A new heuristic and an exact approach for a production planning problem

Peter Auer³ · György Dósa¹ · Tibor Dulai¹ · Armin Fügenschuh² · Peggy Näser² · Ronald Ortner³ · Ágnes Werner-Stark¹

Published online: 11 June 2020
© The Author(s) 2020

Abstract

We deal with a very complex and hard scheduling problem. Two types of products are processed by a heterogeneous resource set, where resources have different operating capabilities and setup times are considered. The processing of the products follows different workflows, allowing also assembly lines. The complexity of the problem arises from having a huge number of products from both types. The goal is to process all products in minimum time, i.e., the makespan is to be minimized. We consider a special case, where there are two job types on four different tasks, and four types of machines. Some of the machines are multi-purpose and some operations can be processed by different machine types. The processing time of an operation may depend also on the machine that processes it. The problem is very difficult to solve even in this special setting. Because of the complexity of the problem an exact solver would require too much running time. We propose a compound method where a heuristic is combined with an exact solver. Our proposed heuristic is composed of several phases applying different smart strategies. In order to reduce the computational complexity of the exact approach, we exploit the makespan determined by the heuristic as an upper bound for the time horizon, which has a direct influence on the instance size used in the exact approach. We demonstrate the efficiency of our combined method on multiple problem classes. With the help of the heuristic the exact solver is able to obtain an optimal solution in a much shorter amount of time.

Keywords Production planning · Mixed-integer programming · Heuristics · Simulation

✉ Tibor Dulai
dulai.tibor@virt.uni-pannon.hu

¹ University of Pannonia, Egyetem Str. 10., Veszprém 8200, Hungary

² Brandenburgische Technische Universität Cottbus-Senftenberg, Cottbus, Germany

³ Montanuniversität Leoben, Leoben, Austria

1 Introduction

We consider a production planning problem for a hypothetical factory that produces a number of different products. The production is carried out on various machines. Some production steps can be carried out in parallel in order to speed-up the production, where the machines may operate at different speeds. Other steps are carried out sequentially, and pre-manufactured products from previous production steps have to arrive simultaneously in order to carry out the current step. Some machines are flexible to play different roles in the process, but due to specialization, the machine can be slower or faster compared to others. Also set-up times need to be considered when changing the mode of operation. An operation schedule is sought that describes the precise time and operation for each machine over the whole time horizon. A way of comparing different plans is to focus on the makespan, that is, the time that is needed to produce a given number of products.

The focus of the paper is a quite special case. There are 2 job types on 4 different tasks, and 3 main types of machines. This problem cannot be considered as general, but as we will see, even in this special setting the problem is interesting and challenging to solve. At the moment it is open how it can be applied as a general technique suitable for a wide class of problems. The solution method we propose heavily exploits this structure. We will need to find a collection of tricks that works well together on this special problem.

This production problem describes the main features of a huge class of industrial-type automated manufacturing problems. In this work, we describe an instance of this problem, and give a mathematical formulation as a mixed-integer linear program, which turns out to be a multi-commodity flow problem on a time-expanded graph with precedence constraints. We analyze the capability of state-of-the-art mixed-integer solvers to deal with instances of this problem out-of-the-box. Such solvers produce a proven optimal solution, but due to the combinatorial difficulty of the problem, they tend to need much time for doing so—typically, the time is exponential in the input data. Furthermore, we describe a heuristic algorithm that creates production scheduling plans from scratch. Creating feasible plans is not difficult (from a computational point-of-view), since it is always possible to extend the production time. What is difficult is to find the minimum makespan production schedule. As a final step, we analyze to what extent the exact solution approach (i.e., a mixed-integer solver) benefits if it is given a feasible solution as initial value. Operationally, such starter solutions are used during the branch-and-bound solution process in order to prune the search tree. Thus having a good feasible solution at hand can usually save some time when solving a mixed-integer linear programming (MILP) problem. Using a test-set of instances we will analyze to what extent this is true for our production problem.

A general introduction to production planning with mixed-integer programming techniques can be found, e.g., in Pochet and Wolsey (2006) or Sawik (1999). Unfortunately, we cannot classify exactly our model into any type. We will give an explanation about this later.

Almeder et al. (2015) consider two extreme cases of batching (i.e., a successor item can only be produced if the full batches of the predecessor items are finished)

and lot-streaming (i.e., the successor item can be produced simultaneously with its predecessor items as long as it uses a different resource).

Correa and Schulz (2005) consider precedence constraints but on a single machine. This case can be seen as a subproblem of our model. Gicquel and Minoux (2015) also considers a single resource model (for processing different kinds of products) and give an exact algorithm based on a branch-and-cut procedure. The earlier paper of Gicquel et al. (2009) treats a similar model for making a single product.

We note that the scheduling model we consider in this paper is also quite special, as we assume a setting with a small number of different product types where each product consists only of a few components. Accordingly, none of the mentioned papers is directly applicable to our problem.

One can approach the considered problem also from the theory of scheduling. In the three-field-notation, the problem can be denoted as $R_m|prec|C_{max}$, i.e., given m unrelated machines, there are precedence constraints (in our case related to the operations and not to the jobs), and the makespan is to be minimized. Our model is special in the sense that the length of the chains in the precedence graph is short (contains only chains of length at most 3). We refer to Chen et al. (1998) for a review on scheduling models. Other related work dealing with unrelated machines is by Herrmann et al. (1997), Rocha et al. (2008), which does not quite fit our setting either. The considered model in the former paper does not contain any assembly operation and the preceding constraints are not restricted to be short chains; the latter one does not consider preceding constraints. In particular, there is not much work considering unrelated machines in the presence of precedence constraints except under special conditions, like when the precedence constraints are in fact chains. It is worth noting that in our model some components are assembled, that is, there are points in the precedence graph with more than one predecessor. Kumar et al. (2005) consider the same setting dealt with in this paper, but concentrate on approximation algorithms rather than on an approach to find an optimal solution as we do. Liu and Yang (2011) provide a heuristic algorithm for the problem $R_m|prec|C_{max}$ and give computational experiments. Hassan et al. (2016) consider a version where each job has to be assigned to a unique machine, whereas in our model there are several machines to which a job can be assigned.

The problem we consider could be categorized as a Flexible Job Shop Problem (FJSP), however, it is not clearly defined what is the meaning of “flexible” as there are several different definitions. We are not sure whether our model fits into any of them. Brandimarte (1993) deals with a FJSP model, but the machines in their model are identical. Fattahi et al. (2007) investigate a similar problem to ours, but they do not deal with assembly. There are several other publications on FJSP that present a quite complex problem model. These papers usually apply heuristic or meta-heuristic approaches to solve the problem, for example Tabu Search in Saidi-Mehrabad and Fattahi (2007), genetic algorithm in Pezzella et al. (2008), or simulated annealing in Najid et al. (2002).

In the last years Assembly Job Shop Scheduling (AJSS), introduced by Wan and Yan (2016) has been considered. This class contains job shop problems that process jobs requiring multiple levels of assembly. Within the AJSS framework we did not

find such publication which considers also unrelated machines (i.e., resources with different capabilities) and preceding constraints.

The most relevant paper in the literature is Hurink et al. (1994). This paper considers job-shop scheduling with multi-purpose machines, where each operation has a number of machines from which exactly one must be chosen for scheduling the operation. Each job is a sequence of operations and the processing time of an operation does not depend on the machine that processes it. There are however significant differences between their model and our model. These differences will be described in detail in Sect. 2.2.

Finally, our problem could be classified also as process scheduling, too, however, usually process scheduling does not include assembly, see Floudas and Lin (2005).

As it can be seen, our problem cannot be clearly classified into one of the presented problem classes. We deal with assembly, but we consider a simplified case:

- In our problem there are only a few task types (maximum 4 tasks in a job).
- We apply specialized machine groups.
- The problem can be easily separated into sub-problems because of the previous two properties.
- Some of the sub-problems can be solved efficiently by a greedy method.
- The preemptive case of the complex sub-problem can be solved efficiently.
- There are only a few job types (in fact only 2 job types).
- There is a huge number of jobs from each type.

If a general scheduling problem belongs to the class considered here, then our approach can be applied well. We *did not find any problem in the literature* that completely fits to our problem, which is why we do not compare our method to existing methods from the literature.

We do not aim at providing a general approach for a general production planning problem that could be widely used. Instead, we consider a relatively simple model, which is still hard to solve optimally, for which we propose a combined heuristic algorithm, which performs quite well and often finds an optimal solution. As there is no guarantee that it *always* finds the optimal solution, we also apply another method: write the model as a mixed-integer program and solve it by a MILP solver. Naturally there is a trade-off between the two methods. The former is fast and really effective, but without a guarantee for optimality. The latter surely finds an optimal solution, but for larger instances it needs a huge amount of time. In order to get the best of both worlds, we suggest to combine the two approaches, first running the heuristic and then using the output solution as a starting point for the solver. While this is a typical approach in optimization theory, the novelty of our method lies in the way we build up our heuristic: we split the master problem into subproblems that can be handled separately, applying a list of simplifications, a relaxation, a rounding technique and finally some kind of local search. Another novelty is to exploit the heuristic solution for the MILP solution process: it is not simply fed to the MILP solver as a starting solution, but the optimal solution itself (the makespan) determines the size of the planning horizon for the exact approach. Hence in general, the better the heuristic, the faster the exact approach will work. We believe that this kind of treatment can be used more generally and can also be applied to more complex problems. The efficiency of our method is confirmed by computer experiments.

2 General problem formulation

We first describe the project scheduling model we deal with in general terms. Assume N kinds of products (denoted by $P_1, P_2, \dots, P_N$) are given that have to be produced with cardinalities $n_1, n_2, \dots, n_N$. Any product is produced by executing several activities. There are B kinds of activities, also called tasks, denoted by $A_\alpha, \alpha = 1, \dots, B$. Each product is built up from these tasks, and the same kind of task can be necessary for different products.

There are precedence relations between the tasks, given in advance by a directed precedence graph G_j for each product $P_j, j = 1, \dots, N$. The vertices correspond to the tasks from which the product is built, and if task A_α has to be completed before task A_β can be started, then there is a directed edge from A_α to A_β .

We assume that r resources denoted by $R_1, \dots, R_r$ are given, which work as unrelated machines. That is, the ability of a machine for producing a task is given by pairs (A_α, R_i) and the processing time of task A_α is $p_{R_i}^{A_\alpha}$, if this task is processed by resource R_i (the processing time is set to infinite if the machine is unable to process the task). For any resource there are setup times between processing different types of tasks using the same resource, denoted by $u(R_i, A_\alpha, A_\beta)$. The goal is to minimize the makespan, i.e., to complete all products (in the requested cardinalities) in minimum time.

2.1 A special case

We consider a special instance of the problem setting described before for which finding an optimal solution is still a challenging task. We deal with the production of two different kinds of pans, that is, $N = 2$. The first product is a “stew pan”, the second a “ticker pan”. We assume that n^{stew} stew pans and n^{ticker} ticker pans have to be produced. A stew pan consists of a tiller part and a head part (we use the name “can” for the head part). A ticker pan also consists of a tiller and a can, the only difference is that the can part is punched.

The products are built from four tasks according to Fig. 1, that is, $B = 4$. Task A_1 is the production of the handle (or tiller), task A_2 is the extrusion of the can, and task A_4 is the screwing. In case of ticker pans there is one more operation, namely task A_3 , which is the punching of the can (this operation is not needed for stew pans). We suppose—for the sake of simplicity—that tasks A_1, A_2 and A_4 are identical for both products: the tillers are identical, the extrusion is the same operation and screwing is also identical for both products.

We will use two more (technical) parameters, which can be computed from the numbers of products: m^{tiller} is the “amount” of tiller raw material, so that $m^{tiller} = n^{stew} + n^{ticker}$. Similarly, $m^{can} = n^{stew} + n^{ticker}$ denotes the “amount” of can raw material.

We assume that there are 8 resources in total, i.e., $r = 8$. These resources differ in their capabilities. We can distinguish several resource types: the first three resources are identical compactors, the fourth resource is a puncheon. We assume that a compactor is also able to do punching, yet with less efficiency than a puncheon; and a puncheon

Fig. 1 Production processes of the two products

is also able to do extrusion, yet less efficiently than a compactor. Further, there are two identical lathes (R_5 and R_6), which are only able to do turning; and two identical screw-drivers (R_7 and R_8), which are responsible for final assembly. For the sake of simplicity, in the model we do not distinguish between resource types, but the parameters of resources of the same type are identical (e.g., R_1 – R_3).

Setup time is considered only for the first four resources. These resources can execute only tasks A_2 and A_3 . When they switch between different tasks (from A_2 to A_3 , or from A_3 to A_2), some amount of time is needed to change the resource settings for making the resource capable to perform the other task. This setup time is u_{R_i} (both from A_2 to A_3 and from A_3 to A_2).

Our model has a very important and peculiar property. We deal with some kind of mass production and we can postpone the decision of pairing some tasks of a job. In a typical scheduling model these two tasks belong to each other before any scheduling is made as this is the property of the input. But in our problem we can postpone the decision about linking the tasks of a job. Our heuristic scheduling algorithm will make this decision during the scheduling process. Without exploiting this option any algorithm can perform significantly worse.

The goal is to find a schedule which minimizes the makespan of the problem. That is, we want to process all stew pans and all ticker pans by the given resources in minimum total time.

2.2 Comparing our model with the model of Hurink et al.

In this subsection we point out what are the differences between the model of Hurink et al. (1994) and our model. In the job-shop scheduling model with multi-purpose machines of Hurink et al.

- there is a fixed order for the operations of any job.
- the processing time of one job does not depend on the machine that processes the job. More exactly, there are two cases: the machine is capable to process the job or it is not capable. If some machines are capable, they are the same regarding the processing time.
- for any job, the operations belonging to this job are given in advance.

In contrast, our model has the following features:

- there is not a fixed order regarding the operations made in Room1 and Room2, see Fig. 1.
- what is more important, in Room2 the processing time of an operation depends on the machine that processes the job. For example, if resource R1 makes the operation then the processing time is 7, but if resource R4 processes the job, the processing time is 11. And vice versa, if some other operation is made on resource R4 the processing time is 8 and it is 12 if the operation is made on resource R1. So resource R1 is smarter than resource R4 for some of the operations and resource R4 is smarter than resource R1 for some other operation. This is a significant difference.
- in our model in Room1 a mass-production is performed. It means that the tillers that are made here are completely the same. A tiller is not matched to a can in advance (unlike that it could be the case that for example very expensive products are made, and for any product a customer chooses its color, pattern, material and so on; then a tiller belongs to a can in advance). Here, the considered case is much different.

3 A mixed-integer linear model

In this section we give the MILP formulation of our model. It is based on the abstract and general production planning models given by Pochet and Wolsey (2006). The general idea is to use a network flow model on a time-expanded graph with flow conservation constraints at each node of the network. We remark that this approach differs from other scheduling models (such as job shop scheduling models, which are, for example, discussed by Ku and Beck 2016).

3.1 Sets and parameters

Denote by $\mathcal{T} := \{0, 1, \dots, T\}$ the set of time points. As abbreviations we set $\mathcal{T}' := \mathcal{T} \setminus \{T\}$ and $\mathcal{T}^- := \mathcal{T} \setminus \{0\}$.

The parameters given in Table 1 describe the processing capacities of the resources (machines). We emphasize that there can be significant differences between the machine capabilities. For example, there are four machines referred to by the parameters $p_{R_i}^{\text{extru}}$, $i = 1 \dots 4$, three of which are similar, and the fourth is different, cf. Table 6. This is insignificant for the model at this point, but will be important later. Further parameters describe the amount of raw material for tillers and cans, and the number of desired final products, stew pans and ticker pans, see Table 2. In Sect. 7 we discuss

Table 1 Parameters of the model

Name	Index i	Description
p_{R_i}	5&6	Processing time of R_i for each tiller (A_1)
$p_{R_i}^{\text{extru}}$	1–4	Processing time of R_i for extruding (A_2)
$p_{R_i}^{\text{punch}}$	1–4	Processing time of R_i for punching (A_3)
u_{R_i}	1–4	Setup time for R_i for changing between extruding and punching ($A_2 \leftrightarrow A_3$)
p_{R_i}	7&8	Processing time of R_i for final assembly (A_4)

Table 2 Production input and output parameters

Name	Description
m^{tiller}	Tiller raw material input
m^{can}	Can raw material input
n^{stew}	Stew pan output
n^{ticker}	Ticker pan output

Table 3 Stock variables (associated with nodes in the production network)

Name	Range	Description
$s^{\text{tillerraw}}(t)$	$\mathbb{Z}_+$	Tiller raw material at time point t
$s^{\text{tiller}}(t)$	$\mathbb{Z}_+$	Finished tillers at time point t
$s^{\text{canraw}}(t)$	$\mathbb{Z}_+$	Can raw material at time point t
$s^{\text{extruded}}(t)$	$\mathbb{Z}_+$	Extruded cans at time point t
$s^{\text{punched}}(t)$	$\mathbb{Z}_+$	Punched cans at time point t
$s^{\text{stew}}(t)$	$\mathbb{Z}_+$	Finished stew pans at time point t
$s^{\text{ticker}}(t)$	$\mathbb{Z}_+$	Finished ticker pans at time point t

two different alternatives to compute the time horizon $\mathcal{T}$, based on these production input data.

3.2 Variables

The decision variables used in this model are all binary or integer. They are divided into three categories: stocks, flows, and times. Consider the production as a directed network graph, then the stock variables s are the nodes. They describe the amount of material available for production. The resources (machines) are the arcs, which correspond to the flow variables f . Both variables are time-indexed, since they describe a dynamic setting that changes from time point to time point. The final class of variables represent certain time points in the schedule, such as the end of the production. The details can be found in Tables 3, 4, and 5.

Table 4 Flow variables (associated with arcs in the production network)

Name	i	Range	Description
$f_{R_i}(t)$	5&6	$\{0, 1\}$	R_i starts processing at time point t
$f_{R_i}^{\text{extru}}(t)$	1–4	$\{0, 1\}$	R_i starts extruding at time point t
$f_{R_i}^{\text{punch}}(t)$	1–4	$\{0, 1\}$	R_i starts punching at time point t
$f_{R_i}^{\text{extruded}}(t)$	7&8	$\{0, 1\}$	R_i starts finishing stew pan at time point t
$f_{R_i}^{\text{punched}}(t)$	7&8	$\{0, 1\}$	R_i starts finishing ticker pan at time point t
$f_{R_i}^{\text{tiller}}(t)$	7&8	$\{0, 1\}$	R_i starts finishing any pan with tiller at time point t

Table 5 Time variables

Name	i	Range	Description
$C_{\max}$		$\mathbb{Z}_+$	Makespan
ℓ_{R_i}	7&8	$\mathbb{Z}_+$	Time point of end of manufacturing on R_i

3.3 Constraints

We divide the model into three subproblems handled in three “rooms” described in detail in the following.

3.3.1 Room1: Producing the handles

In Room1, the tillers are produced by resources R_5 and R_6 from tiller raw material.

We have the following initial conditions:

$$\begin{aligned}s^{\text{tillerraw}}(0) &= m^{\text{tiller}}, \\ s^{\text{tiller}}(0) &= 0,\end{aligned}$$

which places all tiller raw material in front of R_5 and R_6 and assumes an empty stock of finished tillers at time point 0. The boundary conditions enforce no production in the very last time point T :

$$f_{R_5}(T) = f_{R_6}(T) = 0,$$

There are two nodes associated with Room1, which both give rise to flow conservation constraints. The tiller raw material of time point t is moved into the two resources R_5 , R_6 , and the remaining tillers are moved to the next time point:

$$s^{\text{tillerraw}}(t) = s^{\text{tillerraw}}(t+1) + f_{R_5}(t) + f_{R_6}(t), \quad \forall t \in \mathcal{T}'.$$

The flow conservation at the node at the other end of R_5 , R_6 takes the finished materials and either stores them as a stock or moves them forward to Room3 for the final assembly:

$$s^{\text{tiller}}(t-1) + f_{R_5}(t-p_{R_5}) + f_{R_6}(t-p_{R_6}) \\ = s^{\text{tiller}}(t) + f_{R_7}^{\text{tiller}}(t) + f_{R_8}^{\text{tiller}}(t), \quad \forall t \in \mathcal{T}^-.$$

Note that here and in all following equations, if the time index of a variable is not in $\mathcal{T}$, we replace the variable by the value 0. In the above equation that means we replace $f_{R_5}(t-p_{R_5})$ by 0, whenever $t-p_{R_5} \notin \mathcal{T}$.

The production within R_5, R_6 takes a certain number of time steps, and only one item (raw material) can be processed within that time span:

$$\sum_{\Delta \in \{0, \dots, p_{R_i}-1\}} f_{R_i}(t-\Delta) \leq 1, \quad \forall t \in \mathcal{T}, i = 5, 6.$$

3.3.2 Room2: Extrusion and punching

In Room2, the can raw material is extruded on one of the resources $R_1, \dots, R_4$. The extruded cans are either moved to Room3 for final assembly, or they are further processed in Room2, where they are punched on one of the resources $R_1, \dots, R_4$.

As initial condition, all can raw material is placed in front of machines $R_1, \dots, R_4$ at time point 0:

$$s^{\text{canraw}}(0) = m^{\text{can}}.$$

There is no possibility for punching at time point 0:

$$f_{R_1}^{\text{punch}}(0) = \dots = f_{R_4}^{\text{punch}}(0) = 0,$$

There is an empty stock of extruded and punched cans at time point 0:

$$s^{\text{extruded}}(0) = s^{\text{punched}}(0) = 0.$$

As boundary conditions, it is required that no processing takes place in the very last time point T :

$$f_{R_1}^{\text{extru}}(T) = \dots = f_{R_4}^{\text{extru}}(T) = f_{R_1}^{\text{punch}}(T) = \dots = f_{R_4}^{\text{punch}}(T) = 0.$$

Can raw material is either extruded on one of the machines $R_1, \dots, R_4$, or it is moved to the next time point as raw material:

$$s^{\text{canraw}}(t) = s^{\text{canraw}}(t+1) + f_{R_1}^{\text{extru}}(t) + \dots + f_{R_4}^{\text{extru}}(t), \quad \forall t \in \mathcal{T}'.$$

The extruded cans are either stored, or punched on one of the machines $R_1, \dots, R_4$, or moved for final assembly to the machines R_7, R_8 in Room3:

$$s^{\text{extruded}}(t-1) + f_{R_1}^{\text{extru}}(t-p_{R_1}^{\text{extru}}) + \dots + f_{R_4}^{\text{extru}}(t-p_{R_4}^{\text{extru}}) \\ = s^{\text{extruded}}(t) + f_{R_1}^{\text{punch}}(t) + \dots + f_{R_4}^{\text{punch}}(t) + f_{R_7}^{\text{extruded}}(t) + f_{R_8}^{\text{extruded}}(t), \quad \forall t \in \mathcal{T}^-.$$

The punched cans are either stored, or moved for final assembly to the machines R_7, R_8 in Room3:

$$\begin{aligned} s^{\text{punched}}(t-1) + f_{R_1}^{\text{punch}}(t - p_{R_1}^{\text{punch}}) + \dots + f_{R_4}^{\text{punch}}(t - p_{R_4}^{\text{punch}}) \\ = s^{\text{punched}}(t) + f_{R_7}^{\text{punched}}(t) + f_{R_8}^{\text{punched}}(t), \quad \forall t \in \mathcal{T}^-. \end{aligned}$$

The extrusion within $R_1, \dots, R_4$ takes a certain number of time steps, and only one item (can raw material) can be processed within that time span:

$$\sum_{\Delta \in \{0, \dots, p_{R_i}^{\text{extru}} - 1\}} f_{R_i}^{\text{extru}}(t - \Delta) \leq 1, \quad \forall t \in \mathcal{T}, i = 1, \dots, 4.$$

Similarly, the punching within $R_1, \dots, R_4$ takes a certain number of time steps, and only one item (extruded can) can be processed within that time span:

$$\sum_{\Delta \in \{0, \dots, p_{R_i}^{\text{punch}} - 1\}} f_{R_i}^{\text{punch}}(t - \Delta) \leq 1, \quad \forall t \in \mathcal{T}, i = 1, \dots, 4.$$

When changing from extrusion to punching, the setup time must be obeyed:

$$f_{R_i}^{\text{extru}}(t) + f_{R_i}^{\text{punch}}(t + \Delta) \leq 1, \quad \forall \Delta \in \{0, \dots, p_{R_i}^{\text{extru}} + u_{R_i} - 1\}, i = 1, \dots, 4.$$

Similarly, when changing from punching to extrusion, the setup time must be obeyed:

$$f_{R_i}^{\text{punch}}(t) + f_{R_i}^{\text{extru}}(t + \Delta) \leq 1, \quad \forall \Delta \in \{0, \dots, p_{R_i}^{\text{punch}} + u_{R_i} - 1\}, i = 1, \dots, 4.$$

3.3.3 Room3: Final assembly

In Room3, two resources R_7, R_8 are screwing tillers to extruded cans or punched cans as final assembly step, which results in stew pans or ticker pans.

As initial conditions, we assume that the stock of stew pans and ticker pans is empty:

$$s^{\text{stew}}(0) = s^{\text{ticker}}(0) = 0.$$

The flow of extruded and punched cans and tillers through resources R_7, R_8 is initially empty:

$$f_{R_i}^{\text{extruded}}(0) = f_{R_i}^{\text{punched}}(0) = f_{R_i}^{\text{tiller}}(0) = 0, \quad i = 7, 8.$$

As boundary conditions, there is no flow in the very last time point:

$$f_{R_i}^{\text{extruded}}(T) = f_{R_i}^{\text{punched}}(T) = f_{R_i}^{\text{tiller}}(T) = 0, \quad i = 7, 8.$$

The final stocks of stew pans and tillerpan must meet the required demand:

$$\begin{aligned}s^{\text{stew}}(T) &= n^{\text{stew}}, \\ s^{\text{ticker}}(T) &= n^{\text{ticker}}.\end{aligned}$$

The stock of stew pans is increased by adding finished products coming out of R_7, R_8 :

$$s^{\text{stew}}(t-1) + f_{R_7}^{\text{extruded}}(t - p_{R_7}) + f_{R_8}^{\text{extruded}}(t - p_{R_8}) = s^{\text{stew}}(t), \quad \forall t \in \mathcal{T}^-.$$

Similarly, the stock of ticker pans is increased:

$$s^{\text{ticker}}(t-1) + f_{R_7}^{\text{punched}}(t - p_{R_7}) + f_{R_8}^{\text{punched}}(t - p_{R_8}) = s^{\text{ticker}}(t), \quad \forall t \in \mathcal{T}^-.$$

Only one tiller can be handled during the processing time in R_7, R_8 :

$$\sum_{\Delta \in \{0, \dots, p_{R_i}-1\}} f_{R_i}^{\text{tiller}}(t - \Delta) \leq 1, \quad \forall t \in \mathcal{T}, i = 7, 8.$$

Only one extruded can can be handled during the processing time in R_7, R_8 :

$$\sum_{\Delta \in \{0, \dots, p_{R_i}-1\}} f_{R_i}^{\text{extruded}}(t - \Delta) \leq 1, \quad \forall t \in \mathcal{T}, i = 7, 8.$$

Only one punched can can be handled during the processing time in R_7, R_8 :

$$\sum_{\Delta \in \{0, \dots, p_{R_i}-1\}} f_{R_i}^{\text{punched}}(t - \Delta) \leq 1, \quad \forall t \in \mathcal{T}, i = 7, 8.$$

A pan consists of a tiller together with either one punched can or one extruded can, which must be assembled at the same time:

$$f_{R_i}^{\text{tiller}}(t) = f_{R_i}^{\text{punched}}(t) + f_{R_i}^{\text{extruded}}(t), \quad \forall t \in \mathcal{T}, i = 7, 8.$$

3.4 Makespan and objective

The makespan is the time when the last pan is completed on R_7 or R_8 :

$$(t + p_{R_i}) \cdot f_{R_i}^{\text{tiller}}(t) \leq \ell_{R_i}, \quad \forall t \in \mathcal{T}, i = 7, 8,$$

and

$$\ell_{R_i} \leq C_{\max}, \quad \forall i = 7, 8,$$

and our objective is to minimize the total makespan $C_{\max}$.

4 Divide et Impera

The previous section shows that an exact description of the model is quite complex. Also finding an exact solution is a challenging task, even in the considered special case. As will be seen in Sect. 7, without the help of a heuristic, a standard solver can already have problems solving the model optimally even in this moderately sized setting. In this section, we propose a heuristic method to obtain a more effective way of finding a solution. Our first consideration is that the problem can be separated into three quite independent subproblems corresponding to the “rooms” introduced in the previous section. We handle them one-by-one, this is the meaning of the ancient latin phrase “Divide et Impera”.

In this section we give a general description about the heuristics to give some insight into the machinery.

Room1: producing the handles

It is easily confirmed that in this room a greedy method is sufficient. Since the handles are the same (having the same processing times, there is no release time and no deadline dedicated to the handles), we distribute the handles equally (or almost equally if the number of machines does not divide the number of handles) among the machines, which must be an optimal solution for this subproblem.

Room3: the final assembly

For this kind of problem a greedy method is sufficient, too. Here the solution is a bit more difficult than in Room1, but still easy. As soon as a handle is made ready (and transported from Room1 to Room3) and also a head arrives (transported from Room2 to Room3), and also a machine is free to start a new operation, we start to assemble the handle and the head together just at this time. This is an optimal strategy, if all stew pans and ticker pans are identical regarding their assembly times.¹ Hence, we do not need to wait or insert idle times in this room and can solve the subproblem of this room by a greedy strategy.

Room2: the room of “secrets”

This is the hardest subproblem among the three, which can be still easy in some cases, depending on the processing times of the operations (including the processing times of operations of *other* rooms). That is, if the “bottleneck” occurs in the first room (the production of the handles needs much time compared to the other operations in the other rooms), making a “lazy” schedule in Room2 will be still good enough to reach an overall optimal schedule. Otherwise, if we have very constrained time in Room2

¹ Generally, it could happen that we make dedicated products for different customers, where a high-quality product needs larger processing times compared to a common quality product. However, this is not the case in our simplified model.

(compared to the amount of work in the other rooms), with the help of some tricks, introduced in the next section, we are still able to get a quite good overall schedule.

5 Room2: A heuristic through simplifications

We will make a sequence of simplifications regarding the subproblem of the second room. Suppose there are k machines in Room2 ($k = 4$ in our case). Assume moreover, that these machines are of two types.

There are k_1 machines which are better for extrusion and less effective for punching. We refer to these machines as “extruding machines”. Also there are k_2 machines which are better for punching and less effective for extruding. We call these machines “punching machines”. Naturally, $k_1 + k_2 = k$, and in our case $k_1 = 3$ and $k_2 = 1$.

The question is how to distribute the extruding and punching operations among the machines. At this point we make several simplifications. It can happen that some of our simplifications hold in any optimal solution of the problem; while others will not hold in general. However, our goal is to find a feasible solution, which in turn can be used to find an optimal solution using a MILP solver, see below for details. For this purpose we do not prove optimality because we do not build on it at all.

The simplifying assumptions made for Room2 (S1–S5) are all about the *hypothetical* form of the optimal solution. The first two simplifications are as follows.

(S1) On each machine, there is at most one setup.

(S2) On each machine, any extruding operation precedes any punching operation.

The intuitive reasoning for these two assumptions is as follows. Suppose we have already assigned operations to some machine. Then moving the extruding operations to the beginning (followed by the punching operations), the makespan will not increase. By computer examinations we found that assumptions S1 and S2 hold in many cases. However, in some rare cases assumption S1 is not satisfied. In those cases when the optimum is better than the solution provided by the heuristic, some punching must be made before extruding, so that the assembly operations of the third room can be started earlier. In this case there will be more than one setup time on some machines. So the satisfaction of S1 and S2 strongly depends on the processing times of the different operations. They are often satisfied, but not always.

(S3) Either extruding machines make no punching, or punching machines make no extruding.

Suppose this is not the case. It means that there is an extruding machine which makes also punching (for this operation this machine is less effective than the punching machine), and also, there is a punching machine that makes also extruding operations (for which this machine is less effective). If we swap the two operations in consideration, the same number of extruding and punching operations will be made. Moreover the completion time of both machines in consideration will decrease. Unfortunately, in general it does not hold that the makespan will decrease or at least, remain the same.

According to S3, there are two cases to consider:

Case 1: Any punching machine makes only punching operation.

Case 2: Any extruding machine makes only extruding operations.

Moreover, within Case 1 we can distinguish the following possible situations: There are $i \in \{0, 1, \dots, k_1\}$ extruding machines that make only extruding operations, and the other $k_1 - i$ extruding machines make both extruding and punching operations (and all punching machines make only punching operations). Similarly, within Case 2 there are $k_2 + 1$ possible subcases, according to the number of punching machines that make both operations. We will consider all these $k_1 + 1 + k_2 + 1 - 1 = k + 1$ possible subcases one by one, and finally we will choose that subcase in which we find the best solution for our heuristic.

From now on, let us suppose that we consider a subcase within Case 1 (Case 2 can be handled similarly), where (simplifying the notation) there are α extruding machines that make only extruding operations, there are β extruding machines that make both extruding and punching operations, and there are γ punching machines making only punching operations. (Naturally, $0 \leq \alpha \leq k_1$, $\alpha + \beta = k_1$, and $\gamma = k_2$.) We will refer to this subcase as “chosen subcase” or “subcase in consideration”. We also assume for the sake of simplicity that all values α , β , γ are positive. If any of them is zero, this subcase is handled similarly.

5.1 Assignment versus schedule

An *assignment* is a mapping of the operations to the machines, without setting exact starting times of the operations. If we also determine the starting times (and finishing times) of all operations in a feasible way, we call it a *schedule*. Now we make a further simplification.

- (S4) Any punching machine can make all its work without any idle time (except that any punching machine starts at time ϵ , where ϵ is the processing time of extruding on the extruding machines), and any extruding machine which makes also punching, can work on punching continuously after the setup time is inserted after the last extruding.

After this simplification, our main perception is that we easily can make a schedule from an assignment. It means that in fact, we look for an optimal assignment. If we find an optimal assignment of the operations to the machines (and all previous simplifications hold), we can easily obtain an optimal schedule from this assignment as follows: Any extruding operation will be started without idle time. We know (by the simplifications) that in the “chosen subcase” when a punching operation is started, we do have an appropriate head to punch it. This means that on any extruding machine that also does punching, these punching operations can be started immediately after inserting a setup time between the last extruding operation and first punching operation on that machine. Moreover, any machine that makes only punching, will start the work just when the first head is made ready on the extruding machine.

5.2 Optimal preemptive assignment

We consider the chosen subcase. We have seen that our main task is to find a good assignment, so we focus on this in the present subsection. We will construct a preemptive (optimal) assignment of the operations to the machines. (Preemptive schedule means that some part of a job is made by some machine, and then, the remainder part of this job is continued on the same, or possibly on some other machine.) How shall we distribute the extruding and punching operations among the machines? Naturally, the punching machines (in the chosen subcase) get only punching operations. We can assume (because of the preemptive solution) that all these machines get the same (possibly fractional) number of punching operations, let this number be x . Also, the α extruding machines that get only extruding operations, will all get the same number y of extruding operations. Finally, we assume that each extruding machine among the β machines gets v extruding operations and w punching operations.

Then the optimal values of x , y , v , w can be found easily solving the next system of equations. Simplifying the notation, let ζ denote the setup time between the extruding and punching operation, moreover let ϵ be the processing time of extruding (by an extruding machine), let η be the processing time of punching by a punching machine, and let θ be the processing time of punching for an extruding machine.

$$\alpha \cdot y + \beta \cdot v = n_1 + n_2 \quad (1)$$

$$\beta \cdot w + \gamma \cdot x = n_2 \quad (2)$$

$$\gamma \cdot \epsilon = v \cdot \epsilon + w \cdot \theta + \zeta = x \cdot \eta + \epsilon \quad (3)$$

We have four variables (x , y , v and w) and four equations. Here (1) means that we sum up the number of extruding operations, similarly in (2) we count the number of punching operations. Finally in (3) we assume that in the optimal preemptive solution the completion times for all machines are the same. For the β machines that make both extruding and punching we have taken into account the setup time, moreover for the punching machines we have taken into account that they can start their work only after a head is ready (on some other machine) as no punching can be made until a head is ready for punching it.

We have discussed why the punching machines start at time ϵ , when the extruding machines give off their first ready heads. The other part of the assumption (i.e., punching is made without idle times) not surely holds for *any* subcase (within the $k + 1$ subcases). But we assume that there will be such a subcase among all subcases for which this assumption also holds, thus this assumption will hold in the chosen subcase. In fact, we are interested only in this optimal subcase. It means, we used implicitly also the next assumption:

(S5) The solution of the above system of Eqs. (1)–(3) is nonnegative, i.e., $x, y, v, w \geq 0$.

If at least one variable is negative in the solution, we simply omit the subcase from consideration (as it cannot be the chosen subcase). This negative value can happen for example if there are too many machines for punching, and (consequently) too few machines for extruding. In this case (3) cannot hold with non-negative variables, or, in

other words, if we assume that (3) holds, this will provide some negative variable in the solution. But, as mentioned before, in case of considering the best possible subcase, the assumptions are assumed to be valid.

5.2.1 A small example

Let us consider a small example. Let $n_1 + n_2 = 35$, $n_2 = 20$ (that means 35 is the total number of products, all of them have to be extruded but only 20 of them require punching), the setup time is $\zeta = 3$, the processing time of extruding by an extruding machine $\epsilon = 6$, processing time of punching by a punching machine $\eta = 7$, processing time of punching for an extruding machine $\theta = 10$. Moreover, if $\alpha = 1$, $\beta = 2$, $\gamma = 1$, we get the solution: $x = 398/31 \approx 12.839$, $y = 1486/93 \approx 15.978$, $v = 1769/186 \approx 9.5108$ and $w = 111/31 \approx 3.5806$.

We see that all variables are non-negative, but “unfortunately” the values of the variables are fractional values.

5.3 Non-preemptive from preemptive: rounding

We got the optimal preemptive solution for the subproblem of Room2 for a considered subcase. But we need a non-preemptive schedule. For this purpose we perform the next simple rounding procedure: first all variables are rounded down. Next, the number of extruding operations assigned to the machines is increased one by one, so we increase by 1 the number of such operations for the first machine among the k_1 machines, then we increase by 1 the number of such operations for the second machine among the k_1 machines, and so on, until we get the needed n_1 number of such operations. The number of punching operations is increased by 1 similarly for the punching machines one by one, and if needed, we continue for the extruding machines that also do punching.

For the above considered subcase this works as follows. After rounding down the variables we get that there are 15 extruding operations assigned to the first extruding machine, and 9 extruding operations are assigned to any extruding machine that performs also punching. These two machines get one by one 3 punching operations, and finally the punching machine makes 12 punching operations. Now the total number of extruding operations assigned to the machines so far is $15 + 2 \cdot 9 = 33$ but we need 35 such operations. Also, the number of punching operations assigned to the machines is $2 \cdot 3 + 12 = 18$ but we need 20 such operations.

After increasing the number of operations, any of the first two machines gets one more extruding, and any of the last two machines gets one more punching operation. Thus the first machine will execute 16 extrudings, the second machine 10 extrudings and 3 punchings, the third machine 9 extrudings and 4 punchings, and the last machine will execute 13 punching operations.

5.4 Further improvements by Local Search or Tabu Search

After we get a “rough assignment” (we will use this term in the following) for the subproblem of Room2, we try to improve it. For the improvement we apply Local

Search and independently Tabu Search. Then we take the better solution and this will be given to the next step of our heuristic. We describe the Local Search and the Tabu Search in details below.

5.4.1 Local Search added

For the Local Search we use the following elementary steps.

- MOVE: We take an extrusion operation or a punching operation assigned to a certain machine and move this operation to some other machine.
- SWAP: We swap one extrusion operation assigned to a certain machine with a punching operation assigned to some other machine.

We will make these steps in a “clever” way, i.e., if we already decided, e.g., that some machine will not perform extrusion operations, then this machine cannot get extrusion operations by local changes. It is important to note that we consider only critical operations for moving and swapping. From this point of view the notions of *critical path* and *block* within a critical path are crucial, see the exact definition in Hurink et al. (1994). An operation is critical if it has a direct influence on the makespan. It means (in our model) that this operation is a member of some chain of operations such that each operation starts as soon as it can be started after its preceeding operation and this chain of operations determines the makespan. We get a critical block if we take a critical path of operations and we consider some operations of this path of operations which are made just after each other by a certain machine. We note that the sequence on the same machine may be altered by moving critical operations before or after critical blocks. Note that in the definition of Hurink et al. any block contains at least two operations, and we note also that there may be more than one critical paths. Our Local Search procedure works as follows: The initial state is the rough assignment we get after the rounding procedure. We make a local change by MOVE or SWAP, and calculate the schedule from the new assignment, and the makespan of the new schedule. If the makespan of the new schedule is better than that of the “old” schedule, we accept the local change, otherwise we reject it. We perform the local changes several times, where the number of trials is prescribed in advance. There is no guarantee that the final solution is really optimal, but we do not care about this, as we construct “only” a heuristic solution. In fact, our experiments below show that many times we find the optimal solution.

5.4.2 Tabu search added

Instead of Local Search for the improvement of the rough assignment we can apply also some other methods. We chose the method of Hurink et al. (1994) for this purpose, as this method has been proved very efficient for job-shop scheduling problems with multi-purpose machines. The jobs consist of a predefined sequence of operations. These operations have to be processed in a given order, any operation has to be performed by exactly one machine out of a set of machines, and the processing time does not depend on the chosen machine. The method starts from an initial “good” solution that is obtained by a tricky greedy heuristic.

Starting from the initial schedule, Hurink et al. apply Tabu Search where neighbourhood techniques are used that are built on the disjunctive graph model of the schedule. The base of their neighbourhood techniques is the finding of blocks of the critical path(s). A block has three important features:

- it has at least two operations.
- the operations of a block are successive operations on the same machine.
- increasing the size of the block by one operation in the disjunctive graph model of the schedule results in a sequence that violates the previous assumption.

After finding the blocks in a schedule, a neighbour of a solution can be generated in three different ways:

- an arbitrary (not first) operation of a block is moved to the first (or first feasible) position of its block.
- an arbitrary (not last) operation of a block is moved to the last (or last feasible) position of its block.
- an arbitrary operation of a block is moved to another machine that is able to operate.

We implemented this method, but note that our problem is not the same, since we do not have dedicated products. That is for each tiller a predefined can part has to be assembled, and it is also predefined which can has to be punched and which does not have to be. Contrary to that, we deal with mass production and the products are not dedicated. It means that a tiller can be paired with any can. What is more important, any can can be punched, their selection for punching is not prescribed in advance. As a consequence we have a huge freedom in Room2 for choosing the appropriate operations. Of course, if we do not utilize this freedom then we get much worse results. That is the reason why our problem differs a lot from the problem of Hurink et al. (1994).

Still, we implemented the algorithm of Hurink et al. to our problem: after a rough assignment we discovered its critical path(s), determined the block on it/them and applied Tabu Search for improving the solution by defining the neighbourhood in the same three ways as Hurink et al. did.

We experienced that both Local Search and Tabu Search were able to improve the rough assignment a bit but not much. We describe the results in details later in Sect. 7.2.1.

5.5 Exact calculation of the schedule

The assignment we obtain after applying the mentioned assumptions and simplifications finally needs to be extended to give a schedule. Note that the assignment we made is always feasible. The only point in question is the schedule of the punching operations. We try to do these operations as early as possible. For this, for any punching operation we need to find a head that is ready to being processed. We can obtain a schedule in the following simple way: we execute the extruding operations (as soon as possible). Any head which is ready is stored in a virtual warehouse. Then any punching operation is started as soon as possible, supposing that there is a machine that can start

the execution, and there exists a ready head in the virtual warehouse. By this simple (greedy) method we get from the given assignment a feasible schedule.

5.6 Short summary of the heuristic

Here we summarize our heuristic solution. We divided the master problem to three subproblems (Room1–Room2–Room3). The subproblem of the first room is solved optimally by a simple greedy method.

Then we consider the subproblem of Room2. We make simplifications, and divide the subproblem into several $(k + 1)$ subcases, where k is the number of machines in Room2. In any subcase, we calculate a preemptive optimal solution (if there is at least one negative variable in this solution, these subcases are excluded from subsequent considerations, all other subcases remain). Then by a rounding procedure we get a rough non-preemptive solution for the subcase. Performing a Local Search or Tabu Search method we try to improve the solution. Afterwards, we compare the solutions we got for all subcases that remained in competition, and choose the best one (this is the moment when we decide that this is the “chosen” subcase). For this chosen assignment we calculate the schedule with our exact method described in Sect. 5.5.

Now we transport the ready handles from Room1 into a virtual warehouse. Any handle is transported as soon it is made ready. We also transport the ready items (after extruding and punching them) from Room2 into this virtual warehouse. In other words, any handle and any head gets a release time for the assembly operation.

Finally in Room3 we assemble the handles and heads by another greedy method: As soon as a handle and also a head is ready for execution and a machine is free, we assembly the two parts together. This concludes the description of our heuristic method for the master problem.

6 Discussion on the MILP model

The model has $7|T|$ many integer and $16|T|$ many binary variables, and $28 + 25|T| + 4|T|c_1 + 4|T|c_2$ many constraints, with $c_1 := \max_{i=1,\dots,4}\{p_{R_i}^{\text{extru}} + u_{R_i}\}$ and $c_2 := \max_{i=1,\dots,4}\{p_{R_i}^{\text{punch}} + u_{R_i}\}$.

We notice that in the proposed integer program we are able to have the advantages of the ideas of Sect. 4 to strengthen the MILP model. One such point is that we know how to obtain an optimal solution regarding Room1. This means that we can fix the values of the appropriate variables of the raw-materials when they are produced. That means, we fix

$$f_{R_i}(t) := \begin{cases} 1, & \text{if } t \bmod p_{R_i} \equiv 0 \text{ and } t < \frac{1}{2}p_{R_i} \cdot m^{\text{can}}, \\ 0, & \text{else} \end{cases}, \quad i = 5, 6, t \in T.$$

As for Room3, we cannot determine the schedule at once, but still we are able to make some simplifications also here. It is well known, that excluding the “symmetry” in the solution will reduce the computation time. Let us recall that the two machines

in Room3 are completely the same, so we can include some property about this in the model. One option is requiring that the completion times of these two machines cannot differ “too much”. Another option is to prescribe that the number of operations made by these two machines cannot differ too much. We applied another option to exclude the symmetric solution, prescribing that the completion time of one of the machines is not bigger than the completion time of the other machine. In terms of linear constraints, these conditions are formulated as follows:

$$\ell_{R_7} \geq \ell_{R_8},$$

and

$$\sum_{t \in \mathcal{T}} f_{R_7}^{\text{tiller}}(t) = \left\lceil \frac{m^{\text{can}}}{2} \right\rceil,$$

$$\sum_{t \in \mathcal{T}} f_{R_8}^{\text{tiller}}(t) = \left\lceil \frac{m^{\text{can}}}{2} \right\rceil.$$

In our experiments regarding the modified model it turns out that these modifications are able to speed up the solution process by a factor of 3 on average. We give more details about the speeding up in Sect. 7.2.

7 Computational experiments

7.1 Evaluation settings

The MILP problem can be solved either by the heuristic alone, or by a MILP solver, which is by today’s standard using a branch-and-cut framework (solving linear relaxations by the revised dual version of Dantzig’s Simplex algorithm). In order to set up an instance of the model, it is necessary to select an appropriate value for the horizon $\mathcal{T}$. To this end, we experiment with one lower bound and three upper bound approaches.

All upper bound approaches utilize an estimation of $\mathcal{T}$ from above. In this way, the variable space is large enough to host an optimal solution. However, a coarse estimation leads to large instances, and thus results in long solution times for the MILP solver. The estimation is based on the idea that all machines in one room operate in parallel, but the operations in the next room only start after all jobs in the previous room are finished. We then get:

$$\begin{aligned} \mathcal{T}^{\text{ub}} := & \lceil m^{\text{tiller}} \cdot (p_{R_5}^{-1} + p_{R_6}^{-1})^{-1} \rceil \\ & + \lceil m^{\text{can}} \cdot ((p_{R_1}^{\text{extru}})^{-1} + \dots + (p_{R_4}^{\text{extru}})^{-1})^{-1} \rceil \\ & + \max\{u_{R_1}, \dots, u_{R_4}\} \\ & + \lceil n^{\text{ticker}} \cdot ((p_{R_1}^{\text{punch}})^{-1} + \dots + (p_{R_4}^{\text{punch}})^{-1})^{-1} \rceil \\ & + \lceil m^{\text{tiller}} \cdot (p_{R_7}^{-1} + p_{R_8}^{-1})^{-1} \rceil. \end{aligned}$$

Table 6 Intervals for random data

Name	Interval	Constraints
p_{R_5}, p_{R_6}	[2,6]	$p_{R_5} = p_{R_6}$
$p_{R_1}^{\text{extru}}, p_{R_2}^{\text{extru}}, p_{R_3}^{\text{extru}}$	[4,8]	$p_{R_1}^{\text{extru}} = p_{R_2}^{\text{extru}} = p_{R_3}^{\text{extru}}$
$p_{R_4}^{\text{extru}}$	[9,13]	
$p_{R_1}^{\text{punch}}, p_{R_2}^{\text{punch}}, p_{R_3}^{\text{punch}}$	[10,15]	$p_{R_1}^{\text{punch}} = p_{R_2}^{\text{punch}} = p_{R_3}^{\text{punch}}$
$p_{R_4}^{\text{punch}}$	[5,7]	
$u_{R_1}, u_{R_2}, u_{R_3}, u_{R_4}$	[0,5]	$u_{R_1} = u_{R_2} = u_{R_3}$
p_{R_7}, p_{R_8}	[2,4]	$p_{R_7} = p_{R_8}$
n^{stew}	[30,70]	
n^{ticker}	[30,70]	$n^{\text{ticker}} = 100 - n^{\text{stew}}$

This bound can be computed directly from the given input data in $O(1)$, but turns out to be rather sloppy. A much better bound is given by the heuristic. Thus in a further variant of the upper bound approach, we first compute a feasible solution using the heuristic, with a makespan of M^* . Then we set up the mixed-integer model with a time horizon of $\mathcal{T}^{\text{heur}} := M^*$, and additionally hand over this feasible solution as a starter. For the third variant of the upper bound approaches, we set up the model with a time horizon of $\mathcal{T}^{\text{ub2}} := M^* - 1$, which renders the heuristic solution infeasible, but further reduces the burden for the MILP solver.

The lower bound approach estimates $\mathcal{T}$ from below. Here we consider an idealized situation where all processors can work in parallel in all rooms, and there is no set up time between the processes:

$$\begin{aligned} \mathcal{T}^{\text{lb}} := \min \{ & \lfloor m^{\text{tiller}} \cdot (p_{R_5}^{-1} + p_{R_6}^{-1})^{-1} \rfloor, \\ & \lfloor m^{\text{can}} \cdot ((p_{R_1}^{\text{extru}})^{-1} + \dots + (p_{R_4}^{\text{extru}})^{-1})^{-1} \rfloor \\ & + \lfloor n^{\text{ticker}} \cdot ((p_{R_1}^{\text{punch}})^{-1} + \dots + (p_{R_4}^{\text{punch}})^{-1})^{-1} \rfloor, \\ & \lfloor m^{\text{tiller}} \cdot (p_{R_7}^{-1} + p_{R_8}^{-1})^{-1} \rfloor \}. \end{aligned}$$

In general, setting up the model with such value for $\mathcal{T}$ will lead to an infeasible MILP. It turned out that the solver is actually quite fast in detecting this infeasibility. Then $\mathcal{T}$ is incremented by 1, and this whole procedure is iterated, until a feasible (thus optimal) solution is found.

We created a test set of 1000 randomly generated instances, where the parameters are chosen according to a uniform distribution over the integers in the intervals shown in Table 6.

For numerically solving the MILP instances, we run IBM ILOG CPLEX 12.7.0.0 (or Cplex, for short) on a 2014 MacBookPro with a 2.8 GHz Intel Core i7 CPU and 16 GB 1600 MHz DDR3 RAM.

Fig. 2 Makespan via heuristic and exact solvers. Horizontal axis: index of the instance, vertical axis: makespan

7.2 Evaluation of the results

7.2.1 The heuristic

Figure 2 shows the makespan values of the example instances that we got applying the heuristic solver (blue curve), and the optimum values of makespan we got applying the exact solver (red curve).

For example, let us focus on instance No. 121. For this instance the optimum is 212 and the heuristic gave 254. It means that the absolute error is 42, while the relative error is almost 20%. However, for instance No. 318 both the optimal and heuristic makespan are 252. All the 1000 instances are ordered according to increasing makespan. This result will be considered in more detail in Fig. 4. The figure shows the frequency of the cases when the heuristic provides a relatively weak solution compared to the optimal solution and the frequency of finding the optimum or finding a value that is very close to the optimum.

Based on Fig. 2 an interesting fact can be noticed. Consider the instances where the slope of the graph of the optimal makespan is close to zero (instances 303–409 and instances 689–883). Here the heuristic solution was almost always optimal. This is not the case for the other instances. Let us divide the remaining instances into three parts: the first part is where the index of the instance is at most 302, part 2 is in the middle, where the indexes are between 303 and 688, while the third part is where the indexes are bigger than 884. Here the slope significantly differs from 0. We note that the heuristic performed the worst in the first part, it is much better in the middle part and almost optimal in the third part. Some more details are visualized in Fig. 3 (part a.), where

Fig. 3 Comparing the heuristic and the optimal solution on different ranges of the makespan

for the more detailed analysis we divided the 1000 instances into 5 ranges. In Fig. 3 (part b.) we show the differences between the averages of heuristic and the optimal solutions. The reason may be that if the optimal makespan is relatively small, then after making a wrong decision, the heuristic is not able to correct it. If the makespan is relatively big (this can be the consequence of the higher number of products), then the heuristic does have some opportunity to correct some wrong decision later.

The computation time for the heuristic was quick: two minutes for the 1000 instances (together with the resulting text file generation), which means 0.12 s for one instance in average. The calculation of the exact solution usually takes more time: there were cases when 10,000 s were not enough for one instance.

Figure 4 presents the relative error distribution where the makespan of the heuristic solution is compared to the optimal solution. The first (tall) bar shows that in about 460 instances among the 1000 instances the heuristic solution is in fact optimal. In case of further 240 instances (second bar) the relative error is smaller than 1%. It means that in about 70% of the cases the relative error is really small, below 1%. In 80% of the cases the relative error is at most 2%, moreover, the relative error is at most 5% in 95% of the cases. On the basis of these results it can be stated that the heuristic algorithm is effective.

Next, we investigated how the heuristic performs as a function of the ratio of the numbers of stew pans and ticker pans (in this order). Recall, that the number of ticker pans is hundred minus the number of stew pans. The random number of stew pans—and so, the number of ticker pans, too—was between 30 and 70. On Fig. 5 part a. the number of stew pans is growing from 30 to 70 on the horizontal axis. Accordingly, on Fig. 5 part b. the ratio of the stew pans and ticker pans grows from $3/7$ to $7/3$. In Fig. 6 the results are summarized by taking the averages of four ranges.

On one hand, we can recognize that for any range of the horizontal axis there are instances for which the heuristic solution is optimal, but there are also a few weaker solutions. However, we can detect certain characteristics. The relative error of the heuristic is the smallest when the number of stew pans is small (at most 39 stew pans and at least 61 ticker pans). The conclusion comes from the fact that the bar on the left hand side on Fig. 6 is the lowest and the top-left corner of Fig. 5 is empty.

Recall that for the improvement of the rough assignment during the heuristic we applied two methods. One was Local Search (LS) (see Sect. 5.4.1), the other option was Tabu Search (TS) (see Sect. 5.4.2). Our main experiment shows that applying Tabu Search instead of Local Search is a little bit better, see Fig. 7.

Fig. 4 Relative error distribution. Horizontal axis: relative error of the heuristic solver, vertical axis: number of the cases

Fig. 5 Impact of the ratio of stew pans and ticker pans on the relative error of the heuristic

Recall that we investigated 1000 instances. For each of them Tabu Search was performed with different lengths of the tabu list, as given in the picture (10, 30, 50, 70, 100). (The size of the tabu list was chosen to be 30 by Hurink et al.) Let us denote the heuristic algorithm applying LS or TS by Heu_{LS} or Heu_{TS} , respectively.

We experienced that for more than 850 cases the makespan given by Heu_{LS} and Heu_{TS} are the same. The remaining fewer than 150 instances are visualized in Fig. 7. Since we experienced that TS is not very sensitive to the size of the tabu list, let us

Fig. 6 Impact of the ratio of stew pans and ticker pans on the relative error of the heuristic—averaged by ranges

Fig. 7 Tabu Search versus Local Search

focus on the case when this size is 30. In 31 cases the result of Heu_{TS} is better by 1 unit. In another 29 cases the improvement of Heu_{TS} over Heu_{LS} is 2. In 19 cases the improvement is 3, in 5 cases the improvement is 4, while the maximal improvement is 5 which occurred in 2 cases. We must note that in some cases Heu_{LS} was better than Heu_{TS} (in 19 cases it is better by 1 unit, in 13 cases by 2 units and in 3 cases by 3). If the size of the tabu list is 10, the improvement of Heu_{LS} upon Heu_{TS} is 4 and it is 8 in one case when the size of the tabu list is 100.

We know that the method of Hurink et al. (1994) is very efficient for job-shop scheduling problems with multi-purpose machines. However, we have seen that this method is only a little bit more efficient than applying LS for improving the raw assignment in our problem. The reason may be that, as it was explained in Sect. 2.2, our problem differs significantly from the problem class investigated by Hurink et al.

We tried to detect what could be the reason for the fact that the heuristic is really very efficient in many of the cases and it is a little bit weak in some other cases. We think the reason of this phenomenon can be the following: the setup time was determined randomly among the 1000 instances, similarly, as it was for the other input, like the processing times of the operations. If for a certain (randomly generated) instance the setup time is “small” compared to the processing times, then one of our main simplifications assumes that only one setup time can be made on any machine: this is not a strong assumption. It still admits to reach an optimal or near-optimal solution. But in some other cases the setup time is negligible small compared to the processing times. In these cases it can happen that in the optimal schedule in Room2 the multi-

Fig. 8 Computation time of the heuristic-aided exact solver related to the relative error of the heuristics

purpose machine swaps several time between performing the different operations. In this case setup times will appear several times but the schedule is still better. The reason is that in the opposite case there is only one setup time but as a consequence some machine in Room3 will be idle for a long time as it does not get material from Room2. It is important to note that the Local Search keeps the property of the schedule that each machine swaps between the operation types at most once, but in case of Tabu Search this constraint is relaxed.

In Fig. 8 we illustrated the computation time of the exact solver related to the relative error of the heuristic. It has to be noted that we applied a 10,000 s time limit: the illustrated cases with 10,000 s mean that in those cases the exact solver did not find the optimum in 10,000 s.

We were interested in whether there is a connection between problems that are “hard” for the heuristic and problems “hard” for the solver. We consider the problem hard for the heuristic if the relative error of the heuristic is high, while we consider the problem hard for the solver if the running time is long. We concluded that we do not see a strong connection between these two factors. The dots in the left top part of the figure mean that even if the heuristic finds the optimal solution, to optimally solve the problem is hard for the solver.

7.2.2 Running times using the basic versus advanced MILP model

We made comparisons on the running time of MILP solver if we use the basic model or if we applied the advanced MILP model which is described in Sect. 6. Generally we experienced that the modifications are able to speed up the solution process by a factor of 3 on average. However, the advanced model is not always faster than the original one; and the speeding factor is not the same if we divide the 1000 instances into groups (Group 1–5) similarly, as we have made before (in Sect. 7.2.1 paragraph 2, see also Fig. 3). The details can be seen in Fig. 9.

There are instances for which the “improvement” is not really an improvement as the running time for the advanced model (RT_A) is significantly bigger than the running time of the original model (RT_O). For one such input $RT_A = 2280.58$ and $RT_O = 1408.63$ (all values are given in sec.). Taking another instance these two running times are $RT_A = 84.53$ and $RT_O = 296.33$. Note that both these instances are within Group 1.

Fig. 9 The average and the standard deviation of the improvement in the MILP model

On Fig. 9 part a. we show the average improvement in the different groups of instances. As we see, the improvement is bigger if the makespan is “not too small” and “not too big”, i.e., in cases Group 2 and Group 3. The standard deviation within the groups, that shows how the ratios of the running times differ from the average value in every group, can be seen in Fig. 9 part b.

7.2.3 Evaluation of the different settings of the solver

The exact MILP solver needs the horizon $\mathcal{T}$ as input value. Usually, when such a solver is applied, a feasible solution is soon found. The difference between the objective values of the feasible solution and the lower bound is called “gap”. We can stop the running if we are satisfied with the feasible solution (the gap is small enough) or we can enforce the solver to close completely the gap, i.e., do not stop until a proven optimal solution is found. During the following comparisons the original version of the MILP model was applied.

As described in Sect. 7.1, the following methods to set this value were analyzed:

- *M0*: using a simple computed coarse upper bound, the solver is stopped when the first feasible solution is found, then the upper bound is decreased one by one until there is not a feasible solution. It is natural, that it is better if instead of using the coarse upper bound we apply the upper bound provided by the heuristic.
- *M1*: using the heuristic as an upper bound and run the solver until optimality.
- *M2*: using the result of the heuristic decreased by 1, and run the solver until optimality. (If the optimal solution is not found, it means that the heuristic solution was optimal.) Figure 10 shows the correlation between the run of *M2* (i.e., heuristic minus 1) on the horizontal axis versus *M1* (i.e., heuristic) on the vertical axis, on a double-logarithmic scale. A good fit is the function $y = 1.1424 \cdot x^{0.9769}$. This function returns the average expected time y for the „heuristic run” (that is, Cplex using the heuristic as starter), given the time for the „heuristic minus 1” run as x . For $x < 300$, the „heuristic run” is slower, for $x > 300$ the „heuristic run” is faster. From the experiments we conclude that there is no significant difference.
- *M3*: using a lower bound and increasing it one by one until a feasible solution is found. In Fig. 11 the horizontal axis shows the solution of Cplex with the aid of heuristic (*M1*) and the vertical axis represents the lower bound (*M3*) runtime. Points above the red line mean that Cplex with the aid of heuristic beats the lower

Fig. 10 Computation time of the heuristic-aided exact solver with the heuristic result (M1) versus the heuristic result-1 (M2)

Fig. 11 Computation time of the Cplex started from the lower bound (M3) versus the Cplex aided by the heuristic (M1)

Fig. 12 Computation time of the logarithmic step size lower bound method (M4) related to the one-by-one increasing lower bound method (M3)

bound method, and points below the red line are those where the lower bound based method is better.

- *M4*: Instead of increasing the horizon one by one starting from the lower bound and stop when a feasible solution is found, we also investigated the option of increasing by a logarithmic step size, i.e., taking into account the heuristic result as an upper bound. Figure 12 shows the results (of *M4*) compared to the simple one-by-one increasing method (*M3*) and Fig. 13 illustrates the results of *M4* compared to the heuristic-based upper bound strategy (*M1*). As one could expect, the logarithmic search beats any of these two previous approaches.

7.3 Instances of the special case

We generated 1000 instances. One of them is presented here in detail, see Table 7. The other 999 are generated randomly by drawing uniformly distributed numbers from certain given intervals.

7.3.1 Evaluation of the results using commercial plant simulation software

Siemens Tecnomatix Plant Simulation is a commercial software that is an industrial standard for the simulation of production processes. It also supports the visualization

Fig. 13 Computation time of the logarithmic step size lower bound method (M4) related to the heuristic based upper bound method (M1)

Table 7 Operation time for (resource, task)-pairs

	A_1	A_2	A_3	A_4
R_1	–	5	11	–
R_2	–	5	11	–
R_3	–	5	11	–
R_4	–	12	5	–
R_5	2	–	–	–
R_6	2	–	–	–
R_7	–	–	–	4
R_8	–	–	–	4

$$u_{R_1} = u_{R_2} = u_{R_3} = 2; u_{R_4} = 3; n^{\text{stew}} = 60; n^{\text{ticker}} = 40$$

of systems in 2D and 3D. It enables to study the flow of materials and the utilization of machines and buffers within a production line (Bangsow, 2015; Klos and Trebuna, 2015; Debevec et al., 2014). We modeled the production line described in Sect. 2.1 using this simulation environment. The graphical representation of our production line is depicted in Fig. 14. The data describing the resources in the different rooms from Table 6 are also used here.

In comparison to the optimization approach, there was a random instance for what the simulation produced a schedule with makespan 430, whereas the optimal solution has makespan 308.

8 Conclusion

In this paper we considered a complex scheduling problem with unrelated machines. We proposed an efficient heuristic method, which is able to find an optimal solution in many cases and runs very quickly. On the other hand, we also solved the problem with an exact solver which first needs an estimation of the optimal makespan, always provides an optimal solution but requires much more computational effort. Our conclusion is that the smart combination of the heuristic method and the exact solver is the most effective. While this is the case for many optimization problems, the proposed heuristic is far from trivial. We first divided the problem into smaller sub-problems. Some of them can be solved easily, while one sub-problem is still hard. Here first we create a preemptive solution, then apply a rounding procedure and finally a Local Search or Tabu Search.

We have detected several points where our algorithm could be improved.

We think that the solution provided by our method works quite well locally in any of the three rooms. In fact, in Room1 we can generate an optimal solution (locally, only considering this Room alone). Also, the solution provided in Room3 is also (locally) almost optimal, if we get the products from the previous two rooms as input. The solution for Room2 is also acceptable, close to be locally optimal. The problem is the combination of the three locally optimal or almost optimal solutions to one global solution. We think that the main difficulty of the problem arises from the possible alternating operation modes of the multi-purpose machines, as described in the following.

Recall, that we made a list of simplifications in Room2 to be able to handle the problem easily. A main simplification is that for avoiding many setup times, we allowed only at most one setup time for any machine. It means that any A2 operation is performed before any A3 operation is made by multi-purpose machines. If we keep this simplification, the heuristic cannot be made significantly better (and let us note that the heuristic works quite well in fact). If we would like to get a still better performance by some heuristic, we must set aside this mentioned property, i.e., we must allow several setup times per machine. Then the whole heuristic must be changed. Recall, that our goal in this paper was not to solve as well as possible a real problem, but as we described it, we wanted to show how a heuristic solution can help a solver, with a list of tricky inventions building on the special features of a system. For this purpose we believe that our heuristic algorithm is suitable.

A commercial simulation software was also applied showing that for this problem our special treatment is much more efficient than the general simulation tool.

Our computational experiments show that the combined method (heuristic + exact solver) is really efficient and fast. The most effective combination was the logarithmic search based application of the heuristic. A challenge for the future is to examine more general cases (with an arbitrary number of resources and with different process

structure). Another interesting question for further research is the investigation of the separability of complex workflows into smaller sub-problems.

Acknowledgements Open access funding provided by University of Pannonia (PE). Tibor Dulai, György Dósa and Ágnes Werner-Stark acknowledge the financial support of Széchenyi 2020 under the EFOP-3.6.1-16-2016-00015, Armin Fügenschuh acknowledges the financial support of DFG Grant FU860/1-1 and Dósa was also supported by the National Research, Development and Innovation Office – NKFIH under the Grant SNN 129364. Peter Auer, György Dósa, Tibor Dulai, Ronald Ortner and Ágnes Werner-Stark are supported by Stiftung Aktion Österreich-Ungarn 99öu1. Moreover, we would like to thank the valuable comments and proposals of the two anonymous referees which helped us in the presentation of our paper.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Almeder C, Klabjan D, Traxler R, Almada-Lobo B (2015) Lead time considerations for the multi-level capacitated lot-sizing problem. *Eur J Oper Res* 241(3):727–738
- Bangsow S (2015) Tecnomatix plant simulation. Springer, Cham
- Brandimarte P (1993) Routing and scheduling in a flexible job shop by tabu search. *Ann Oper Res* 41:157–183. <https://doi.org/10.1007/BF02023073>
- Chen B, Potts C, Woeginger G (1998) A review of machine scheduling: complexity, algorithms and approximability. In: Du D-Z, Pardalos P (eds) *Handbook of combinatorial optimization*, vol 3. Kluwer Academic Publishers, Dordrecht
- Correa JR, Schulz AS (2005) Single-machine scheduling with precedence constraints. *Math Oper Res* 30(4):1005–1021
- Debevec M, Simic M, Herakovic N (2014) Virtual factory as an advanced approach for production process optimization. *Int J Simul Model* 13(1):66–78
- Fattahi P, Saidi-Mehrabad M, Jolai F (2007) Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. *J Intell Manuf* 18:331–342. <https://doi.org/10.1007/s10845-007-0026-8>
- Floudas C, Lin X (2005) Mixed integer linear programming in process scheduling: modeling, algorithms, and applications. *Annals OR* 139:131–162. <https://doi.org/10.1007/s10479-005-3446-x>
- Gicquel C, Minoux M (2015) Multi-product valid inequalities for the discrete lot-sizing and scheduling problem. *Comput Oper Res* 54:12–20
- Gicquel C, Minoux M, Dallery Y (2009) On the discrete lot-sizing and scheduling problem with sequence-dependent changeover times. *Oper Res Lett* 37:32–36
- Hassan M, Kacem I, Martin S, Osman IM (2016) Unrelated parallel machine scheduling problem with precedence constraints: polyhedral analysis and branch-and-cut. In: Cerulli R, Fujishige S, Mahjoub A (eds) *Combinatorial optimization ISCO 2016 Lecture Notes in Computer Science*, vol 9849, pp 308–319
- Herrmann J, Proth J, Sauer N (1997) Heuristics for unrelated machine scheduling with precedence constraints. *Eur J Oper Res* 102(3):528–537
- Hurink J, Jurisch B, Thole M (1994) Tabu search for the job-shop scheduling problem with multi-purpose machines. *Oper-Res-Spektrum* 15:205–215
- Klos S, Trebuna P (2015) Using computer simulation method to improve throughput of production systems by buffers and workers allocation. *Manag Prod Eng Rev* 6(4):60–69
- Ku WY, Beck C (2016) Mixed integer programming models for job shop scheduling: a computational analysis. *Comput Oper Res* 73:165–173

- Kumar VSA, Marathe MV, Parthasarathy S, Srinivasan A (2005) Scheduling on unrelated machines under tree-like precedence constraints. In: Chekuri C et al (eds) APPROX and RANDOM 2005, LNCS, vol 3624, pp 146–157
- Liu C, Yang S (2011) A heuristic serial schedule algorithm for unrelated parallel machine scheduling with precedence constraints. *J Softw* 6(6):1146–1153
- Najid N, Dauzè-Pérès S, Zaidat A (2002) A modified simulated annealing method for flexible job shop scheduling problem, vol 5, p 6. <https://doi.org/10.1109/ICSMC.2002.1176334>
- Pezzella F, Morganti G, Ciaschetti G (2008) A genetic algorithm for the flexible job-shop scheduling problem. *Comput Oper Res* 35(10):3202–3212. <https://doi.org/10.1016/j.cor.2007.02.014>
- Pochet Y, Wolsey LA (2006) Production planning by mixed integer programming. Springer, New York
- Rocha PL, Ravetti MG, Mateus GR, Pardalos PM (2008) Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine-dependent setup times. *Comput Oper Res* 35:1250–1264
- Saidi-Mehrabad M, Fattahi P (2007) Flexible job shop scheduling with tabu search algorithms. *Int J Adv Manuf Technol* 32:563–570. <https://doi.org/10.1007/s00170-005-0375-4>
- Sawik T (1999) Production planning and scheduling in flexible assembly systems. Springer, Berlin
- Wan X, Yan H (2016) An algorithm for assembly job shop scheduling problem. In: 2nd International Conference on Electronics, Network and Computer Engineering (ICENCE 2016). Atlantis Press. <https://doi.org/10.2991/icence-16.2016.48>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.