Brandenburg
University of Technology
Cottbus - Senftenberg

A New Heuristic and an Exact Approach for a Production Planning Problem

Peter Auer
György Dósa
Tibor Dulai
Armin Fügenschuh
Peggy Näser
Ronald Ortner
Ágnes Werner-Stark

Cottbus Mathematical Preprints
COMP # 5(2019)

A New Heuristic and an Exact Approach for a Production Planning Problem

Peter Auer[*] György Dósa[†] Tibor Dulai[‡] Armin Fügenschuh[§]
Peggy Näser[¶] Ronald Ortner[‖] Ágnes Werner-Stark[**]

April 23, 2019

Abstract

We deal with a very complex and hard scheduling problem. Several 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 goal is to process all products in minimum time, i.e., the makespan is to be minimized. 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.

1 Introduction

We consider a production planning problem for a 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.

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),

[*]Montanuniversität Leoben
[†]University of Pannonia
[‡]University of Pannonia, H-8200 Veszprém, Egyetem str. 10., Hungary, dulai.tibor@virt.uni-pannon.hu
[§]Brandenburgische Technische Universität Cottbus-Senftenberg
[¶]Brandenburgische Technische Universität Cottbus-Senftenberg
[‖]Montanuniversität Leoben
[**]University of Pannonia

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), Seeanner (2013) and Voß and Woodruff (2006). 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.

Jans and Degraeve (2007) review various meta-heuristic methods for solving different kinds of lot sizing models. A classical work in this topic is Wagner and Whitin (1958). Generally in these models there is only one resource, but the model can be extended by different constraints. Several papers consider very special models, such as Terrazas-Moreno et al. (2012).

We note that the production planning 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.

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.

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 $P_1, P_2, \ldots, P_N$) are given that have to be produced with cardinalities $n_1, n_2, \ldots, n_N$. Any product is produced by executing several activities. There are B kinds of activities, also called tasks, denoted by $A_\alpha, \alpha = 1, \ldots, B$. 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, $j = 1, \ldots, N$. 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 $R_1, \ldots, R_r$ 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.

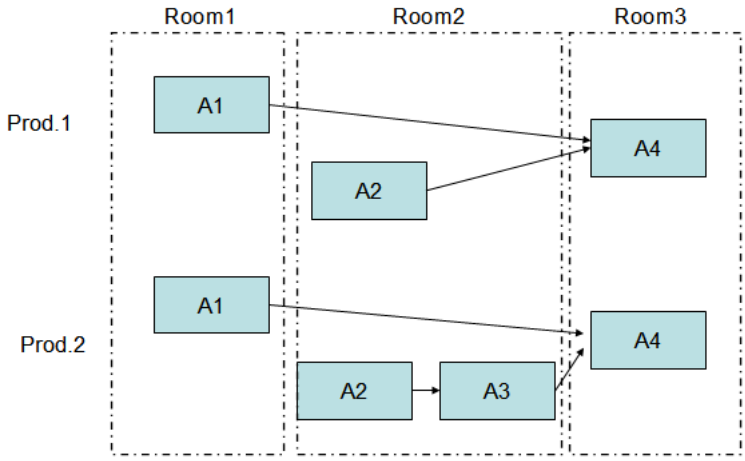

Figure 1: processes of the two products

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 "stewpan", the second a "tickerpan". We assume that n^{stew} stewpans and n^{ticker} tickerpans have to be produced. A stewpan consists of a tiller part and a head part (we use the name "can" for the head part). A tickerpan 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 Figure 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 tickerpans there is one more operation, namely task A_3, which is the punching of the can (this operation is not needed for stewpans). 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 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).

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

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)).

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 1: Parameters of the model.

name	description
m^{tiller}	tiller raw material input
m^{can}	can raw material input
n^{stew}	stewpan output
n^{ticker}	tickerpan output

Table 2: Production input and output parameters.

3.1 Sets and Parameters

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

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 $p_{R_i}^{\text{extru}}, i = 1...4$, three of which are similar, and the fourth is different, cf. Table 5. 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, stewpans and tickerpans, see Table 2. In Section 6 we discuss 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 two categories: stocks and flows. 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 step to time step. The details can be found in Tables 3 and 4. Since the objective is to minimize the total makespan, we introduce a further integer variable $M \in \mathbb{Z}_+$ for the makespan.

name	range	description
$s^{\text{tillerraw}}(t)$	$\mathbb{Z}_+$	tiller raw material at time t
$s^{\text{tiller}}(t)$	$\mathbb{Z}_+$	finished tillers at time t
$s^{\text{canraw}}(t)$	$\mathbb{Z}_+$	can raw material at time t
$s^{\text{extruded}}(t)$	$\mathbb{Z}_+$	extruded cans at time t
$s^{\text{punched}}(t)$	$\mathbb{Z}_+$	punched cans at time t
$s^{\text{stew}}(t)$	$\mathbb{Z}_+$	finished stewpans at time t
$s^{\text{ticker}}(t)$	$\mathbb{Z}_+$	finished tickerpans at time t

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

name	i	range	description
$f_{R_i}(t)$	5&6	$\{0,1\}$	R_i starts processing at time t
$f_{R_i}^{\text{extru}}(t)$	1–4	$\{0,1\}$	R_i starts extruding at time t
$f_{R_i}^{\text{punch}}(t)$	1–4	$\{0,1\}$	R_i starts punching at time t
$f_{R_i}^{\text{extruded}}(t)$	7&8	$\{0,1\}$	R_i starts finishing stewpan at time t
$f_{R_i}^{\text{punched}}(t)$	7&8	$\{0,1\}$	R_i starts finishing tickerpan at time t
$f_{R_i}^{\text{tiller}}(t)$	7&8	$\{0,1\}$	R_i starts finishing any pan with tiller at time t

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

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:

$$s^{\text{tillerraw}}(0) = m^{\text{tiller}},$$
$$s^{\text{tiller}}(0) = 0,$$

which places all tiller raw material in front of R_5 and R_6 and assumes an empty stock of finished tillers at time step 0. The boundary conditions enforce no production in the very last time step 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 step t is moved into the two resources R_5, R_6, and the remaining tillers are moved to the next time step:

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

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:

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

3.3.2 Room2: Extrusion and Punching

In Room2, the can raw material is extruded on one of the resources $R_1, \ldots, R_4$. 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 $R_1, \ldots, R_4$. As initial condition, all can raw material is placed in front of machines $R_1, \ldots, R_4$ at time step 0:

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

There is no possibility for punching at time step 0:

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

There is an empty stock of extruded and punched cans at time step 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 step T:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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 stewpans or tickerpans. As initial conditions, we assume that the stock of stewpans and tickerpans 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 step:

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

The final stocks of stewpans and tillerpans must meet the required demand:

$$s^{\text{stew}}(T) = n^{\text{stew}},$$

$$s^{\text{ticker}}(T) = n^{\text{ticker}}.$$

The stock of stewpans 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 tickerpans 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:

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

and our objective is to minimize the total makespan M.

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 Section 6, 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".

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 stewpans and tickerpans 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 (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

[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.

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 $i \in \{0, 1, \ldots, k_1\}$ 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.

$$\alpha \cdot y + \beta \cdot v = n_1 + n_2 \tag{1}$$
$$\beta \cdot w + \gamma \cdot x = n_2 \tag{2}$$
$$\gamma \cdot \epsilon = v \cdot \epsilon + w \cdot \theta + \zeta = x \cdot \eta + \epsilon \tag{3}$$

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 equations (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 Local Search Added

After we get a rough assignment for the subproblem of Room2, we try to improve it by local search using 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. There are two ways for managing the local changes. We can do it in a directed way, i.e., we choose a machine with maximum completion time, and from this machine we move an operation to some other machine. An alternative is to choose the changes randomly.

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.

5.5 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 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 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 Subsection 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 Computational Experiments

6.1 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:

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

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 $\mathcal{T}^{\mathrm{heur}} := M^*$, 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 $\mathcal{T}^{\mathrm{ub2}} := M^* - 1$, 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:

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

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 1,000 randomly generated instances, where the parameters are chosen according to a uniform distribution over the integers in the intervals shown in Table 5. 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.

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]	$n^{\text{ticker}} = 100 - n^{\text{stew}}$

Table 5: Intervals for random data.

6.2 Evaluation of the Results

6.2.1 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).

The computation time for the heuristic was quick: two minutes for the 1,000 instances (together with the resulting text file generation), which means 0.12 seconds for one instance in average. The calculation of the exact solution usually takes more time: there were cases when 10,000 seconds were not enough for one instance.

Figure 3 presents the relative error distribution where the makespan of the heuristic solution is compared to the optimal solution. One can see that about in half of the cases the heuristic found the optimum, 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. Based on Figure 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 305–365 and instances 685–889). 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 304, part 2 is in the middle, where the indexes are between 366 and 684, while the third part is where the indexes are bigger than 889. 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. 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.

In Figure 4 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 seconds time limit: the illustrated cases with 10,000 seconds mean that in those cases the exact solver did not find the optimum in 10,000 seconds.

6.2.2 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.

As described in Subsection 6.1, 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

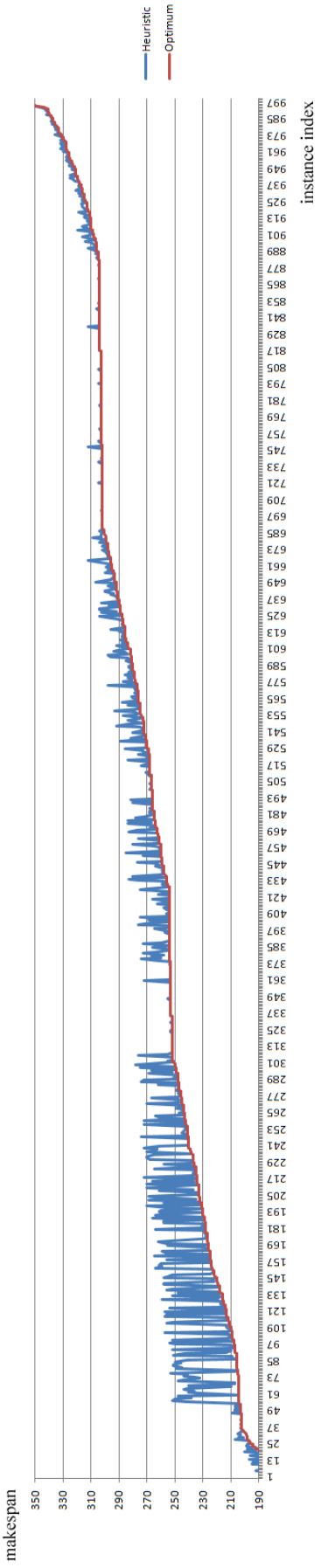

Figure 2: Makespan via heuristic and exact solvers.

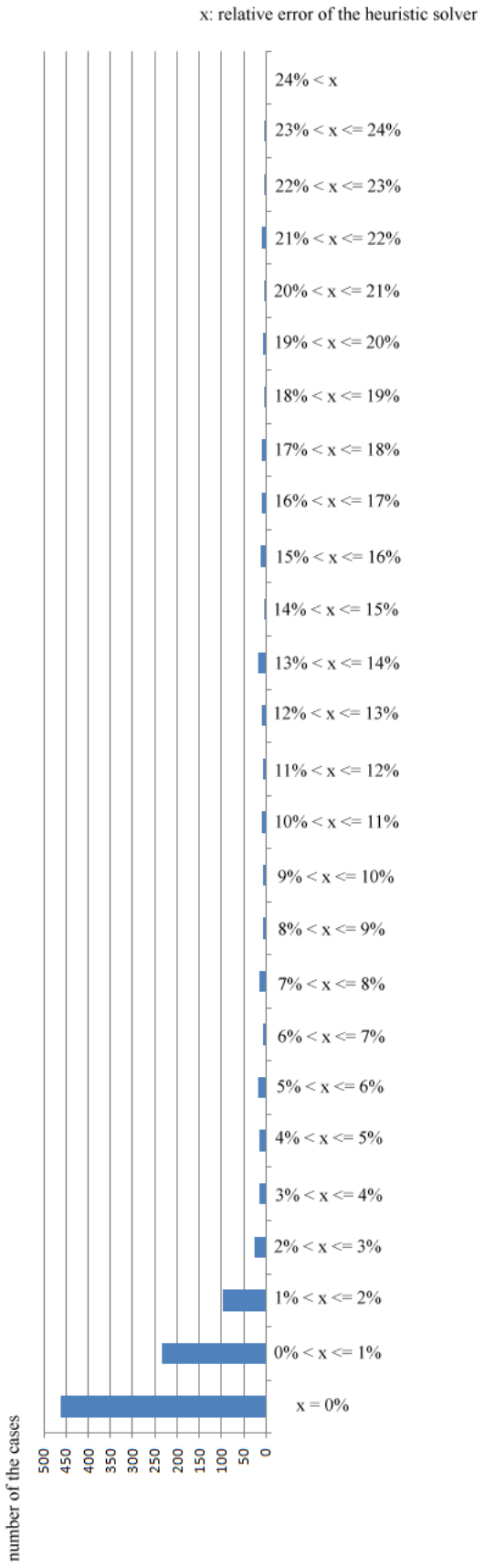

Figure 3: Relative error distribution.

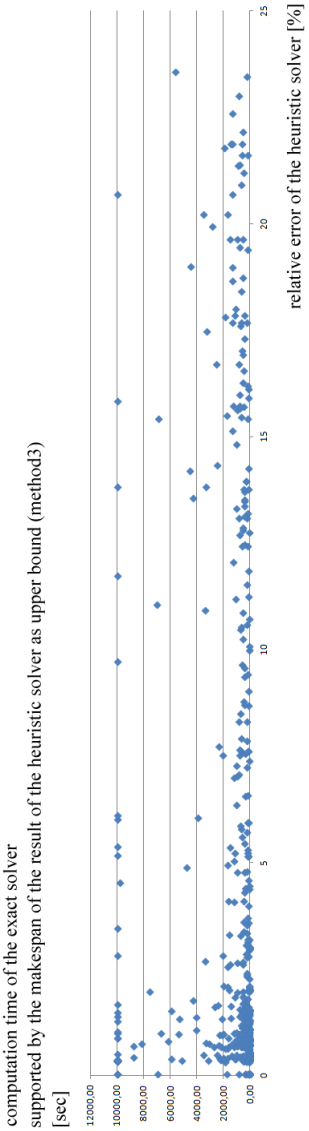

Figure 4: Computation time of the heuristic-aided exact solver related to the relative error of the heuristics.

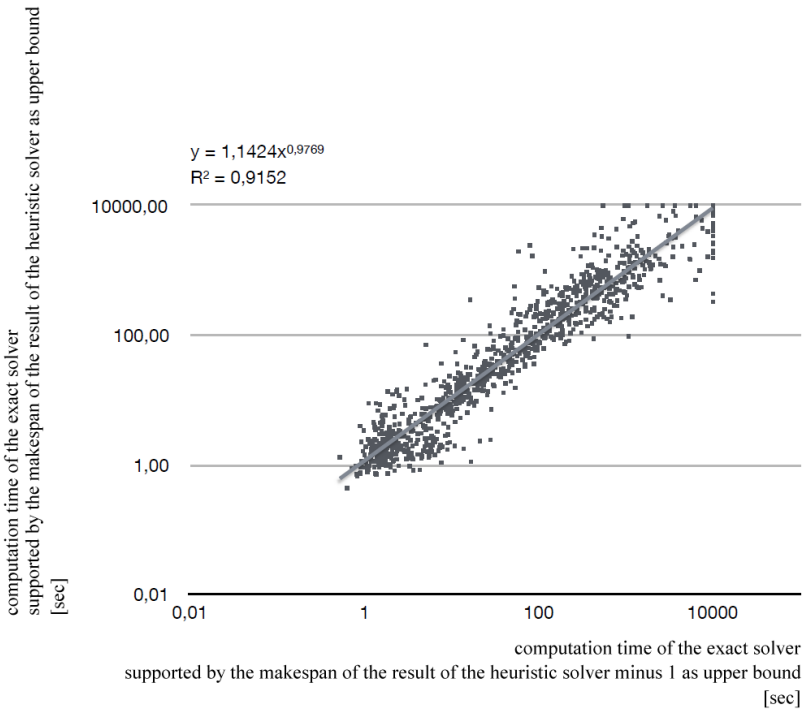

Figure 5: Computation time of the heuristic-aided exact solver with the heuristic result vs. the heuristic result−1.

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 5 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 Figure 6 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 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 7 shows the results (of $M4$) compared to the simple one-by-one increasing method ($M3$) and Figure 8 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.

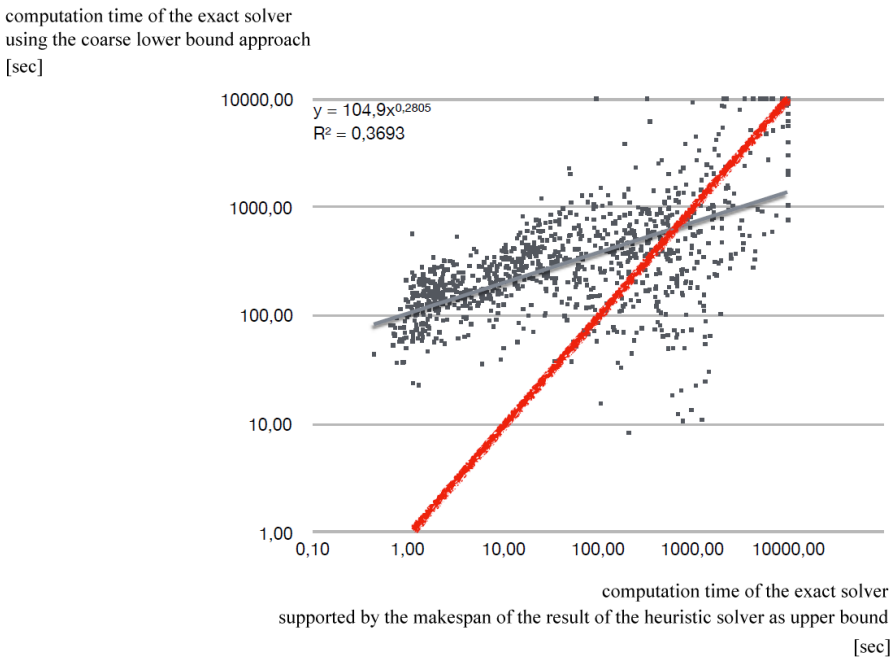

Figure 6: Computation time of the Cplex started from the lower bound vs. the Cplex aided by the heuristic.

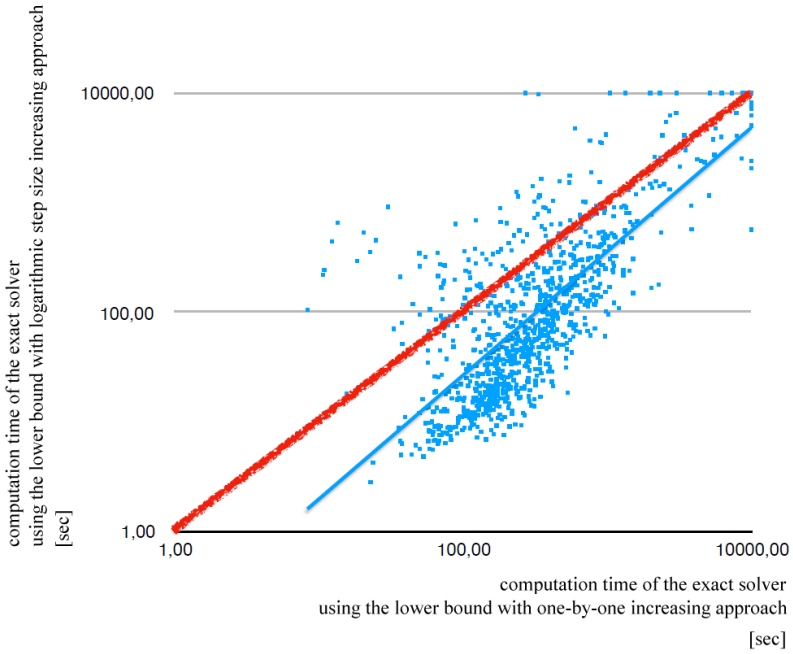

Figure 7: Computation time of the logarithmic step size lower bound method related to the one-by-one increasing lower bound method.

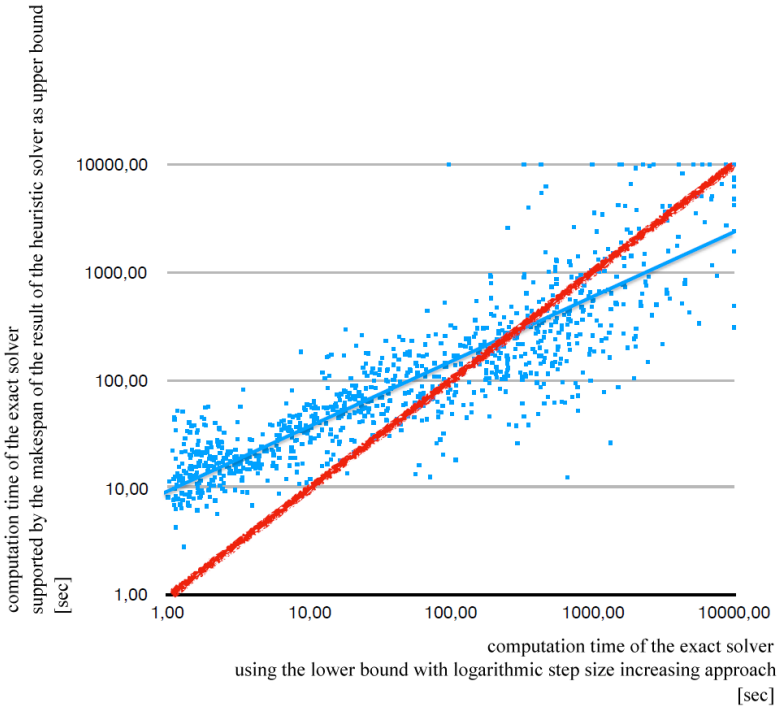

Figure 8: Computation time of the logarithmic step size lower bound method related to the heuristic based upper bound method.

	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

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

Table 6: Operation time for (resource, task)-pairs.

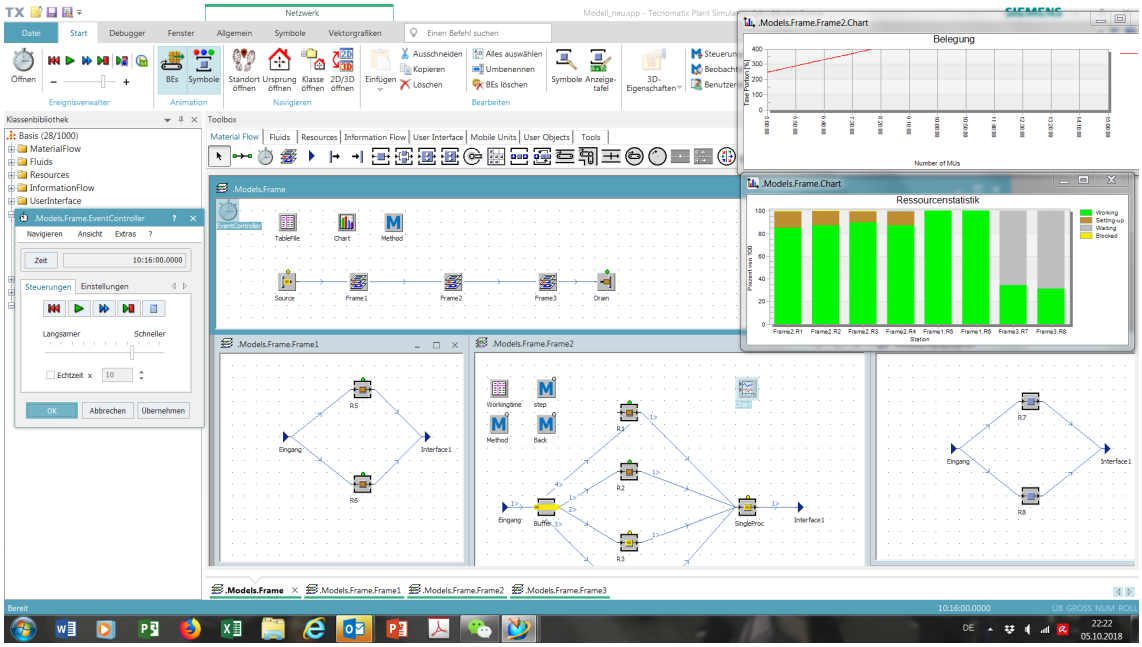

Figure 9: Screenshot from Siemens Tecnomatix Plant Simulation.

6.3 Instances of the Special Case

We generated 1,000 instances. One of them is presented here in detail, see Table 6. The other 999 are generated randomly by drawing uniformly distributed numbers from certain given intervals.

6.3.1 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 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 Section 2.1 using this simulation environment. The graphical representation of our production line is depicted in Figure 9. The data describing the resources in the different rooms from Table 5 are also used here.

Being a simulation tool, it is necessary to set up a certain rule for distributing incoming raw material in Room1, Room2, and Room3 to the respective resources. Hereto, the tool offers a list of possible settings, which can be endowed with further parameters. As an example, one of the settings is a proportional distribution of the raw material using a user-defined percentage for each parallel working resource. During our investigations, we experimented with various strategies to find out which one leads to the smallest overall makespan.

The resulting outcome as a statistic over the 8 resources is shown in Figure 10. It shows that

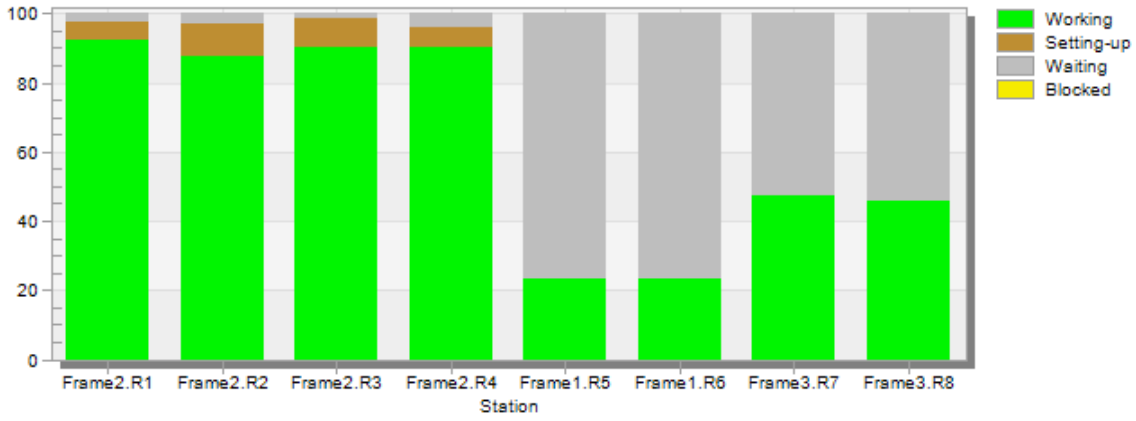

Figure 10: Resource statistics for the eight resources. Each column adds up to 100%.

the production bottleneck is in Room2, where all four resources are working almost all the time with only very little idle time (which actually is at the very beginning of the time horizon, when the pipeline is still empty). The resources in Room1 and Room3 are not in use the whole time, here a lot of idle times accumulate.

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. This shows that such problems should not be solved with simulation tools and experienced planners alone. In fact, there is a huge benefit that only a proper optimization routine can make accessible.

7 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. 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

Tibor Dulai, György Dósa and Ágnes Werner-Stark acknowledge the financial support of Széchenyi 2020 under the EFOP-3.6.1-16-2016-00015 and Armin Fügenschuh acknowledges the financial support of DFG grant FU860/1-1.

References

Almeder C, Klabjan D, Traxler R, Almada-Lobo B (2015) Lead time considerations for the multi-level capacitated lot-sizing problem. European Journal of Operational Research 241(3):727–738

Bangsow S (2015) Tecnomatix Plant Simulation. Springer International Publishing Switzerland

Brandimarte P (1993) Routing and scheduling in a flexible job shop by tabu search. Annals of Operations Research 41:157–183, DOI 10.1007/BF02023073

Chen B, Potts C, Woeginger G (1998) A review of machine scheduling: Complexity, algorithms and approximability, Handbook of Combinatorial Optimization (Vol 3). Kluwer Academic Publishers

Correa JR, Schulz AS (2005) Single-Machine Scheduling with Precedence Constraints. Mathematics of Operations Research 30(4):1005–1021

Debevec M, Simic M, Herakovic N (2014) Virtual Factory as an Advanced Approach for Production Process Optimization. International Journal of Simulation Modelling 13(1):66–78

Fattahi P, Saidi-Mehrabad M, Jolai F (2007) Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. Journal of Intelligent Manufacturing 18:331–342, DOI 10.1007/s10845-007-0026-8

Floudas C, Lin X (2005) Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications. Annals of Operations Research 139:131–162, DOI 10.1007/s10479-005-3446-x

Gicquel C, Minoux M (2015) Multi-product valid inequalities for the discrete lot-sizing and scheduling problem. Computers and Operations Research 54:12–20

Gicquel C, Minoux M, Dallery Y (2009) On the discrete lot-sizing and scheduling problem with sequence-dependent changeover times. Operations Research Letters 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 9849:308–319

Herrmann J, Proth J, Sauer N (1997) Heuristics for unrelated machine scheduling with precedence constraints. European Journal of Operational Research 102(3):528–537

Jans R, Degraeve Z (2007) Meta-heuristics for dynamic lot sizing: A review and comparison of solution approaches. European Journal of Operational Research 177:1855–1875

Klos S, Trebuna P (2015) Using Computer Simulation Method to Improve Throughput of Production Systems by Buffers and Workers Allocation. Management and Production Engineering Review 6(4):60–69

Ku WY, Beck C (2016) Mixed Integer Programming models for job shop scheduling: A computational analysis. Computers and Operations Research 73:165–173

Kumar VSA, Marathe MV, Parthasarathy S, Srinivasan A (2005) Scheduling on Unrelated Machines Under Tree-Like Precedence Constraints. C Chekuri et al (Eds): APPROX and RANDOM 2005, LNCS 3624:146–157

Liu C, Yang S (2011) A heuristic serial schedule algorithm for unrelated parallel machine scheduling with precedence constraints. Journal of Software 6(6):1146–1153

Najid NM, Dauzere-Peres S, Zaidat A (2002) A modified simulated annealing method for flexible job shop scheduling problem. In: IEEE International Conference on Systems, Man and Cybernetics, vol 5, p 6, DOI 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, DOI 10.1016/j.cor.2007.02.014

Pochet Y, Wolsey LA (2006) Production Planning by Mixed Integer Programming. Springer Verlag, New York, USA

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. Computers and Operations Research 35:1250–1264

Saidi-Mehrabad M, Fattahi P (2007) Flexible job shop scheduling with tabu search algorithms. The International Journal of Advanced Manufacturing Technology 32:563–570, DOI 10.1007/s00170-005-0375-4

Sawik T (1999) Production Planning and Scheduling in Flexible Assembly Systems. Springer-Verlag Berlin, Heidelberg, New York

Seeanner F (2013) Multi-Stage Simultaneous Lot-Sizing and Scheduling. Springer Fachmedien, Wiesbaden

Terrazas-Moreno S, Grossmann IE, Wassick JM (2012) A Mixed-Integer Linear Programming Model for Optimizing the Scheduling and Assignment of Tank Farm Operations. Industrial & Engineering Chemistry Research 51:6441–6454

Voß S, Woodruff DL (2006) Introduction to Computational Optimization Models for Production Planning in a Supply Chain. Springer-Verlag Berlin, Heidelberg

Wagner HM, Whitin TM (1958) Dynamic Version of the Economic Lot Size Model. Management Science 5(1):89–96

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, DOI 10.2991/icence-16.2016.48

IMPRESSUM

Brandenburgische Technische Universität Cottbus-Senftenberg
Fakultät 1 | MINT - Mathematik, Informatik, Physik, Elektro- und Informationstechnik
Institut für Mathematik
Platz der Deutschen Einheit 1
D-03046 Cottbus

Professur für Ingenieurmathematik und Numerik der Optimierung
Professor Dr. rer. nat. Armin Fügenschuh

E fuegenschuh@b-tu.de
T +49 (0)355 69 3127
F +49 (0)355 69 2307

Cottbus Mathematical Preprints (COMP), ISSN (Print) 2627-4019
Cottbus Mathematical Preprints (COMP), ISSN (Online) 2627-6100

www.b-tu.de/cottbus-mathematical-preprints
cottbus-mathematical-preprints@b-tu.de
doi.org/10.26127/btuopen-4827