended using search algorithms with long-term memory with fixed tabu list size for solving medium size problems and to use variable tabu list size for large instances. Rocha et al. (2008) developed a branch-and-bound (B&B) algorithm to solve this scheduling problem where setup times are machine dependent. They used the greedy randomized adaptive search procedure (GRASP) metaheuristic to establish an upper bound on the objective function value and used this upper bound within their B&B algorithm. They compared their results to those obtained by using the MIP commercial code CPLEX 9.0 and solving two integer programming formulations (Fanjul-Peyro, 2020).

Tavakkoli-Moghaddam et al. (2009) addressed the bi-objective UPMS with precedence constraints and proposed a genetic algorithm to solve it. The objective is to minimize both the number of tardy jobs and the make span. Ko et al. (2010) presented a new dispatching rule to solve the UPMS that selects the machine to use for each job in a way that guarantees a minimum required job quality. Unlu et al. (2010) proposed four different mixed integer programming formulations for the UPMS and compared the performance of these formulations. Kramer et al. (2021) developed five integer programming formulations of the UPMS and, using 1300 test instances, compared their performance concluding that two of them are more efficient.

Recently, Gokhale and Mathirajan (2012) considered the identical parallel machine problem with eligibility restrictions (a job can be processed by some but not all machines), sequence dependent setup times and release dates with the objective of minimizing the total weighted flow time. Wang et al. (2013) considered the unrelated machines version of this problem and proposed a mixed integer programming model and some local search heuristics to solve the problem. Finally, Afzalirad and Rezaeian (2016) addressed the unrelated parallel machine scheduling problem with sequence dependent setup time, limited additional resources (like tools and workers), job release dates, precedence constraints, and machine eligibility restrictions with the objective of minimizing the make span. In all the above-mentioned publications, each job requires only one machine.

3. Mathematical Formulation

It was observed that extruders and dies are bottleneck pieces of equipment in this industrial setting. Cooling baths, printers, haul-off machines, and other equipment are available in sufficient numbers. Consequently, in the following mathematical formulation, only the availability of extruders and dies is taken into consideration. Thus, the problem considered here is that of scheduling several jobs using the available extruders and dies. Each job can be executed on some but not all extruders, has a due date and a priority index(weight), and its processing time depends on the speed of the extruder to which it is assigned. The time to set up the extrusion line is necessary before starting the processing of a new job. This setup time depends on the preceding job on the extruder or extruders to be used. The objective is to minimize the total weighted jobs' tardiness. To formulate this problem, we introduce, for each job which requires a co-extruder, an associated fictitious job that should be processed (started and completed) at the same time as the original one. We require that the original job be executed by one of the possible primary extruders of the job and the fictitious job by one of the possible co-extruders. The setup time (the time to install the extrusion line) for the original job is the sum of times required for all the setup operations including the setup time of the co-extruder die while the set-up time for the fictitious one is the time to install the co-extruder die. The following notation is used:

Sets and indexes

N	number of jobs, indexed i or j
I	the set of all original jobs to schedule
I'	the set of jobs requiring a co-extruder; $I' \subseteq I$
J	the set of fictitious jobs; $\|J\| = \|I'\|$
D	the set of dies, indexed d
K	the set of all extruders, indexed k
K_i	the set of extruders that can process job i.
K_{ij}	the set of extruders that can process jobs i and j
N_d	the set of jobs (original and fictitious) that require die d
I_k	the set of jobs (original and fictitious) that can be processed by extruder k

Parameters

l_i	the due date of job i
w_i	a tardiness weight associated with job i
p_{ik}	processing time of job i if executed by extruder k
j_i	fictitious job associated to job $i \in I'$
s	time required to change a die
s_{ij}	changeover time before job j if it follows job i on the same extruder

s_{0j} setup time before job j if it is the first job to schedule on its extruder.
u_k upper bound on the number of jobs that can be assigned to extruder k; $u_k \leq |I_k|$
M Large number

Decision variables

x_{ikp} a binary taking the value 1 if job i is scheduled on extruder k in position p
y_{ij} a binary taking the value 1 if job i is scheduled to finish before job j
z_{ij} a variable taking the value 1 if jobs i and j are assigned to the same extruder and requires the same die, and 0 otherwise. Notice that z_{ij} takes the value 0 if $K_{ij}=\emptyset$
c_i completion (finish) time of job i
r_i tardiness of job i

Using the above notation, the problem can be modeled by the following mixed integer linear model:

$$determine: x_{ikp} \in \{0,1\}; i \in I \cup J; k \in K_i; p = 1,..,u_k,$$

$$r_i \geq 0; i \in I\ , c_i \geq 0\ ; i \in I \cup J\ and$$

$$y_{ij} \in \{0,1\}, z_{ij} \geq \in \{0,1\}0; i,j \in N_d, d \in D, j \neq i;\ which$$

$$min: \sum_{i \in I} w_i r_i$$

subject to:

$$r_i \geq c_i - l_i \quad ; \forall i \in I \tag{1}$$

$$c_i \leq c_j - p_{jk} - s_{ij} + M\left(2 - x_{jkp} - x_{ik,p-1}\right) \quad ; \forall i \in I \cup J, \forall k \in K_i, \forall\, j \in I_k, j \neq i, p > 1 \tag{2}$$

$$\sum_{k \in K_i} \sum_{p \leq u_k} x_{ikp} = 1 \quad ; \forall i \in I \cup J \tag{3}$$

$$\sum_{i \in I_k} x_{ikp} \leq 1 \quad ; \forall k \in K, p \leq u_k \tag{4}$$

$$\sum_{i \in I_k} x_{ikp} \leq \sum_{i \in I_k} x_{ikp-1} \quad ; \forall k \in K, 2 \leq p \leq u_k \tag{5}$$

$$c_i \geq (p_{ik} + s_{0i}) x_{ik1} \quad ; \forall i \in I, k \in K_i \tag{6}$$

$$c_i = c_{j_i} \quad ; \forall i \in I' \tag{7}$$

$$z_{ij} \geq \{\sum_{p \leq u_k} (x_{ikp} - x_{jkp})\} - 1 \quad ; \forall i,j \in N_d, i \neq j, d \in D, k \in K_{ij} if\ K_{ij} \neq \emptyset \tag{8}$$

$$c_i \leq c_j - \sum_{k \in K_j} \sum_{p \leq u_k} p_{jk} x_{jpk} - s\left(1 - z_{ij}\right) + M\left(1 - y_{ij}\right) \quad ; \forall i,j \in N_d, i \neq j, d \in D \tag{9}$$

$$y_{ij} + y_{ji} = 1 \quad ; \forall i,j \in N_d, i \neq j, d \in D \tag{10}$$

The objective function to minimize is the weighted sum of job tardiness. Constraint (1) defines the tardiness variable r_i and constraint (2) is to make sure that if job i is in position p-1 and job j is in position p on extruder k, then job i should be completed before we start to setup extruder k to process job j. Constraint (3) ensures that each job will be assigned to only one position on only one extruder. Constraint (4) implies that extruders will process at most one job at a time. Constraint (5) ensures that if no job is assigned to position p-1 on extruder k, then no job is assigned to position p on the same extruder. Constraint (6) implies that if a job is assigned to the first position on an extruder, then its completion time is larger than or equal to its processing time plus the required setup time and constraint (7) is to make sure that both the fictitious job and its associated job are completed at the same time. Constraint (8) assigns z_{ij} the value 1 if jobs i and j are assigned to the same extruder and 0 otherwise. Constraint (9) assures that if jobs i and j use the same die but are not assigned to the same extruder, then they are not executed at the same time while constraint (10) states that if jobs i and j use the same die, then either i should precede j or j should precede i.

In the studied industrial setting, we need to schedule from 25 to 35 jobs per week using 11 extruders and about 20 dies. So, to use the above presented MIP model we need to use 8125 binary variables and 50 continuous variables. Obviously, with the available commercial MIP solvers we are not able to solve problem instances of this size. As it will be shown later, we were able to solve instances with no more than 15 jobs.

4. Numerical Illustration

As illustration consider the following 8-job instance of the problem. The jobs to execute in this example are given in Table 1. This table gives the color, the material, the shape, the secondary operations to perform, the quantity to produce, due date (in hours) and tardiness weight assigned to each job. It also indicates whether one or two extruders are required. Table 2

gives for each job the extruders that can be used and the corresponding production rate (units/hour). The processing time p_{ik} of job i on extruder k can then be calculated by dividing the number of units required for job i by the production rate of extruder k.

Table 1 allows us to calculate the job changeover time between all pairs of jobs if executed on the same extruder. As mentioned above, there is only one person to do the setup operations required for a given job changeover. So, the set-up time is the sum of times required to do all setup operations (die change, color change, etc.) including the co-extruder die change if the job requires a co-extruder.

Table 1
Main data for the 8-job example

Job number	Co-extruder needed	Die numbers	Color	Product shape	Material	Secondary operation	Units to produce	Due date (hours)	Priority weight
1	no	1	10	21	30	41	1200	8	5
2	no	1	11	21	31	41	1500	16	1
3	yes	2 and 5	10	20	30	40	1500	8	1
4	yes	2 and 5	10	21	30	41	1800	16	5
5	no	3	11	22	31	41	1100	24	5
6	no	3	10	20	30	41	1050	8	5
7	no	4	11	21	30	40	1300	8	1
8	no	4	11	20	30	41	1400	16	1

Table 2
Production rate and co-extruders for the 8-job example

Job Number	Production rate for primary extruders				Processing time (in hours)				Co- extruder
	1	2	3	4	1	2	3	4	
1	120	150	-	-	10	8	-	-	-
2	110	140	-	-	13.64	10.71	-	-	-
3	-	150	-	-	-	10	-	-	3 or 4
4	-	140	-	-	-	12.86	-	-	3 or 4
5	-	-	100	100	-	-	11	11	-
6	-	-	110	80	-	-	9.55	13.13	-
7	-	130	90	120	-	10	14.44	10.83	-
8	140	-	120	90	10	-	11.67	15.56	-

Table 3 shows the setup operations and the changeover time for all job pairs i,j that can be performed on a same extruder while Table 4 presents the changeover time matrix for all these pairs. In this table NA (not applicable) indicates that we do not calculate this changeover time as the corresponding jobs cannot be done on the same extruder. The optimal solution of this example (as obtained by solving its mathematical model) is to use the production sequence 6-4-1-8-5-3-7-2 and assign each job to the extruder that can finish it as soon as possible. This solution is given in Table 5 and Figure 3. Its total weighted tardiness is 106.86.

Table 3
Setup operations and job changeover time for job pairs

Pair of jobs i,j	1,2	1,3	1,4	1,7	1,8	2,3	2,4	2,7	2,8	3,4	3,5	3,6
Die change (45 minutes)	-	✓	✓	✓	✓	✓	✓	✓	✓	-	✓	✓
Co-extruder die change (45 minutes)	-	✓	✓	-	-	✓	✓	-	-	-	-	-
Color change (30 minutes)	-	-	-	✓	✓			-	-	-	-	-
Shape option change (15 minutes)	-	✓	-	-	✓	✓	-	-	✓	✓	✓	-
Raw material change (120 minutes)	✓	-	-	-	-	✓	✓	✓	✓	-	✓	-
Secondary operation setup (15 minutes)	-	✓	-	✓	-	✓	-	✓	-	✓	✓	✓
Setup time (in minutes)	120	120	90	90	90	240	210	180	180	30	195	60

Table 3
Setup operations and job changeover time for job pairs (Continued)

Pair of jobs i,j	3,7	3,8	4,5	4,6	4,7	4,8	5,6	5,7	5,8	6,7	6,8	7,8
Die change (45 minutes)	✓	✓	✓	✓	✓	✓	-	✓	✓	✓	✓	-
Co-extruder die change (45 minutes)	-	-	-	-	-	-	-	-	-	-	-	-
Color change (30 minutes)	✓	✓	-	-	✓	✓	-	-	-	✓	✓	-
Shape option change (15 minutes)	✓	-	✓	✓	-	✓	✓	✓	✓	✓	-	✓
Raw material change (120 minutes)	-	-	✓	-	-	-	✓	✓	✓	-	-	-
Secondary operation setup (15 minutes)	-	✓	-	-	✓	-	-	✓	-	✓	-	✓
Setup time (in minutes)	90	90	180	60	90	90	135	195	180	105	75	30

Table 4
Sequence dependent setup changeover matrix (minutes)

Job	1	2	3	4	5	6	7	8
1	-	120	120	90	*NA*	*NA*	90	90
2	120	-	240	210	*NA*	*NA*	180	180
3	120	240	-	30	195	60	90	90
4	90	210	30	-	180	60	90	90
5	*NA*	*NA*	195	180	-	135	195	180
6	*NA*	*NA*	60	60	135	-	105	75
7	90	180	90	90	195	105	-	30
8	90	180	90	90	180	75	30	-

Table 5
The optimal schedule for the optimal sequence 6-4-1-8-5-3-7-2

Job	Due date	Priority weight	Die number	Assigned extruder	Starting time	Changeover time (hrs)	Processing time (hrs)	Finishing time	Tardiness	Weighted tardiness
6	8	5	3	3	0	1.25	9.55	10.80	2.80	14.00
4	16	5	2 and 5	2 and 4	0	2.00	12.86	14.86	0	0
1	8	5	1	1	0	1.25	10	11.25	3.25	16.25
8	16	1	4	1	11.25	1.50	10	22.75	6.75	6.75
5	24	5	3	3	10.80	2.25	11	24.05	0.05	0.25
3	8	1	2 and 5	2 and 4	14.86	0.50	10	25.36	17.37	17.36
7	8	1	4	2	25.36	1.50	10	36.86	28.86	28.86
2	16	1	1	1	22.75	3.00	13.64	39.39	23.39	23.39
								Total weighted tardiness		106.86

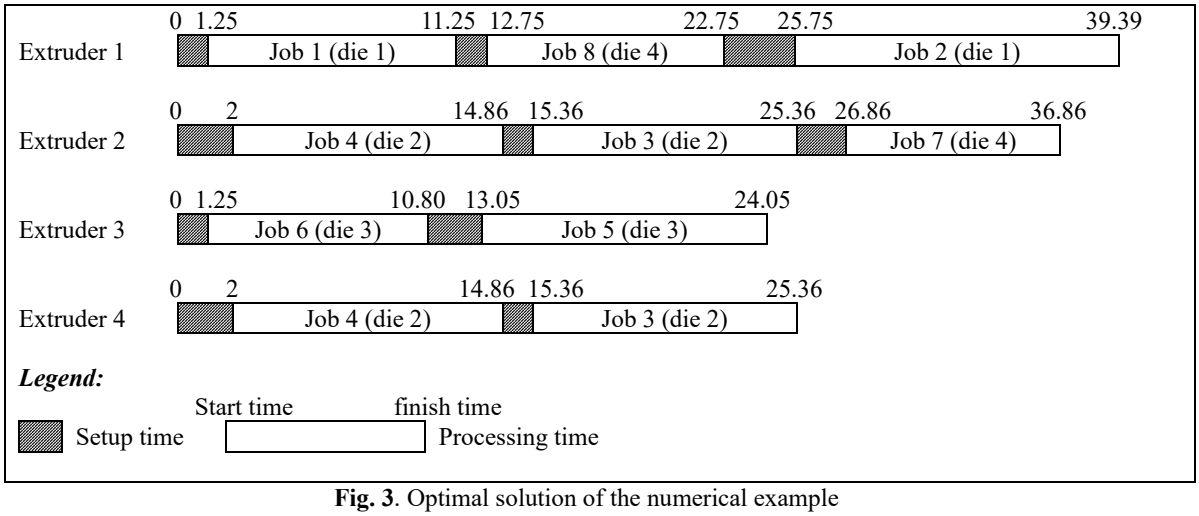

Fig. 3. Optimal solution of the numerical example

5. The Proposed Heuristics And Metaheuristics

First, let us mention that within our heuristic solution methods, we do not add fictitious jobs to those requiring two extruders. Such jobs will be scheduled simultaneously on the primary extruder and the co-extruder. Heuristic solution methods proposed in this paper can be divided into two groups: construction heuristics and neighborhood search heuristics or metaheuristics. Also, construction heuristics are divided into two sub-groups: onward construction heuristics and sequence-based construction heuristics. In onward construction heuristics, we use a decision criterion (for example the earliest possible finish time of non-scheduled jobs) to choose and schedule the first job and then update the values of this criterion and choose the second one and so on. In sequence-based heuristics we sequence jobs, based on one of their characteristics, and then schedule them as early as possible starting from the first one in the obtained sequence up to the last one. On the other hand, metaheuristics tested in this paper are an adaptation of the simulated annealing algorithm (SA), a genetic algorithm (GA), a tabu search method (TS) and a greedy randomized adaptive search procedure (GRASP). Although, metaheuristics are usually seen as efficient solution methods, it has been decided to try four of them to assess their performance with respect to the studied problem. Construction heuristics were mainly used to construct good initial solutions to be used by the proposed metaheuristics.

5.1 Onward construction heuristics

The steps of the onward construction heuristic to be used in this research can be summarized as follows:

1- For each non-scheduled job, calculate the value of its weighted tardiness if it is scheduled to start as early as

possible.
2- Choose and schedule the job having the maximum weighted tardiness.
3- If all jobs are scheduled, stop. Otherwise update the values of the weighted tardiness for each of the remaining jobs and go back to step 2.

5.2 Sequence-based construction heuristics

The steps of sequence-based construction heuristics we use in our study are:

1- Arrange all jobs according to a given sequencing rule (e.g., arrange jobs in the ascending order of their due date);
2- Starting with the first job in the sequence and based on the availability of extruders and dies, schedule each job in the sequence obtained in step 1, and assign to it the extruder (and the co-extruder, if any) that allow to finish it as early as possible. In case of tie, i.e., if there is more than one extruder (or combination of extruders) that are available and can finish the job at the same earliest time, choose the extruder k (or the extruder and co-extruder) having the smallest value of $|I_k|$ (the number of jobs that can be processed by extruder k).

Four sequencing rules are used within this study, but many others can be used. Within these four rules we use the job number to break ties. The first rule is the earliest due date (EDD) rule. The second, called the earliest weighted due date (EWDD), in which jobs are indexed in the ascending order of d_i/w_i. The third rule arranges jobs in the ascending order of $(d_i - \max_{k\in K_i} p_{i,k})$; this rule will be called earliest reduced due date (ERDD). The fourth rule, called the earliest weighted reduced due date (EWRDD), arranges jobs in the ascending order of e_i where:

$$e_i = \frac{d_i - \max_{k\in K_i} p_{i,k}}{w_i} \text{ if } \left(d_i - \max_{k\in K_i} p_{i,k}\right) \geq 0; \text{ and } e_i = w_i(d_i - \max_{k\in K_i} p_{i,k}) \text{ if } \left(d_i - \max_{k\in K_i} p_{i,k}\right) < 0.$$

5.3 The Proposed Metaheuristics

As mentioned above, adaptations of the simulated annealing algorithm (SA), the genetic algorithm (GA), the tabu search (TS) heuristic and the greedy randomized adaptive search procedure (GRASP) are designed to solve the problem. In the following we present the details of these proposed adaptations but first let us discuss two common aspects of these metaheuristics.

Solution coding

Within all these heuristics, a solution is coded by a sequence or an ordered list of all jobs to execute. The corresponding schedule can be constructed as in the second step of the sequence-based construction heuristic presented in section 5.2.

Neighborhood definitions

Five neighborhood definitions are used within the four tested metaheuristics. The first (N1) is a job permutation neighborhood where a randomly selected job swaps its position with its succeeding job in the sequence. In the second (N2), we randomly generate two integers p and q where p is between 1 and n-4, and q is between p+1 and p+4. Then we exchange the job in position p with the one in position q. The third neighborhood (N3) is defined by all neighbors where a randomly selected job is moved forward x positions where x is also a uniformly distributed number between 1 and a preselected value, say x_1. In our experimental study we used x_1=4. If the new position is larger than n then we reduce it to n. The fourth neighborhood (N4) contains all solutions where we randomly select two positions $p \leq n$ and $q \leq n$ and swap the jobs in these two positions. Finally, the fifth one (N5) is to move a randomly selected job from its position to any randomly selected different positions.

5.3.1 Adaptation of the simulated annealing algorithm (SA)

The proposed simulated annealing algorithm is composed of several reheating (annealing) cycles, several cooling steps within each annealing cycle, and uses a variable cooling rate (for more details see Boctor 1996). The main steps of this adaptation are exhibited in Fig. 4. To use this adaptation, we need to choose the value of several parameters. These parameters are the initial temperature *TMAX*, the number of reheating cycles *HMAX*, the number of cooling steps *CMAX*, the temperature reduction coefficient α, and N_c, the number of repetitions at each cooling step c. In our numerical study reported in section 6, we used *HMAX* =10, *CMAX* =4, *TMAX* =32, α=0.25 and N_c =100, 75, 75 and 250 respectively for the 4 cooling steps.

5.3.2 Adaptation of greedy randomized adaptive search procedure (GRASP)

The greedy randomized adaptive search procedure (GRASP) generates several production sequences and their corresponding production schedules, improves each schedule and retains the best one. Figure 5 gives the main procedure and the subprocedures of the proposed GRASP. Let PEN_{ik} be a penalty (the weighted tardiness) of scheduling job i to start at t on

extruder $k \in K_i$. A sequence is obtained by the following construction heuristic. Start at t=0, the beginning of the planning horizon, and calculate PEN_{ik} for each job i and each possible extruder $k \in K_i$. Calculate $PEN_i = min_k \{PEN_{ik}\}$, build a restricted candidate list (RCL) including all jobs l such that:

$$PEN_l \leq \min_i PEN_i + \beta\left(\max_i PEN_i - \min_i PEN_i\right),$$

and randomly choose among these jobs the one to schedule. Then, move to the time moment where there is at least one job that can start and use the same method to choose the next job, and so on.

```
CALL initial (generates an initial solution by applying the sequence-based construction heuristic using the EWRDD sequencing rule)
CALL current (store the solution obtained as the current solution)
CALL best (store the current solution as the best found so far)
h := 0 (initialize heating cycle counter)
REPEAT until h = HMAX
    T := TMAX (initialize the cooling temperature)
    c := 0 (initialize the cooling step counter)
    REPEAT until c = CMAX
        r := 0 (initialize the number of repetitions counter)
        R := Nc
        REPEAT until r = R
            CALL neighbor (randomly generate a neighbor solution)
            d := Objective function (neighbor) – Objective function (current)
            IF d < 0 OR random (0, 1) < e^(-d/T) THEN
                CALL current (store as current solution)
                IF Objective function (current) < Objective function (best) THEN
                    CALL best (store as best solution found so far)
                END IF
            END IF
        END REPEAT
        T : = αT (reduce cooling temperature)
    END REPEAT
END REPEAT
```

Fig. 4. The developed Simulated Annealing algorithm

```
GRASP main procedure
    REPEAT G times
        Build a production sequence σ and the corresponding production schedule (using the SCHEDULE BUILDING procedure presented
        hereafter)
        Improve the built sequence until getting a local optimum σLocal (SEQUENCE IMPROVEMENT procedure)
        IF the cost of σLocal is smaller than the cost of the best-found sequence σbest, THEN σbest= σLocal
    END REPEAT
    RETURN σbest

SCHEDULE BUILDING procedure
    REPEAT until all jobs are scheduled
        Go to the nearest date t where there is at least one job that can be started
        Establish the list of candidate jobs that can start at time t
        Calculate the penalty PENi = mink {PENik} for each of these jobs
        Determine PENmin and PENmax, the minimum and maximum penalty over all candidates
        Randomly choose a value for β from a uniform distribution between 0 and 1
        Put in the restricted candidate list (RCL) all jobs i such that PENi≤PENmin+β(PENmax -PENmin)
        Randomly choose and schedule a job from the RCL
    END REPEAT
    Calculate the total weighted tardiness of the obtained schedule

SEQUENCE IMPROVEMENT procedure
        WHILE improvement=TRUE
            improvement=FALSE
            FOR j =1 to number of jobs
                FOR all positions p
                    Calculate Djp, the penalty change if j is moved to position p in the sequence σ
                NEXT p
                IF Djq =minp {Djp}<0 THEN
                    Improvement=TRUE: Move job j to the corresponding position q
                END IF
            NEXT j
        WHILE END
```

Fig. 5. The developed GRASP

As indicated by Resende and Ribeiro (2008), the value to attribute to β influences the quality of the produced solution. They indicated that choosing β at random from a uniform probability distribution, leads to good results. This method is used in

our GRASP implementation. Another important parameter is G, the number of iterations to use. In our experimental study, several values of G are tested.

5.3.3 Adaptation of the genetic algorithm (GA)

Within our adaptation of the genetic algorithm, each solution is coded by a production sequence. The initial population is composed of N_s solutions generated by the sequence building procedure used by the GRASP presented above. Crossover of parent individuals is done by the random key crossover mechanism developed by Bean (1994) where a random number (called random key) between 0 and 1 is associated to each job such that if job i precedes job j in the sequence, then the value of the random key of job i should be less than the value of the key of job j. A cross-over position is then randomly chosen, and the first offspring is obtained by concatenating the keys of the first part of the first parent keys vector with the keys of the second part of the second parent vector. Similarly, the second offspring is obtained by concatenating the keys of the first part of the second parent vector and the keys of the second part of the first parent vector. The production sequence of each offspring is then determined by arranging the jobs in the ascending order of the resulting keys vector. Figure 6 presents the framework of the proposed GA and Figure 7 depicts an example of the used crossover mechanism where the first parent solution has the job sequence 4-2-5-3-6-1 and the second parent has the sequence 2-4-3-5-1-6. A crossover at position 2 produces the offspring sequences: 2-4-3-5-6-1 and 4-2-5-3-1-6. The mating pool is composed of the best 80% solutions of the current generation and a second copy of the best 20% solutions. Mutation is done randomly where only one job is moved to a randomly chosen new position in the sequence. This mutation procedure is applied to M solutions of the current generation. The genetic algorithm stops after W generations (including the initial generation). In our implementation of the GA, we used N_s=200 and M=10. We tested two values of W (W=10 and W=25) which makes the total number of generated solution equals 200W (i.e., 2000 and 5000 generated solutions).

```
Use the sequence and scheduling building procedure presented in Figure 5 to generate Ns solutions to use as initial population (generation)
Let S be the best solution in this initial generation
Set: w = 1
WHILE w < W
    Arrange the solutions of the current population in the ascending order of their weighted tardiness value
    Replace the 20% worst solutions in the current generation by the best 20% solutions
    FOR n =1 TO N/2
        Randomly select two different solutions from the current population
        Use the random key crossover operator to crossover the codes of the selected 2 solutions and generate two offspring solution codes
        Calculate the total weighted tardiness of each offspring solution.
    NEXT n
    For m=1 to M
        Randomly select a solution among those of the new generation
        Randomly select a job in the sequence of selected solution
        Move the selected job to another randomly selected position in the sequence
        Calculate the total weighted tardiness of muted solution
    NEXT m
    Let S' be the best solution in the new generation
    IF S' is better than S THEN put S=S'
    Replace the current population by the new generation
    Set: w=w+1|
END WHILE
```

Fig. 6. The developed Genetic Algorithm (GA)

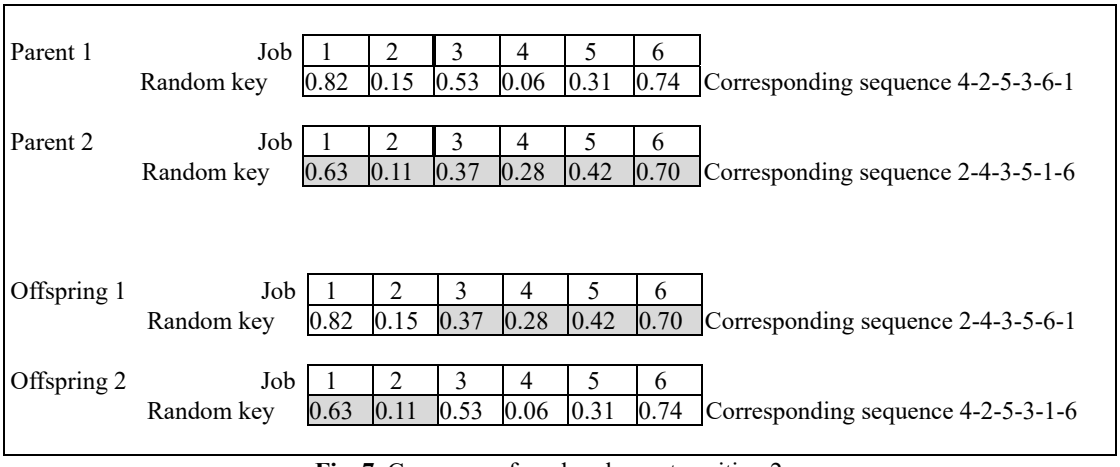

Fig. 7. Crossover of random keys at position 2

5.3.4 Adaptation of the tabu search heuristic (TS)

Fig. 8 presents the framework of the proposed adaptation of the tabu heuristic. The initial solution and the neighborhood definitions of this implementation of the tabu search heuristic are the same as those of our simulated annealing algorithm. The tabu list is composed of variable number of records (between 5 and 10) and each record is composed of some selected characteristics (called attributes) of the solutions considered as tabu. The attributes to put in a record differ depending on the used neighborhood definition. Table 6 gives the five used neighborhoods and the corresponding attributes we put in the tabu list. For example, if the neighborhood contains all the neighbor solutions where only one job is swapped with the one in the next position of the execution sequence, we put in the tabu list the number of the moved job and during a number of iterations, we consider that moving again this job is tabu. The usual aspiration criterion is also used. Thus, if the search procedure reaches a solution considered as tabu solution (has the same attribute or attributes as one of the records in the tabu list) but is better (has a lower value of the total weighted tardiness) than the best-found solution, then the tabu nature of this solution is revoked, and the solution is accepted as the next current solution. The search stops once a predetermined number of solutions are visited. In our evaluation experiment reported in section 6, we fixed this number to either 2000 or 5000.

Tabu Search main procedure
Use the sequence-based construction heuristic to generate an initial solution S_i
Put the current solution $S_c = S_i$
Put the best solution $S_{best} = S_i$
Set the number of visited solutions counter k=0
Set the length (number of records) of the tabu list L_{tabu}=0
tabu list = empty

REPEAT until $k = K_{max}$
    k=k+1
    UPDATE Current and Best solutions
    UPDATE Tabu List
END REPEAT
RETURN S_{best}

UPDATE Current and Best solutions
Find S_b the best solution in the neighbors N(S_c) of the current solution S_c
Indicator = false
While Indicator=false
    IF S_b is tabu (has the same attributes as those of one of the records in the tabu list) THEN
        IF S_b is better than S_{best} THEN
            Indicator=true
        ELSE
            Remove S_b from N(S_c)
            Find S_b be the best solution in the remaining neighborhood
        ENF IF
    ELSE
        Indictor=true
    END IF
END WHILE
Put the current solution $S_c = S_b$
IF S_c is better than S_{best} THEN $S_{best} = S_c$

UPDATE tabu List
Let L=Random (5,10)
IF $L < L_{tabu}$ THEN
    Remove the ($L_{\text{tabu}} - L + 1$) most ancient records from the tabu list
    $L_{\text{tabu}} = L$
END IF
Add to the tabu list a record containing the attributes (selected characteristics) of S_c

Fig. 8. The developed tabu search heuristic

Table 6
Attributes of the tabu solutions for the five used neighborhoods

	Neighborhood definition	Attributes
N1	Swap a randomly selected job with the succeeding one.	The number of the moved job
N2	Randomly select two positions p and q such that: $1 \leq p \leq n-4$ and $p+1 \leq q \leq p+4$. Swap jobs in positions p and q.	The number of the swapped jobs
N3	Randomly select a job, say job in position p, and an integer x between 1 and 4. Move the job in position p to position $p+x$.	The numbers of the moved job
N4	Swap the positions of two randomly selected jobs	The numbers of the swapped jobs
N5	Move a randomly selected job to a randomly selected different position	The number of the moved job

6. Heuristics And Metaheuristics Performance

To evaluate the performance of the developed solution methods we first randomly generated one hundred instances of 15 jobs each. These instances were solved to optimality using the MIP commercial code GUROBI. They were also solved by all the proposed heuristics and metaheuristics. Heuristics and metaheuristics were coded in Python and solved using a computer equipped with an intel core 2 Quad and 4 Gb RAM. We only used instances of this size as the MIP code was not able to obtain the optimal solution for instances with more than 15 jobs. Recall that real-life instances observed in our industrial setting involve between 25 and 35 jobs.

6.1 Test instances generation

Test instances were randomly generated, and their structure and parameters are similar to those observed in the real industrial setting. So, all these instances involve 11 different extruders and 15 jobs to execute. For each job we draw a random number x from a uniform distribution between 0 and 1. Then, if the $x < 0.25$ we consider that the job requires two extruders. The number of extruders that can process any given job is between 1 and 3 and the required extruders are randomly selected among the 11 available extruders. The die or dies to use are randomly selected from the available 20 dies. The color is also randomly selected from the 60 possible colors where the probability of selecting one of the five most frequent colors is 75%. Then the shape, the material, and the secondary operation are randomly selected among the 7 possible shapes, the 17 materials and the 5 possible secondary operations. The quantity to produce is randomly drawn between 1200 and 3000 units and the production rate between 120 and 250 units per hour. The due date is an integer multiple m of 8 hours where m is randomly chosen between 1 and 5. Finally, the priority weight is 1, 2 or 3.

6.2 Optimal solutions

All instances were solved using the commercial MIP code GUROBI. The average CPU time to solve the mathematical models of the generated instances was 1879.37 seconds with a standard deviation of 382.13. The maximal run time was 2364.62 seconds, and the minimum time was 0.67 seconds. The average total weighted tardiness is 78.91.

6.3 Performance of the proposed construction heuristics

Table 7 summarizes the results obtained by the proposed construction heuristics. The priority-based construction heuristic was tested using four different priority rules: EDD, EWDD, ERDD and EWRDD. Also, the onward construction heuristic is tested where at each step we select the job having the largest weighted tardiness. This table shows clearly that simple construction heuristics cannot produce sufficiently good solutions. The best one of these construction heuristics, the EWDD, gave an average deviation from the optimum of 57.7%. It also produced the best solution (among those obtained by the construction heuristics) for 45 of the 100 test instances. As it will be shown in the following, metaheuristics produce much better solutions.

Table 7
Performance of the proposed construction heuristics

Performance indicator	Priority-based heuristic with				Onward
	EDD	EWDD	ERDD	EWRDD	heuristic
Average total weighted tardiness	133.9	115.7	137.4	124.0	157.5
Average percentage deviation from the optimum	77.0	57.7	86.5	67.3	129.2
Standard deviation (%)	54.6	56.5	69.8	53.0	146.9
Maximum percentage deviation from the optimum	289.9	423.8	378.7	297.0	989.6
Minimum percentage deviation from the optimum	0	0	0	0	0
Number of times the heuristic found the best solution	21	45	17	25	12
Number of times the heuristic found the optimal solution	1	1	2	2	1

6.4 Performance of the proposed metaheuristics

Table 8 gives the results obtained by the four tested metaheuristics: the simulated annealing algorithm (SA), the tabu search (TS) heuristic, the genetic algorithm (GA) and the greedy randomized adaptive search procedure (GRASP). As mentioned

before, for the first two heuristics we tested five neighborhood definitions, N1 to N5 (presented in Table 6). This table gives the average percentage deviation from the optimum for each tested heuristic, the minimum and maximum percentage deviation from the optimum, the standard deviation of the percentage deviation, the number of times the heuristic obtained the optimum solution, the number of times it obtained the best solution among those obtained by all tested heuristics, and the average computational time in seconds.

The best average percentage deviation from the optimum was obtained by the simulated annealing algorithm N5 (1.95 %) and by the tabu search heuristic using the neighborhood N4 (1.97%). Good results were also obtained by the SA with N3 and N4 as well as by the TS heuristic with N5. It is worth noting that these two of the tested metaheuristics allowed obtaining the optimal solutions for 30 0f the 100 test instances. We can expect to even improve these results if we use larger neighborhoods within these metaheuristics. However, using larger neighborhoods may significantly increase computation times.

The results given in table 8 were obtained while limiting the number of visited (evaluated) solutions to 2000 for all the tested heuristics. For the SA and TS heuristics, the number of examined solutions is the number examined neighbors or equivalently, the number of iterations. For the GA, the number of examined solutions is the number of generated individuals (initial population and generated offspring solutions) while for the GRASP it is the number of generated solutions. We also tested these methods with a limit of 5000 examined solutions; these results are given in table 9. The results obtained with this larger number of visited solutions were very close to those obtained with 2000 examined solutions. For example, the average percentage deviation from the optimum obtained by the SA with N5 was reduced from 1.95% to 1.92% when the number of examined solution was increased from 2000 to 5000.

In general, increasing the number of examined or visited solutions should improve the obtained results. However, because of the random nature of these neighborhood search techniques, sometimes we may obtain solutions of lower quality. Comparing tables 8 and 9, we can see that most heuristics performance indicators were improved by increasing the number of examined solutions. However, some of them deteriorated. Such indicators are given in boldface characters in table 9. For example, the number of optimal solutions obtained by the tabu search method when using the neighborhood definition N4 decreases from 30 to 28 when the number of examined solutions goes from 2000 to 5000.

Table 8
Performance of the tested metaheuristics (with 2000 visited neighbor solutions)

Performance indicator	Simulated Annealing (SA) with					Tabu Search (TS) with					GA	GRASP
	N1	N2	N3	N4	N5	N1	N2	N3	N4	N5		
Average total weighted tardiness	82.91	80.64	80.47	80.53	80.47	87.97	81.10	80.67	80.59	80.54	80.98	81.00
Average percentage deviation from the optimum	6.53	2.51	2.12	2.13	1.95	11.69	3.10	2.58	1.97	2.13	2.55	2.71
Maximum percentage deviation from the optimum	47.20	14.80	12.10	12.10	12.10	63.23	20.25	15.53	12.10	12.10	13.56	15.18
Minimum percentage deviation from the optimum	0	0	0	0	0	0	0	0	0	0	0	0
Standard deviation of the average % deviation	8.41	2.80	2.25	2.25	2.23	11.59	3.77	2.83	2.30	2.25	2.75	3.15
Number of times the heuristic found the best solution	39	75	87	86	95	24	66	75	97	86	75	77
Number of times the heuristic found the optimal solution	13	26	26	26	30	8	20	25	30	26	25	24
Average computational time (seconds)	139.7	141.1	138.1	138.1	111.8	148.6	144.9	149.9	149.1	154.1	164.7	159.5

Table 9
Performance of the tested metaheuristics (with 5000 visited neighbor solutions)

Performance indicator	Simulated Annealing (SA) with					Tabu Search (TS) with					GA	GRASP
	N1	N2	N3	N4	N5	N1	N2	N3	N4	N5		
Average total weighted tardiness	82.87	80.41	80.36	80.45	80.44	**88.21**	80.98	80.44	**80.69**	80.48	**81.05**	80.99
Average percentage deviation from the optimum	6.37	2.22	1.97	2.03	1.92	**11.98**	2.95	2.26	**2.10**	2.05	**2.64**	2.69
Maximum percentage deviation from the optimum	47.2	12.1	12.1	12.1	12.1	**68.9**	**21.59**	12.1	12.1	12.1	**15.18**	15.18
Minimum percentage deviation from the optimum	0	0	0	0	0	0	0	0	0	0	0	0
Standard deviation of the average % deviation	8.29	2.52	**2.3**	2.25	2.23	**11.92**	3.64	2.47	**2.32**	**2.26**	**3.01**	**3.24**
Number of times the heuristic found the best solution	47	88	99	96	99	27	74	85	**93**	95	**74**	81
Number of times the heuristic found the optimal solution	16	28	30	28	30	8	23	27	**28**	28	25	24
Average computational time (seconds)	374.5	370.1	370.3	355.4	361.7	348.2	350.5	345.0	352.8	364.7	400.1	379.6

6.5 Heuristics comparison: the means test

To see if the difference between the average deviation from the optimum as obtained by any two different heuristics is statistically significant, we can use the means statistical test. Let μ_i and μ_j be the average deviation as obtained by two different heuristics *i* and *j* over the *n* test instances, where $\mu_i > \mu_j$; and let v_i and v_j are the corresponding variances of the solution values obtained by these two heuristics respectively. To test the null hypothesis $H_0: \mu_i = \mu_j$ against the alternative hypothesis $H_1: \mu_i > \mu_j$; we will use the means test. For this statistical test, the test criterion is:

$$T_{ij} = \frac{\mu_i - \mu_j}{\sqrt{(v_i + v_j)/n}}$$

and it follows the Student probability distribution. At a significant level α, the null hypothesis is rejected if $T_{ij} > t_{\alpha,2n-2}$ where $t_{\alpha,2n-2}$ is the critical value of the Student distribution.

Table 10 gives the values of T_{ij} for all pairs of heuristics such that the average deviation of the heuristic in row *i* is greater than the average deviation as obtained by the heuristic in column *j*. These T_{ij} values were calculated based on the solutions reported in Table 8. Knowing that the critical value of the Student distribution for a significance level $\alpha = 5\ \%$ and $n = 100$ equals 1.658, we can see that the null hypothesis $H_0: \mu_i = \mu_j$ is rejected for some pairs of heuristics. The boldface figures in Table 10 indicate the pairs where the null hypothesis is rejected. Mainly, the hypothesis is rejected when we compare the tabu search heuristic using the neighborhood definition N1 (TS-N1) to all other heuristics and when we compare the simulated annealing algorithm using N1 to all other heuristics except TS-N1. For pairs where the null hypothesis is accepted, we can consider the corresponding heuristics as statistically equivalent.

Table 10
Values of the test criterion T_{ij} (for the results with 2000 visited neighbor solutions) *

Heuristic *j*		Simulated Annealing (SA) with					Tabu Search (TS) with					GA	GRASP
Heuristic *i*		N1	N2	N3	N4	N5	N1	N2	N3	N4	N5		
Simulated Annealing (SA)	N1	X	**4.535**	**5.066**	**5.054**	**5.264**		**3.722**	**4.452**	**5.230**	**5.054**	**4.498**	**4,254**
	N2		X	1.086	1.058	1.564				1.490	1.058		
	N3			X		0.537				0.466			
	N4			0.031	X	0.807				0.696	0		
	N5					X							
Tabu Search (TS)	N1	**3.603**	**7.699**	**8.106**	**8.248**	**8.252**	X	**22.78**	**32.19**	**42.26**	**42.48**	**33.23**	**28,508**
	N2		1.256	**2.232**	**2.573**	**2.625**		X	1.103	**2.559**	**2.209**	1.179	0,794
	N3		0.176	1.272	1.590	**1.749**			X	**1.673**	1.245	0.076	
	N4					0.062				X			
	N5			0.031		0.568				0.497	X		
Genetic Algorithm (GA)			0.102	1.210	1.527	**1.695**				1.618	1.182	X	
GRASP			0.475	1.524	**1.841**	**1.969**		0.307		**1.897**	**1.498**	0.383	X

*Boldface figures in the table indicate the pairs where the null hypothesis is rejected.

7. Conclusions

To the best of our knowledge, this paper is the first one to address the problem of scheduling jobs in an extrusion facility. Its contribution is threefold. First it introduces the problem and develops a mathematical formulation to solve it. Unfortunately, this formulation only allows solving small instances of up to 15 jobs. Second, several construction heuristics and metaheuristics are developed. Third, a numerical experiment allowed us to evaluate the performance of the proposed heuristics and metaheuristics. This experiment shows that the proposed metaheuristics perform well, and we suggest using these heuristics to solve real-life instances of the problem.

It is worth noting that using some of the proposed metaheuristics allows to obtain solutions with less than 2% average deviation from the optimum. We may obtain even better solutions if we use larger neighborhoods. But enlarging neighborhoods should also lead to larger computational times. So, it is up to the decision maker to decide whether to use larger neighborhoods or not.

Based on the results of this research work, it seems necessary to develop more efficient heuristics to solve the problem. It is also necessary to develop a specially designed optimal solution method. Finally, we may try to solve other versions of the problem where we consider the availability of workers and other pieces of equipment.

Acknowledgement

This research work was partially supported by Grants OPG0036509 and OPG 0172633 from the Canadian Natural Sciences and Engineering Research Council (NSERC). This support is gratefully acknowledged.

References

Afzalirad, M., & Rezaeian J. (2016). Resource-constrained unrelated parallel machine scheduling problem with sequence dependent setup times, precedence constraints and machine eligibility restrictions. *Computers and Industrial Engineering*, *98*, 40-52.

Allahverdi A. (2015). The third comprehensive survey on scheduling problems with setup times/costs. *European Journal of Operational Research*, *246*(2), 345–378.

Bacalhau, J. B., & Cunha, T. M. (2017). Effect of NI content on the hardenability of bainitic steel for processing of plastics, 24th ABCM International Congress of Mechanical Engineering December 3-8, 2017, Curitiba, PR, Brazil.

Bean, J. (1994). Genetic algorithms and random keys for sequencing and optimization, *INFORMS Journal on Computing*, *6*(2), 154–160.

Bilge, Ü., Kiraç, F., Kurtulan, M., & Peküng P. (2004). A Tabu search algorithm for parallel machine total tardiness problem. *Computers and operations research*, *31*(3), 397-414.

Bilyk, A., & Mönch, L. (2012). A variable neighborhood search approach for planning and scheduling of jobs on unrelated parallel machines, *Journal of Intelligent Manufacturing*, *23*(10), 1621-1635.

Boctor, F.F. (1996). Resource-constrained project scheduling by simulated annealing, *International Journal of Production Research*, *34*(8), 2335-2351.

Dastidar, S.G. & Nagi, R. (2005). Scheduling injection molding operations with multiple resource constraints and sequence dependent setup times and costs, *Computers and Operations Research*, *32*(11), 2987-3005.

Fanjul-Peyro L. & Ruiz R. (2011) Size-reduction heuristics for the unrelated parallel machines scheduling problem. *Computers and Operations Research, 38*(1), 301-309.

Fanjul-Peyro L. (2020) Models and an exact method for the Unrelated Parallel Machine scheduling problem with setups and resources, *Expert systems with application*, 5, 1-15.

Framinan, J-M., Leisten R. & Ruiz Garcia R. (2014). *Manufacturing Scheduling Systems*. Springer.

Gokhale, R., & Mathirajan, M. (2012). Scheduling identical parallel machines with machine eligibility restrictions to minimize total weighted flow time in automobile gear manufacturing. *International journal of Advanced Manufacturing Technology*, *60*, 1190-110.

Ko, H.H., Kim, J., Kim, S.S., & Beak J.G. (2010). Dispatching rule for non-identical parallel machines with sequence dependent setups and quality restrictions, *Computers and Industrial Engineering*, *59*(3), 448-457.

Kramer, A., Lori, M., & Lacomme, Ph. (2021). Mathematical formulations for scheduling jobs on identical parallel machines with family setup times and total weighted completion time minimization, *European Journal of Operational Research, 289*(3), 464-474.

Lee, Y. H., & Pinedo, M. (1997). Scheduling jobs on parallel machines with sequence dependent setup times, *European Journal of Operational Research*, *100*(3), 464-474.

Lee, J. Y., Kim, Y. D., & Lee, T.E. (2018). Minimizing total tardiness on parallel machines subject to flexible maintenance, *International Journal of Industrial Engineering: Theory Applications and Practice*, *25*(4), 472- 489.

Logendran, R., McDonell, B., & Smucker, B. (2007). Scheduling unrelated parallel machines with sequence-dependent setups. *Computers and operations research*, *34*(11), 3420 – 3438.

Mönch, L., Balasubramanian, H., Fowler, J. W., & Pfund, M. E. (2005). Heuristic scheduling of jobs on parallel batch machines with incompatible job families and unequal ready times, *Computers and Operations Research*, *32*(11), 2731-2750.

Resende, M. G. C., & Ribeiro, C. C. (2008). Greedy Randomized adaptive search procedures: advances and applications, in Handbook of metaheuristics, Gendreau M. and Potvin J.Y. (eds.) Springer.

Rocha, P.L., Ravetti, M.G., Mateus, G.R. & Pardalos, P.M. (2008). Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine-dependent setup times. *Computers and Operations Research*, *35*(4), 1250 – 1264.

Rodriguez, F.J., Lozano, M., Blum, Ch., & Garcia-Martinez, C. (2013) An iterative greedy algorithm for the large-scale unrelated parallel machines scheduling problem. *Computers and Operations Research*, *40*(6), 1829-1841.

Tavakkoli-Moghaddam, R., Taheri, F., Bazzazi, M., Izbad, M. & Sassani, F. (2009). Design of a genetic algorithm for bi-objective unrelated parallel machines scheduling with sequence-dependent setup time and precedence constraints, *Computers and Operations Research*, *36*(12), 3224-3230.

Unlu, Y., & Mason, S.J. (2010). Evaluation of mixed integer programming formulations for non-pre-emptive parallel machine scheduling problems, *Computers and Industrial Engineering*, *58*(4), 785-800.

Wang, I.L., Wang, Y.C., & Chen, C.W. (2013). Scheduling unrelated parallel machines in semiconductors manufacturing by problem reduction and local search heuristics. *Flexible Services and Manufacturing Journal, 25*,343-366.

Wang, H., & Alidaee, B. (2019). Effective heuristic for large-scale unrelated parallel machines scheduling problem, *Omega*, *83*(3), 261-274.

Wisner, J.D., & Siferd, S.P. (1995). A survey of US manufacturing practices in make-to-order machine shops. *Production and Inventory Management Journal*, *36*(1), 1–7.