Scheduling Methods for Efficient Stamping Operations at an Automotive Company

Burcu Caglar Gencosman
Industrial Engineering Department, Uludag University, Bursa, Turkey

Mehmet A. Begen
Ivey Business School, Western University, London, Canada, mbegen@ivey.uwo.ca http://mehmet.begen.net

H. Cenk Ozmutlu
Industrial Engineering Department, Uludag University, Bursa, Turkey

Imren Ozturk Yilmaz
Beycelik Gestamp Bursa, Turkey

We consider scheduling issues at Beyçelik, a Turkish automotive stamping company that uses presses to give shape to metal sheets in order to produce auto parts. The problem concerns the minimization of the total completion time of job orders (i.e., makespan) during a planning horizon. This problem may be classified as a combined generalized flowshop and flexible flowshop problem with special characteristics. We show that the Stamping Scheduling Problem is NP-Hard. We develop an integer programming-based method to build realistic and usable schedules. Our results show that the proposed method is able to find higher quality schedules (i.e., shorter makespan values) than both the company's current process and a model from the literature. However, the proposed method has a relatively long run time, which is not practical for the company in situations when a (new) schedule is needed quickly (e.g., when there is a machine breakdown or a rush order). To improve the solution time, we develop a second method that is inspired by decomposition. We show that the second method provides higher-quality solutions - and in most cases optimal solutions - in a shorter time. We compare the performance of all three methods with the company's schedules. The second method finds a solution in minutes compared to Beyçelik's current process, which takes 28 hours. Further, the makespan values of the second method are about 6.1% shorter than the company's schedules. We estimate that the company can save over €187,000 annually by using the second method. We believe that the models and methods developed in this paper can be used in similar companies and industries.

Keywords: stamping scheduling; machine scheduling; flowshop scheduling; integer programming; decomposition

1. Introduction

When it comes to production assignments and planning, efficient scheduling represents an indispensable tool for automotive stamping companies, among other industries. We consider a real-world Stamping Scheduling Problem at an automotive stamping company, Beyçelik Gestamp in Bursa, Turkey. The company produces automotive parts such as roofs, doors, bumpers, and axles for automotive companies such as Fiat, Ford, Renault, Volkswagen, and Maserati. These parts

are produced in a stamping pressline that includes 13 sequential presses (machines) with different weights, pressure levels, and sizes.

The production of each part involves a different number of operations that must be performed in consecutive machines in order for the final product to meet the necessary requirements. Each operation requires a unique upper and lower *die pair*. The production process starts with loading die pairs onto the machines. The raw material, a steel *blank*, moves through the pressline and is shaped by the dies. Once all operations have been completed, the blank emerges as a final product.

It is possible to produce different parts simultaneously as long as the total number of operations is less than or equal to the total number of machines. For example, three different parts, which require five, three, and five operations (respectively), can be produced in Beyçelik's stamping pressline. As seen in Figure 1, the first operation of part A (A.1) starts at machine 1 (M1), and the blank moves through the pressline to machine 5 (M5). At the end of the processing machine 5 (M5), all operations are complete for part A, and the final product can be sent to inventory.

Figure 1 Loading 3 different products to machines

Machines	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13
Operations	A.1	A.2	A.3	A.4	A.5	B.1	B.2	B.3	C.1	C.2	C.3	C.4	C.5
Final Products	Product A					Product B			Product C				

Due to the weight, size, and pressure differences across the various machines, a given job cannot be produced at any random machine (known as machine eligibility restrictions (Pinedo 2012)). For example, a more powerful (in terms of weight and pressure) and bigger machine can handle a job that requires less power and a smaller size, but not the opposite. Furthermore, the operations assignment of each job must obey machine and die-pair consistency (alternative machine sets). Moreover, due to the high cost of die pairs, the company has only one set of die pairs for each operation, and, thus, a job and its operations can be produced at only one machine set at a time.

Formally, we define the Stamping Scheduling Problem as follows: The machines $(M1, M2, \ldots, M12, M13)$ are arranged linearly as in a flowshop. There are many jobs, and each job requires a certain number of operations on consecutive machines. There are alternative machine sets for each job's operations. The objective is to minimize the makespan of processing all jobs. On one hand, we can think of this problem as a generalized flowshop (Mastrolilli and Svensson 2011) or a flowshop with jumps (Mastrolilli and Svensson 2009) since a job does not need to be processed on all machines. On the other hand, since we have alternative machine sets for each job, the problem can also be considered a flexible flowshop (Behnke and Geiger 2012). We propose

that the Stamping Scheduling Problem can be considered a combined generalized flowshop and flexible flowshop problem in which each job is processed consecutively on a set of machines and there are alternative machine sets for each job. Another way to think about the problem is that after the assignment of jobs to machines, the problem becomes a generalized flowshop; i.e., we need to consider both the assignment and sequencing problems.

Flowshop problems are studied extensively in literature both theoretically and in terms of applications (Garey et al. 1976, Gonzalez and Sahni 1978, Lawler et al. 1993, Waldherr and Knust 2015). Most of the flowshop problems with an objective of minimizing makespan, including generalized flowshop (Mastrolilli and Svensson 2011) and flexible flowshop (Behnke and Geiger 2012) are NP-hard. Even some of the special cases in flowshop problems are NP-hard (see, e.g., Chen et al. (1998)). Unsurprisingly, we show that our Stamping Scheduling Problem is NP-Hard. If the number of machines in the input is a variable then the Stamping Scheduling Problem is strongly NP-hard (Theorem 1). On the other hand, if the the number of machines is a fixed number (as in most practical scenarios, like the one we have in this paper) then the Stamping Scheduling Problem is NP-hard in the ordinary sense (Theorem 2). This latter result does not rule out the possibility of solving the scheduling problem using a pseudo-polynomial time algorithm.

Besides the scheduling challenge, the company must deal with ongoing and sudden difficulties, such as demand fluctuations, machine breakdowns, and die changeovers, in response to which the company may need to reschedule its operations. Therefore, developing a fast, feasible, and effective schedule, especially when rescheduling is needed, becomes a crucial requirement for the company.

To the best of our knowledge, this Stamping Scheduling Problem has not been investigated, except in Caglar Gencosman et al. (2014). As a first attempt, and to investigate the current practices of the company they were studying, these authors built two models: 1) a mixed integer programming model and 2) a constraint programming model (CP1). Their mixed integer program was too slow and could not resolve the real instances of the scheduling problem in the given solution time limit. Although CP1 was better able to generate feasible schedules, it had no optimality (gap) information. (We provide the CP1 model in Appendix A.)

We develop an integer programming model (IP1) to improve the quality of the CP1 solutions (i.e., to reduce the makespan values of CP1), and we provide some information on optimality (gap). Furthermore, to speed up IP1's solution time, we develop a new solution method, inspired by decomposition techniques. Our method employs a slightly modified version of IP1 and a simpler integer program IP2, which we call the IP2/IP1 method.

Decomposition methods divide the main problem into a master problem and a set of sub-problems and then solve them iteratively until the stopping criteria based on lower bound and upper bound is met. We divide the Stamping Scheduling Problem into two parts: a job-machine assignment problem (master problem) and a sequencing problem (sub-problem). The IP2/IP1 method solves the master problem with IP2, which is developed to solve the assignment problem, and it solves the sequencing sub-problem with the slightly modified version of IP1. IP2/IP1 has a faster solution time with shorter makespan values compared to IP1, CP1, and the company's schedules. IP2/IP1 solutions are optimal in most of our experiments.

We evaluate the performance of IP2/IP1 by using a randomly generated dataset and compare it to other methods as well as with the company's historical schedules. We use the empirical probability distribution of real production parameters to generate a dataset and compare the performance of IP2/IP1 with IP1 and CP1. The IP2/IP1 method finds shorter makespan (higher-quality) solutions faster than IP1 and CP1. When we compare the performance of the proposed method using historical production data, we find that IP2/IP1 is able to generate high-quality feasible and real-world schedules for a week in 8.4 minutes (on average) compared to 28 hours of manual schedule generation at Beyçelik. In addition, IP2/IP1's makespan values are 6.1% shorter (on average) than those of the company. Through the use of IP2/IP1, the company can increase its weekly production by 8.7 hours (on average) and can eliminate 28 hours of weekly scheduling efforts. Our analysis also shows that using the IP2/IP1 model can allow the company to handle demand increases of up to 15% without increases in makespan. Furthermore, by using IP2/IP1, we quantify the relationship between demand increase (and decrease) and makespan increase (and decrease).

We estimate that, considering just energy and labour costs, the company can save over €187,000 annually by using the proposed IP2/IP1 method. The estimate does not include the savings of 28 hours of manual scheduling efforts. Furthermore, with this method, the company gains the ability to quickly reschedule when needed, thereby increasing its corporate flexibility and level of competitiveness. We believe that the models developed in this paper are sufficiently generic to be used in similar companies and industries, and that the models can generate effective and usable schedules with significant savings.

Our study has not yet been implemented at Beyçelik, but the company is pleased with the results of our approach, tested them during a study period, and is considering a complete software implementation. The R&D manager of the company, Necip Ceylan, said, "We think that the savings estimated and the results presented in the study are quite realistic. The results are impressive.

They have a huge potential for our company, and we would consider implementing the study [once] ... it is converted to a software [program]."

Our analysis and discussion are organized in our paper as follows: We review the literature in Section 2. In Section 3, we describe the IP1 model and compare its performance with CP1's performance. In Section 4, we present the IP2/IP1 method and our theoretical results, and we conduct numerical experiments to show the effectiveness of the theoretical results and performance of IP2/IP1 compared with IP1. We further evaluate the performance of IP2/IP1 and compare IP2/IP1 schedules with the company's schedules and other models, and we estimate the savings in Section 5. In Section 6, we analyze the potential growth of production capacity and the relationship between demand and makespan by using the IP2/IP1 method. We conclude the paper in Section 7.

2. Literature Review

Many solution methods have been developed by researchers with the aim of solving real-world scheduling challenges (Khayat et al. 2006, Barlatt et al. 2010, 2012, Relvas et al. 2013) related to our Stamping Scheduling Problem. For instance, Barlatt et al. (2010) considered a Stamping Scheduling Problem with jobs that have only one operation and contain identical machines. Hence, these authors were able to consider the scheduling problem as a task-sequencing problem. They first developed a test-and-prune algorithm, and were able to solve small size problems to provable optimality with that method instead of using traditional integer programming. Then, Barlatt et al. (2012) built a decision support tool (i.e., the just-in-time execution and distribution information (JEDI) system), which took into account the whole production environment, including supply chain and workforce allocation. Stamping Scheduling Problems have been considered by researchers, but to the best of our knowledge, none of the problems studied in the literature contains the specific problem characteristics that we consider in this paper; i.e., machine eligibility restrictions and the requirement of consecutive machines for operations.

The Stamping Scheduling Problem that we address in this paper has some similarities to the two dimensional (2D) cutting (e.g., Gilmore and Gomory (1965)), packing (e.g., Martello et al. (2003), Lodi et al. (1999)), and display (Geismar et al. 2015) problems. Wäscher et al. (2007) developed a classification method for cutting and packing problems with five criteria: dimensionality, assortment of large objects, characteristics of large objects, assortment of small items, and characteristics kind of assignment. By using this classification method, we compare our Stamping Scheduling Problem with other similar problem types in Table 1.

Table 1 Comparison of problem types

Problem Type	Dimensionality	Assortment of large objects	Characteristics of large objects	Assortment of small items	Characteristics of assignment
Stamping Scheduling Problem	2	One large object	Variable dimension	Strongly heterogenous	Specific locations
2D Strip Packing Problem (Martello et al. 2003)	2	One large object	Variable dimension	Strongly heterogenous	Any locations
2D Bin Packing Problem (Lodi et al. 1999)	2	More than one large object	All dimensions are fixed	Strongly heterogenous	Any locations
2D Cutting Stock Problem (Gilmore and Gomory 1965)	2	More than one large object	All dimensions are fixed	Weakly heterogenous	Any locations
2D Display Problem (Geismar et al. 2015)	2	More than one large object	All dimensions are fixed	Identical	Any locations

We can see the differences and similarities between the Stamping Scheduling Problem and other problem types from Table 1. In particular, the Stamping Scheduling Problem is more similar to the 2D strip packing problem than other problem types given in Table 1. In both problems, small rectangular items have to be assigned to one large object that has a variable dimension. This variable dimension is called "height" in the 2D strip packing problem while it is called "makespan" in the Stamping Scheduling Problem. However, the Stamping Scheduling Problem differs from the 2D strip packing problem and all other problems in the assignment (cut) locations of small items. In the stamping problem, these assignments must be in specific locations since each assignment represents a suitable machine. In all other problems, these assignments (cuts) can be in any location as long as they are in the large object. This additional restriction distinguishes the Stamping Scheduling Problem from other 2D cutting, packing, and display problems.

Finding optimal solutions for scheduling problems in acceptable times by mixed integer programming has remained a challenge because of the NP-Hard nature of the problems (Tran and Beck 2012). Edis and Ozkarahan (2012) considered a real-life, resource-constrained parallel machine scheduling problem in an injection-molding department and developed an integer programming (IP) model for minimizing the makespan. However, a high number of variables and constraints prevented the model from finding an optimal solution. The researchers then considered constraint programming and developed two solution approaches - IP/IP and IP/CP - by dividing the whole problem into two sub-problems: a job-machine assignment problem and a sequencing problem. The researchers concluded that the IP/IP model performed better in test problems, whereas the IP/CP model provided fast and practical results in all test problems and reached more effective solutions in less than one minute when the resource constraints were tight. Similarly, we divide our problem into two problems and develop an algorithm (IP2/IP1) that combines two integer programming models; however, our problem is different since it requires additional constraints for machine eligibility restrictions and consecutive operations of jobs.

To combine the strengths of different methodologies, researchers tend to develop hybrid methods using decomposition algorithms that have similarities to generalized Benders decomposition (Jain and Grossmann 2001, Harjunkoski and Grossmann 2002, Canto 2008). However, Benders decomposition requires linear programming for sub-problems, whereas scheduling problems have a combinatorial nature that requires integer programming. Hooker and Yan (1995) developed the logic-based Benders decomposition (LBBD) method to implement decomposition to combinatorial problems. The LBBD method has been implemented to solve many different kinds of combinatorial problems (Cambazard et al. 2004, Hooker 2005, 2007, Fazel-Zarandi and Beck 2009, Coban and Hooker 2010, Heinz and Beck 2012, Tran and Beck 2012, Xiaolu et al. 2014, Camargo et al. 2014). Hooker (2007) developed an LBBD algorithm for decomposing a planning and scheduling problem. He tested the algorithm with different objective functions, such as minimizing cost, makespan, and total tardiness. He also compared the results with mixed integer linear programming (MILP) and CP. Although the decomposition method could not solve larger problems to optimality, the method outperformed MILP. Fazel-Zarandi and Beck (2009) developed an LBBD approach for solving a location-allocation problem. They compared the LBBD with an integer program and a Tabu Search approach. Results showed that LBBD found optimal solutions faster than did integer programming, and it reached better feasible solutions in less time than did Tabu Search. Xiaolu et al. (2014) developed a decomposition method by combining ant colony and dynamic programming techniques to schedule satellites. These studies may be strengthened with the help of meta-heuristics and exact methods (Camargo et al. 2014). The papers referenced in this paragraph show that various decomposition approaches can be developed by using exact methods and heuristics in LBBD.

We are inspired by from the communication between models in the LBBD algorithm, and we extend and customize this approach for our problem. We first develop a new model IP1, and we compare it with a constraint programming model CP1. Then, we build the IP2/IP1 method by dividing the main problem into two sub-problems: assignment and sequencing. Compared to IP1, CP1, and the company's method, IP2/IP1 is able to solve the scheduling problem with higher quality (shorter makespan), and it reduces the solution time.

3. An Integer Programming Model: IP1

We develop the IP1 model to solve the company's "Stamping Scheduling Problem" by following the company's practices and conventions to mimic its system as closely as possible. Our aim is to quickly build feasible, usable, and effective real-world schedules for the company.

Since the company has only one set of die pairs for each operation, a job and its operations can be produced at only one machine set at a time, which makes the changeover of dies significant.

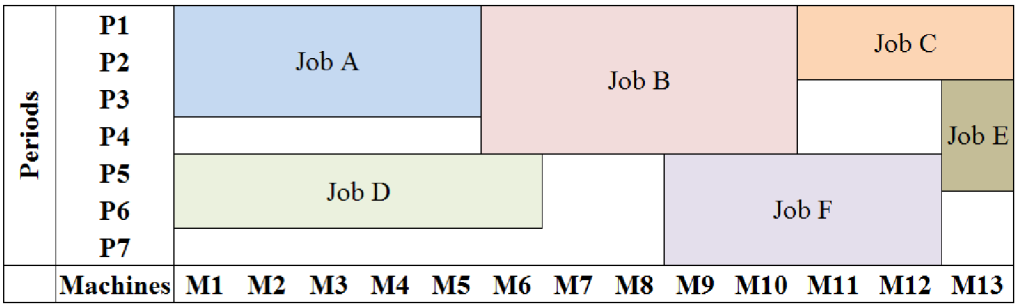
The changeover of dies takes about 30 minutes and increases the total production time. During the changeovers, production must stop and the related machines remain idle, resulting in losses for the company. However the company gives 30 minutes of breaktime every four hours, such as rest breaks, lunch, or shift changes. The company uses this idle time to complete the changeovers, which enables the company to use a four-hour production period (i.e., when a job is assigned to a related machine, it must require at least four hours of run time). Thus, the company can complete the changeovers in idle time and extend the actual production time. Although this practice excludes die changeovers from production times, it may allow production to exceed demand from time to time.

To evaluate the effectiveness of this practice, Caglar Gencosman et al. (2014) used a model to find an optimum period length considering the trade-off between production amount and changeovers. They concluded that the practice of using a four-hour production period was a good choice. Since Beyçelik uses a four-hour production period and does not allow pre-emption of jobs, we follow the same behaviour in our study and use the practice of a four-hour production period so that we can compare the schedules from the company and our models more fairly.

We analyze the company databases to understand Beyçelik's production levels and demand. The company runs two separate databases for production. The first one keeps track of production plans and includes customer demand, while the second one is created by the Manufacturing Execution System (MES) that oversees the entire pressline. The MES collects real-time production data from each machine, including data on scraps and rejected parts. Therefore, the amount of production in the MES database is always greater than that in the production plan database. We take the MES database values for demand; that is, we use real production amounts in our models. We calculate the total duration of job i (J_i) by multiplying its processing times (in seconds) and its demand, and we find the required number of periods (H_i). Figure 2 shows a visualization of this computation. The horizontal axis represents 13 machines ($M_{max} = 13$), and the vertical axis represents the required number of periods (i.e., the total scheduling horizon). We can assign jobs to machines in terms of their height and width. (The total number of jobs is I_{max}.)

The notation of the model is presented in Table 2. The number of periods in the model is given by K_{max}; that is, K_{max} is the last period that a job can start; hence, it does not represent the makespan. Choosing a suitable K_{max} is important since this value directly affects the problem size and its feasibility. Ideally, we would like to choose the smallest K_{max} value that would make the problem instance feasible and speed up the IP1 model. To determine a good value of K_{max} for any given instance of IP1, we develop an algorithm (MinK_{max}). The idea is to run the MinK_{max}

Figure 2 **An example of stamping scheduling**



algorithm first to determine a good K_{max} value and then run the model under consideration with the determined K_{max} value. (We use the MinK_{max} algorithm for all models; therefore, we do not add the algorithm's run time to the models' solution times, except when comparing the models with the company's schedules in Table 10 where we do not use the algorithm.) The algorithm starts with two K_{max} estimates: one *lower bound (lb)* and one *upper bound (ub)*. It then chooses a new K_{max} value based on these bounds and uses that value to determine the feasibility of an instance. Next, based on the feasibility or the infeasibility of that instance, the *lb* and *ub* are updated, and the process is repeated. The algorithm stops when the gap between the *lb* and *ub* is sufficiently small; that is, when the gap is less than two ($ub - lb < 2$).

Note that the MinK_{max} algorithm provides a feasible value for K_{max} only, and it does not yield a solution (i.e., a schedule or a makespan value). We need IP1 - or another method - to come up with a schedule (i.e., value of the decision variables and value of the makespan). Recall that the MinK_{max} algorithm determines K_{max}, which is the highest index value for periods in the models, and it indicates the last period that a job can start. Therefore, the models may have an optimal solution with $C_{max} > K_{max}$. The details of the algorithm are presented in Appendix B. Our model IP1, which solves the Stamping Scheduling Problem, is structured as follows:

We now present our model IP1 which solves the Stamping Scheduling Problem.

$$\text{minimize } IP1_{C_{max}} \tag{1}$$

subject to

$$A_{im}x_{ikm} * (H_i + k - 1) \leq IP1_{C_{max}} \quad \forall i \in I, \forall k \in K, \forall m \in M \tag{2}$$

$$\sum_{i \in I} A_{im}x_{ikm} \leq 1 \quad \forall k \in K, \forall m \in M \tag{3}$$

$$\sum_{k \in K} \sum_{m \in M} A_{im}x_{ikm} = 1 \quad \forall i \in I \tag{4}$$

Table 2 Indices, parameters, sets, and variables for IP1

Indices (for sets see below):	
i	Index of jobs to be scheduled, $i \in I$.
k	Index of periods, $k \in K$.
m	Index of machines, $m \in M$.
Parameters:	
A_{im}	A_{im} is 1 if job i can be processed on machine m and zero otherwise.
J_i	Total duration of job i.
U	Period length (4 hours).
H_i	Total period of job i calculated by $\lceil (J_i/3600)/U \rceil$.
O_i	The number of operations of job i.
Sets:	
I	The set of jobs. I=$\{1,2,\ldots,I_{max}\}$.
K	The set of periods. K=$\{1,2,\ldots,K_{max}\}$.
M	The set of machines. M=$\{1,2,\ldots,M_{max}\}$.
Decision Variables:	
x_{ikm}	$\begin{cases} 1, \text{if job } i \text{ starts in period } k \text{ on machine } m; \\ 0, \text{otherwise.} \end{cases}$
$IP1_{C_{max}}$	Makespan

$$A_{tn}x_{tzn} \leq 1 - A_{im}x_{ikm} \quad \forall i,t \in I, \forall k,z \in K, \forall m,n \in M \mid t \neq i, k \leq z \leq k + H_i - 1, n \leq m + O_i - 1, m \leq n + O_t - 1 \tag{5}$$

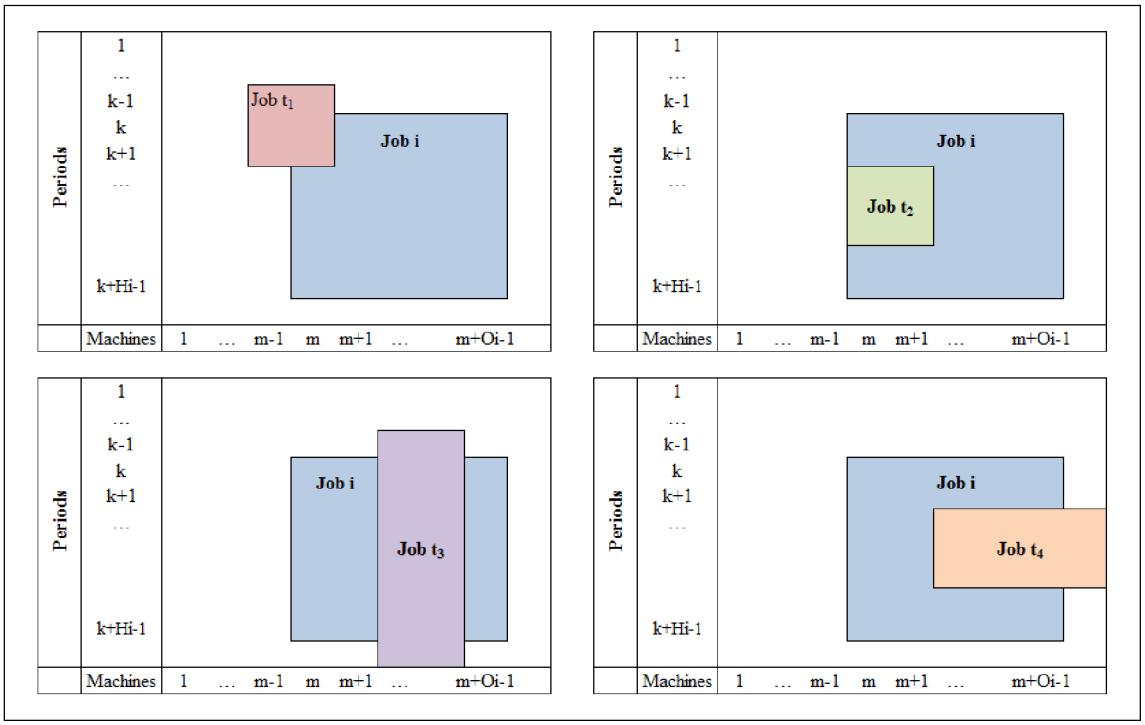
$$IP1_{C_{max}} \geq 0 \tag{6}$$

$$x_{ikm} \in \{0,1\} \quad \forall i \in I, \forall k \in K, \forall m \in M \tag{7}$$

The objective is to minimize the makespan $IP1_{C_{max}}$ (1). For each job i and for each start period k, we find the end time on machine m with constraint (2). Constraint (3) ensures that there can be, at most, one job assigned for each machine m and period k. Each job i must be assigned to only one machine and to only one period, a condition that is achieved by constraint (4). Constraint (5) guarantees that no two jobs can overlap in the schedule. We show a sample of prevented assignments of jobs in Figure 3. We developed four different constraints for this non-overlapping condition, evaluated their performances and chose the one given in (5). (Details are given in Appendix C). Finally, constraints (6-7) ensure non-negativity of $IP1_{C_{max}}$ and binary condition of x_{ikm} variables.

After building IP1, we compare its performance with CP1 of Caglar Gencosman et al. (2014) using instances that mimic historical Beyçelik schedules. To make a comparison between IP1 and CP1, we generate 15 different problem instances using real jobs from Beyçelik's schedules. We select the number of jobs as being between 50 and 70 (since the number of real schedules is between 50 and 70) in increments of five jobs (i.e., 50, 55, 60, 65, 70), and for each of these, we generate three instances by picking the jobs randomly. We solve these instances with IP1 and CP1 in ILOG

Figure 3 Prevented inconsistent assignments of job t by constraint (5)



CPLEX 12.5, with a solution time limit of 900 seconds for both models. (The company wanted a solution of a single schedule to be generated in less than 15 minutes.) We run the models on a PC with a 2.90 GHz i7 processor and six GB RAM. (We use the same settings and the same PC for all runs presented in the paper.) The results of IP1 and CP1 are provided in Table 3.

The first two columns present the number of jobs and the instance number, followed by the results of MinK_{max} for each instance (K_{max} value in periods and elapsed time in seconds). The C_{max} section gives the makespan values (in periods) obtained by IP1 and CP1. The final columns show the elapsed time (in seconds) for IP1 and CP1, the optimality gap percentage for IP1, and the average percentage deviation of the best objective line (ADOL) (Meyr and Mann 2013) for IP1 and CP1, as in (8). We use the same or a similar structure for other tables in the paper.

$$ADOL\% = \frac{CP1_{C_{max}} - IP1_{C_{max}}}{CP1_{C_{max}}} * (100) \tag{8}$$

The results indicate that IP1 performs better than CP1 in all but one instance in which both yield the same makespan value of 31. IP1's solution quality is 7.6% higher (i.e., its makespan values are 7.6% shorter) than CP1's on average, and IP1 runs faster than CP1. However, IP1 does not

Table 3 Comparison of IP1 and CP1 with randomly generated instances from company schedules

# of Jobs	Problem	MinK_{max}		C_{max}(per.)		Elapsed Time(s)		Gap%	ADOL%
		K_{max} (per.)	Elapsed Time(s)	IP1	CP1	IP1	CP1	IP1	CP1-IP1
50	1	24	231.7	24	26	41.4	900.0	-	7.7
	2	26	186.6	26	27	47.1	900.0	-	3.7
	3	24	142.8	25	27	64.3	900.0	-	7.4
55	1	31	318.8	31	31	69.0	900.0	-	0.0
	2	28	313.0	28	31	220.6	900.0	-	9.7
	3	27	218.6	27	29	333.1	900.0	-	6.9
60	1	31	420.0	31	33	104.4	900.0	-	6.1
	2	29	583.9	29	34	520.4	900.0	-	14.7
	3	31	390.6	32	35	900.0	900.0	6.3	8.6
65	1	33	637.8	33	36	900.0	900.0	15.2	8.3
	2	32	626.1	33	37	900.0	900.0	12.1	10.8
	3	35	520.0	35	38	889.8	900.0	-	7.9
70	1	34	900.0	35	38	900.0	900.0	28.6	7.9
	2	36	845.0	38	42	900.0	900.0	70.3	9.5
	3	36	696.2	37	39	900.0	900.0	71.8	5.1
					Avg.	512.7	900.0	13.6	7.6

reach an optimal solution in the given time (of 900 seconds) for most of the instances and still has (large) optimality gaps in some cases.

Unsurprisingly, the Stamping Scheduling Problem, which we can solve with IP1 is NP-hard. The problem is strongly NP-hard if the number of machines in the input is a variable (Theorem 1) and it is NP-hard in the ordinary sense if the the number of machines is fixed (Theorem 2).

Theorem 1 *Stamping Scheduling Problem is strongly NP-hard if the number of machines in the input is a variable.*

A pseudo-polynomial time algorithm for the Stamping Scheduling Problem may be possible since Theorem 2 does not rule out that possibility.

Theorem 2 *Stamping Scheduling Problem is NP-hard in the ordinary sense if the number of machines is fixed.*

We provide the proofs of the theorems in Appendix D. To find a solution method that is faster than IP1, we divide the main problem into two integer programs and develop the IP2/IP1 method. We evaluate the effectiveness of IP2/IP1 by comparing its performance to the performance of IP1 and CP1, and to that of the company's schedules. We present the IP2/IP1 method in the next section.

4. IP2/IP1 Solution Method

We have two decisions to make for the Stamping Scheduling Problem. The first decision concerns the assignment of jobs to machines, and the second concerns the sequence of jobs on those machines. We see this situation more clearly in the formulation of IP1 with the x_{ikm} decision variables that determine whether or not job i starts on processing machine m in period k.

One idea to simplify and make IP1 solution faster is to divide the Stamping Scheduling Problem into two smaller ones: a master problem and a sub-problem. The master problem solves an assignment problem, considering only jobs and machines, and sends the assignment to the sub-problem. The sub-problem then determines the starting periods for each job without changing their assigned machines. We show that this method delivers high-quality solutions in a short computation time and that it dramatically reduces the solution time.

Usually, in the literature, sub-problems are generated for each machine to schedule jobs on those machines. However, in our Stamping Scheduling Problem, we cannot separate the jobs because their operations must be processed in sequential machines. Therefore, we use only one sub-problem for the whole production environment.

For the IP2/IP1 method, we first develop a new integer program IP2 for the assignment (master) problem and then use a slightly modified IP1 ($\overline{IP1}$) for the sequencing problem (sub-problem). The decision variables of IP2 are given in Table 4. (We do not redefine the notation used in IP1.)

Table 4 Decision variables for IP2

Decision Variables:	
x_{im}	$\begin{cases} 1, \text{if job } i \text{ assigned to machine } m; \\ 0, \text{otherwise.} \end{cases}$
$IP2_{C_{max}}$	Makespan(IP2).

Our formulation of IP2 is as follows:

$$\text{minimize } IP2_{C_{max}} \tag{9}$$

subject to

$$\sum_{i \in I} \sum_{n \in M | n \le m \le n+O_i-1} A_{in} x_{in} * H_i \le IP2_{C_{max}} \quad \forall m \in M \tag{10}$$

$$\sum_{m \in M} A_{im} x_{im} = 1 \quad \forall i \in I \tag{11}$$

$$IP2_{C_{max}} \ge 0 \tag{12}$$

$$x_{im} \in \{0,1\} \quad \forall i \in I, \forall m \in M \tag{13}$$

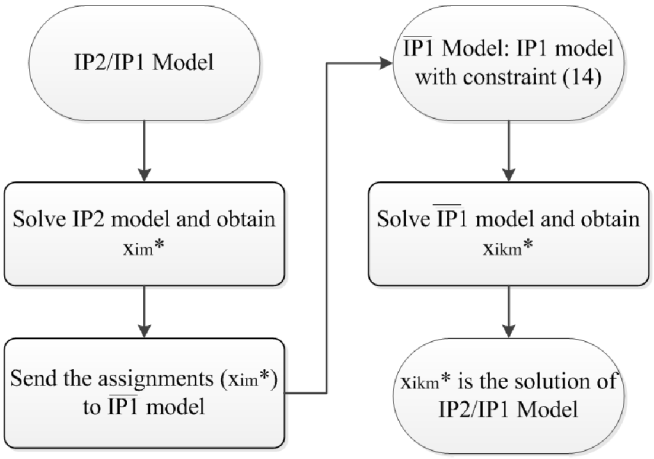
The objective function of the master problem IP2 is the minimization of the total processing time of all machines, makespan(IP2). (We need to differentiate makespan values of IP2 and IP1 since they may not be equal to each other. $IP1_{C_{max}}$ includes processing times and may include idle time between jobs. $IP2_{C_{max}}$ includes only processing times and no idle time.) Constraint (10) calculates

$IP2_{C_{max}}$. Constraint (11) ensures that each job must be assigned to only one machine. Constraints (12) and (13) define the solution space of the decision variables.

The IP2/IP1 method solves IP2 to determine the optimal x^*_{im} variables. Then, IP2/IP1 sends this optimal assignment to the sub-problem $\overline{IP1}$. $\overline{IP1}$ is the same as the original IP1 model except for the addition of constraint (14) and the replacement of the objective value with $\overline{IP1}_{C_{max}}$. Constraint (14) links the master problem and sub-problem by sending the solution of IP2 to $\overline{IP1}$. $\overline{IP1}$ starts with fixed machine-job assignments and determines starting periods of jobs. The solution process of IP2/IP1 is depicted in Figure 4.

$$\sum_{k \in K} A_{im} x_{ikm} = A_{im} x^*_{im} \quad \forall i \in I, m \in M \tag{14}$$

Figure 4 Flow diagram of IP2/IP1 Method



The optimal value of $\overline{IP1}$, $\overline{IP1}_{C_{max}}$, provides an upper bound for $IP1_{C_{max}}$ (Lemma 1). We also find a lower bound on $IP1_{C_{max}}$ (Lemma 2 and Lemma 3). Therefore, we obtain values and solutions for (the original problem) IP1 from below and above. Our results in Table 6 show that these bounds are tight, and we obtain both the lower bound and upper bound quickly. Hence, we conclude that our IP2/IP1 method solves the scheduling problem IP1 with high quality and efficiency.

Lemma 1. $IP1_{C_{max}} \leq \overline{IP1}_{C_{max}}$.

Proof. Any solution of $\overline{IP1}$ is a solution of IP1 since IP1 is a relaxation of $\overline{IP1}$. The result follows. □

To find a lower bound on the value of $IP1_{C_{max}}$, we define the model $\underline{IP1}$ as being the same as IP1, except for replacing its objective with $\underline{IP1}_{C_{max}}$ and updating constraint (2) with constraint (15), as shown below.

$$\sum_{i\in I}\sum_{k\in K}\sum_{n\in M|n\le m\le n+O_i-1} A_{in}x_{ikn} * H_i \le \underline{IP1}_{C_{max}} \quad \forall m \in M \tag{15}$$

We summarize the models $\underline{IP1}$, IP1, $\overline{IP1}$, and IP2 in Table 5.

Table 5 $\underline{IP1}$**, IP1,** $\overline{IP1}$**, and IP2 Models**

IP1: minimize $IP1_{C_{max}}$ *subject to* $A_{im}x_{ikm}(H_i+k-1) \le IP1_{C_{max}} \quad \forall i\in I, \forall k\in K, \forall m\in M$ $\sum_{i\in I} A_{im}x_{ikm} \le 1 \quad \forall k\in K, \forall m\in M$ $\sum_{k\in K}\sum_{m\in M} A_{im}x_{ikm} = 1 \quad \forall i\in I$ $A_{tn}x_{tzn} \le 1 - A_{im}x_{ikm} \quad \forall i,t\in I, \forall k,z\in K, \forall m,n\in M \mid$ $t\ne i, k\le z\le k+H_i-1, n\le m+O_i-1, m\le n+O_t-1$ $IP1_{C_{max}} \ge 0$ $x_{ikm}\in\{0,1\} \quad \forall i\in I, \forall k\in K, \forall m\in M$	**IP2:** minimize $IP2_{C_{max}}$ *subject to* $\sum_{i\in I}\sum_{n\in M\mid n\le m\le n+O_i-1} A_{in}x_{in}H_i \le IP2_{C_{max}} \quad \forall m\in M$ $\sum_{m\in M} A_{im}x_{im} = 1 \quad \forall i\in I$ $IP2_{C_{max}} \ge 0$ $x_{im}\in\{0,1\} \quad \forall i\in I, \forall m\in M$
$\underline{IP1}$: minimize $\underline{IP1}_{C_{max}}$ *subject to* $\sum_{i\in I}\sum_{k\in K}\sum_{n\in M\mid n\le m\le n+O_i-1} A_{in}x_{ikn}H_i \le \underline{IP1}_{C_{max}} \quad \forall m\in M$ $\sum_{i\in I} A_{im}x_{ikm} \le 1 \quad \forall k\in K, \forall m\in M$ $\sum_{k\in K}\sum_{m\in M} A_{im}x_{ikm} = 1 \quad \forall i\in I$ $A_{tn}x_{tzn} \le 1 - A_{im}x_{ikm} \quad \forall i,t\in I, \forall k,z\in K, \forall m,n\in M \mid$ $t\ne i, k\le z\le k+H_i-1, n\le m+O_i-1, m\le n+O_t-1$ $\underline{IP1}_{C_{max}} \ge 0$ $x_{ikm}\in\{0,1\} \quad \forall i\in I, \forall k\in K, \forall m\in M$	**$\overline{IP1}$:** minimize $\overline{IP1}_{C_{max}}$ *subject to* $A_{im}x_{ikm}(H_i+k-1) \le \overline{IP1}_{C_{max}} \quad \forall i\in I, \forall k\in K, \forall m\in M$ $\sum_{i\in I} A_{im}x_{ikm} \le 1 \quad \forall k\in K, \forall m\in M$ $\sum_{k\in K}\sum_{m\in M} A_{im}x_{ikm} = 1 \quad \forall i\in I$ $A_{tn}x_{tzn} \le 1 - A_{im}x_{ikm} \quad \forall i,t\in I, \forall k,z\in K, \forall m,n\in M \mid$ $t\ne i, k\le z\le k+H_i-1, n\le m+O_i-1, m\le n+O_t-1$ $\sum_{k\in K} A_{im}x_{ikm} = A_{im}x^*_{im} \quad \forall i\in I, m\in M$ $\overline{IP1}_{C_{max}} \ge 0$ $x_{ikm}\in\{0,1\} \quad \forall i\in I, \forall k\in K, \forall m\in M$

The IP1 and $\underline{IP1}$ models have the same constraints except for the one that calculates C_{max} values. The difference between $\overline{IP1}$ and IP1 is the addition of constraint (14). IP2 has the same constraint for C_{max} calculation with $\underline{IP1}$, and it has only one similar constraint with IP1, which is constraint (4) stating that each job has to be assigned to a machine.

We now find a lower bound on the value of $IP1_{C_{max}}$.

Lemma 2. $\underline{IP1}_{C_{max}} \leq IP1_{C_{max}}$.

Proof. Both models ($\underline{IP1}$ and IP1) are identical except for one constraint. In $\underline{IP1}$, constraint (15) gives total processing time on each machine (i.e., no idle time exists between jobs in this computation). On the other hand, constraint (2) in IP1 calculates the makespan (i.e., maximum of job finish times on each machine), and it may include idle time. Hence, any IP1 solution is feasible for $\underline{IP1}$ with a lower objective value than $IP1_{C_{max}}$. The result follows. □

Finding this lower bound, $\underline{IP1}_{C_{max}}$ is not easier than solving IP1 itself. Fortunately, however, the optimum value of IP2 is a lower bound on $\underline{IP1}_{C_{max}}$.

Lemma 3. $IP2_{C_{max}} \leq \underline{IP1}_{C_{max}}$.

Proof.

Let x^*_{im} be an optimal solution of IP2. We define

$$\overline{x}_{ikm} = \begin{cases} x^*_{im} \ \textit{if } k\textit{=1}; \\ 0, \ \textit{otherwise.} \end{cases}$$

Then $\overline{x}_{ikm}$ is optimal for the relaxation of $\underline{IP1}$ (omitting constraints (3) and (5)) with the same objective function value of IP2. The result follows. □

By lemmata 1, 2, and 3 above, we obtain our main result.

Proposition. $IP2_{C_{max}} \leq \underline{IP1}_{C_{max}} \leq IP1_{C_{max}} \leq \overline{IP1}_{C_{max}}$.

We compare $IP2_{C_{max}}, IP1_{C_{max}}, \underline{IP1}_{C_{max}}$, and $\overline{IP1}_{C_{max}}$ (IP2/IP1) values in Table 6. We use the generated 15 instances from company schedules and the MinK_{max} algorithm for each instance to determine K_{max}. We see that although IP1 reaches feasible or optimal results with an average of 512.7 seconds, it cannot solve nearly half of the problems optimally within the time limit of 900 seconds; IP1 has 13.6% optimality gap on average. In contrast, IP2/IP1 finds optimal solutions with an average of 114.2 seconds, which is four times faster than IP1. Furthermore, compared to IP1, IP2/IP1 improves IP1 solutions by 1.5%, on average.

The results in the $\underline{IP1}$ and $\overline{IP1}$ columns show that the bounds are tight and that the IP2/IP1 method works well. (In all instances, IP2/IP1 found the optimal solution. Note that in some instances, since IP1 cannot find an optimal solution within 900 seconds, its values can be higher than $\overline{IP1}$.) Hence, we conclude that IP2/IP1 provides good - and, in most instances, optimal - solutions within the desired time set by the company.

Table 6 Comparison of C_{max} values with generated instances from company schedules

# of Jobs	Problem	MinK_{max}		C_{max}(per.)				Elapsed Time(s)		Gap%	ADOL%
		K_{max} (per.)	Elapsed Time(s)	IP1	IP2	$\underline{IP1}$	IP2/IP1 $\overline{IP1}$	IP1	IP2/IP1	IP1	IP1-IP2/IP1
50	1	24	231.7	24	24	24	24	41.4	26.6	-	0.0
	2	26	186.6	26	26	26	26	47.1	29.7	-	0.0
	3	24	142.8	25	24	24	24	64.3	32.7	-	4.2
55	1	31	318.8	31	31	31	31	69.0	82.9	-	0.0
	2	28	313.0	28	28	28	28	220.6	42.0	-	0.0
	3	27	218.6	27	27	27	27	333.1	45.6	-	0.0
60	1	31	420.0	31	31	31	31	104.4	63.9	-	0.0
	2	29	583.9	29	29	29	29	520.4	86.8	-	0.0
	3	31	390.6	32	31	31	31	900.0	227.1	6.3	3.2
65	1	33	637.8	33	33	33	33	900.0	149.5	15.2	0.0
	2	32	626.1	33	32	32	32	900.0	143.2	12.1	3.1
	3	35	520.0	35	35	35	35	889.8	119.6	-	0.0
70	1	34	900.0	35	34	34	34	900.0	129.3	28.6	2.9
	2	36	845.0	38	36	36	36	900.0	321.3	70.3	5.6
	3	36	696.2	37	36	36	36	900.0	213.1	71.8	2.8
							Avg.	512.7	114.2	13.6	1.5

We further evaluate the performance of IP2/IP1 in the next section by using a randomly generated data set and the company's historical schedules.

5. IP2/IP1 Performance

To further evaluate the performance of IP2/IP1 we use two different datasets: 1) a randomly generated dataset from empirical probability distributions of parameters in the historical schedules, and 2) the real production dataset from the company's historical schedules. We compare IP2/IP1 with other models by using the first dataset, and we compare the company-generated (and executed) schedules with those created by IP2/IP1, IP1, and CP1 by using the second dataset.

5.1. Comparison of Models with Randomly Generated Data

To compare the performances of IP1, CP1, and IP2/IP1 models by using a randomly generated dataset, we first analyze the historical data to obtain the frequencies of possible parameters' values, as given in Table 7. Then, we randomly generate problems based on these frequencies. For each instance, the number of jobs varies from 50-70 (considering the company schedules varies from 50-70), and each instance contains three different problems.

Table 7 Empirical distribution of model parameters

Processing Times of Operations	Probability	# of Operations	Probability	Order Amount	Probability	# of Alternative Machines	Probability
4	0.021	1	0.431	1000	0.158	1	0.615
9	0.106	2	0.078	2000	0.263	2	0.308
10	0.043	3	0.078	3000	0.263	3	0.077
12	0.149	4	0.353	4000	0.316		
14	0.064	5	0.059				
15	0.149						
18	0.234						
19	0.021						
20	0.213						

We compare the performances of models and provide the results in Table 8. The first two columns present the number of jobs and the instance number, followed by the results of MinK_{max} for each instance (K_{max} value in periods and elapsed time in seconds). The C_{max} section gives the makespan values (in periods) obtained by IP1, CP1, and IP2/IP1; the last two columns show the elapsed time (in seconds) for models and the optimality gap percentage for IP1 and IP2/IP1. We see that IP2/IP1 reaches the same solutions faster than the other two models within the time limit of 900 seconds. IP2/IP1 is six times faster than CP1 and approximately three times faster than IP1, on average. The IP2/IP1 model is able to find the optimal solution of problems in 148.3 seconds (i.e., in less than three minutes), on average.

Table 8 Comparison of IP1, CP1, and IP2/IP1 with randomly generated data

# of Jobs	Problem	MinK_{max}		C_{max}(per.)				Elapsed Time(s)			Gap%	
		K_{max} (per.)	Elapsed Time(s)	IP1	CP1	$\underline{IP1}$	IP2/IP1 $\overline{IP1}$	IP1	CP1	IP2/IP1	IP1	IP2/IP1
50	1	61	900.0	61	61	61	61	900.0	900.0	80.2	18	-
	2	33	138.4	33	34	33	33	103.7	900.0	33.5	-	-
	3	47	109.6	47	47	47	47	90.9	900.0	65.3	-	-
55	1	72	732.2	72	72	72	72	597.4	900.0	185.9	-	-
	2	61	900.0	61	61	61	61	900.0	900.0	193.1	3.3	-
	3	40	900.0	40	40	40	40	900.0	900.0	133.7	2.5	-
60	1	55	557.8	55	55	55	55	838.7	900.0	171.9	-	-
	2	70	488.9	70	70	70	70	160.7	900.0	122.3	-	-
	3	59	423.5	59	59	59	59	168.3	900.0	81.1	-	-
65	1	66	795.3	66	66	66	66	226.3	900.0	156.5	-	-
	2	60	900.0	46	46	46	46	900.0	900.0	181.8	41.3	-
	3	48	413.5	48	48	48	48	237.6	900.0	136.1	-	-
70	1	89	900.0	81	81	81	81	235.4	900.0	221.1	-	-
	2	76	900.0	76	76	76	76	309.5	900.0	250.9	-	-
	3	69	900.0	68	68	68	68	426.8	900.0	210.4	-	-
							Avg.	466.4	900.0	148.3	4.3	0.0

To investigate the limits of IP1 and IP2/IP1 and compare the models to each other, in Appendix E, we conduct additional experiments with larger data set values than those in the company's schedules. The results show that IP2/IP1 is approximately six times faster than IP1 even with larger data sets (Table 17).

5.2. Comparison of Models with Company-Generated (and Executed) Schedules

We examine the executed schedules and production data of a prior seven week period at Beyçelik. Using the real production dataset, we compare the company schedules with those created by the models. As before, we use the MES database for demand data. The company faces different types of interruptions, such as machine breakdowns, maintenance, or public holidays. We analyze the interruptions and consider them fully in our comparison. To eliminate any biases towards models, we implement the following:

1. Identify start times and durations of each interruption and find the machine(s) on which the interruption occurs.

2. Run the model without any interruptions to determine the schedule.

3. If the schedule includes any interruptions, add each interruption as a dummy job to the model. The dummy job has one operation and one alternative machine, and its total production time is equal to the total stoppage (interruption) time.

If an interruption happens at machine m' in period k', we add dummy job i' to the master problem IP2 of IP2/IP1 with constraint (16), and we add the dummy job i' to other models (sub-problem $\overline{IP1}$ of IP2/IP1, IP1, and CP1) with constraint (17). Similarly, we include constraint (17) to MinK_{max} algorithm when needed.

$$x_{i'm'} = 1 \tag{16}$$

$$x_{i'k'm'} = 1 \tag{17}$$

Due to rest breaks and changeovers, the pressline is active for 22.5 hours of a 24-hour period, and, as previously discussed, the company performs changeovers during shift changes or rest breaks. Beyçelik computes its total production horizon by considering the active production time without changeovers or public holidays. In addition, the company plans and implements *scheduled interruptions*, such as administrative interruptions, training sessions, or meetings. The scheduled interruptions reduce the active production time for an average of five hours per week. To consider this factor, we shorten the weekly active production time by five hours. In keeping with the company's practice, we allow changeovers to happen between periods in the models. In conclusion, the objective function of the IP2/IP1, IP1, CP1, and company schedules represents the active production time only. We present the comparison of company- and model-based schedules in Table 9.

Table 9 Comparison of company- and model-based schedules.

Week	# of Jobs	MinK_{max}		C_{max}(hr)				Elapsed Time(s)			C_{max} Diff.(hr)	ADOL%
		K_{max} (per.)	Elapsed Time(s)	Company	IP1	CP1	IP2/IP1	IP1	CP1	IP2/IP1	Comp-IP2/IP1	Comp-IP2/IP1
1	63	31	343.1	145	124	132	124	76.8	900.0	140.8	21	16.9
2	65	31	369.6	137.5	124	144	124	83.4	900.0	117.6	13.5	10.9
3	60	31	320.4	137.5	124	136	124	773.2	900.0	47.7	13.5	10.9
4	69	34	545.2	145	136	160	136	562.1	900.0	95.7	9.0	6.6
5	60	32	338.4	137.5	128	136	128	209.4	900.0	67.6	9.5	7.4
6	59	33	287.4	137.5	132	140	132	656.7	900.0	74.0	5.5	4.2
7	59	30	406.9	145	124	128	120	900.0	900.0	60.1	25	20.8
		Avg.	373.0				**Avg.**	465.9	900.0	86.2	13.9	11.1

Table 9 shows that the IP2/IP1 method finds solutions with the same or smaller makespan values. IP2/IP1 is also faster than other models and generates schedules in 86.2 seconds, on average.

In addition, IP2/IP1 outperforms the company schedules and is able to produce schedules in less than eight minutes (including the run time of the MinK_{max} algorithm). IP2/IP1 is able to generate shorter schedules, at 13.9 hours, on average, which represents an improvement over the company schedules by 11.1%. We provide a sample schedule generated by IP2/IP1 (of week 7) in Figure 5, where each job is identified with a colour and a number.

Figure 5 The schedule of IP2/IP1 for week 7

	Machines												
Periods	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
1	17	17	17	17	17	38	38	38	53	53	53	53	26
2	17	17	17	17	17	38	38	38	53	53	53	53	26
3	17	17	17	17	17	38	38	38	11	58	6	23	23
4	33	33	33	33	33	38	38	38	11	58	6	23	23
5	35	35	35	35	16	16	16	16	16	58	6	48	48
6	35	35	35	35	16	16	16	16	16	58	6	48	48
7	35	35	35	35	16	16	16	16	16		6	48	48
8	35	35	35	35	1	1	1	1	52	52	52	52	59
9	36	36	31	31	1	1	1	1	52	52	52	52	59
10	36	36	31	31	1	1	1	1	52	52	52	52	59
11	36	36	31	31	1	1	1	1	25	25	25	25	59
12	34	34	34	34	34	34	29	29	25	25	25	25	59
13	34	34	34	34	34	34	39	39	25	25	25	25	59
14	34	34	34	34	34	34	15	20	25	25	25	25	59
15	34	34	34	34	34	34	28	20	51	51	51	51	59
16	5	5	5	5	14	14	14	20	51	51	51	51	59
17	5	5	5	5	14	14	14	20	51	51	51	51	59
18	42	27	10	12	14	14	14	20	51	51	51	51	59
19	56	27	10	12	40	40	40	20	46	46	46	37	59
20	56	43	10	12	40	40	40	20	46	46	46	37	59
21	56	43	10	21	19	57	57	20	46	46	46	37	8
22	56	43		50	19	57	57	20	46	46	46	37	8
23	2	2	2	50	3	3	3	3	46	46	46	37	8
24	2	2	2	30	3	3	3	3	46	46	46	37	8
25	41	41	13	30	3	3	3	3	46	46	46	37	7
26	47	44	13	32	54	54	54	54	46	46	46	49	7
27	47	44	13	22	9	9	9	9	9	45	45	49	7
28	47	44	13	22	9	9	9	9	9	45	45	49	7
29	4	4	4	4	9	9	9	9	9	45	45	18	7
30	4	4	4	4	9	9	9	9	9	55	55	24	24

According to company records, the energy cost of a machine is €22 per hour, and the cost of a worker is €5.61 per hour. Each machine requires two workers to operate. Therefore, the total hourly energy cost of the pressline becomes (€5.61 x 2 workers x 13 machines)+(€22 x 13 machines) = €431.86 per hour.

The company's average weekly time savings is 13.9 hours with IP2/IP1. We use this figure to estimate annual savings. The company works about 50 weeks in a year; therefore, annual estimated savings (i.e., due to only pressline energy and labour cost savings) would be €431.86 x 13.9 hours x 50 weeks = **€300,142** per year, an amount that represents a significant savings.

IP2/IP1 starts its scheduling process with the knowledge of interruptions, which can be seen as an advantage of the model. To investigate this aspect and to make the model fit more closely to the

company's practice, we perform an additional comparison by rescheduling the production horizon when an interruption occurs. We present this analysis in the next section.

5.3. Rescheduling the Company Schedules

We use the same seven instances of weekly schedules as in the previous section. We first run IP2/IP1 without any interruptions and generate a weekly schedule. When an interruption occurs, we freeze the production and reschedule the weekly production, including the interruption and the remaining jobs. Therefore, we have to run IP2/IP1 more than once during each week due to interruptions. For instance, in week 2, we have to reschedule five times (six, if we consider the first scheduling run).

We reschedule the production week by fixing the previous assignments with the addition of constraint (16) to the master problem IP2 and constraint (17) to the sub-problem $\overline{IP1}$ of IP2/IP1 for all finished jobs. If the interruption causes pre-emption of a job, we define a dummy job with the same properties for the remaining processing activities, and we add this new dummy job to the model for rescheduling. When necessary, to prevent job assignments to past periods, we fix the number of jobs processed in past periods. If the allowable number of jobs is defined as L between periods k_1 and k_2, we add constraint (18) to sub-problem $\overline{IP1}$.

$$\sum_{i\in I}\sum_{k\in K|k_1\leq k\leq k_2}\sum_{m\in M} A_{im}x_{ikm} = L \tag{18}$$

We determine K_{max} as 40 for a week (22.5 active work hours of 7 days divided by 4 hours) for the rescheduling experiments in order to avoid solving MinK_{max} over and over again for the same week. We present the rescheduling results in Table 10. The first column contains the week information; the second column shows the number of jobs, which increases in every run because it includes the interruptions and/or preempted jobs as dummy jobs. The third column indicates the IP2/IP1 solution, including $\underline{IP1}$ and $\overline{IP1}$ results in periods. We convert the $\overline{IP1}$ results to hours in the fourth column to compare them with the company results. The fifth column demonstrates the elapsed time of schedules developed by the company and by the IP2/IP1 model. The elapsed time for the company is calculated by observing the scheduling process in a week.

The company schedules are prepared by a planning engineer who spends eight hours on the first day of a given week to develop the first schedule and then spends four hours on each of the successive five days to maintain production in terms of rescheduling the current schedule for any interruptions. In short, the engineer spends 28 hours per week, on average, for scheduling activities, which is presented as 100,800 seconds in the 'Company' column.

The sixth column shows the difference in C_{max} between company schedules and IP2/IP1 solutions in hours. The last column gives the improvement of weekly schedules by using IP2/IP1 instead of company schedules. The company spends 28 hours on planning its schedules (100,800 seconds), whereas IP2/IP1 spends 504.4 seconds (8.4 minutes) on average, which represents a significant improvement and savings. IP2/IP1 finds shorter schedules, which corresponds to a 6.1% improvement and 8.7 hours of savings per week, on average. As in the previous section, we estimate annual savings as €431.86 x 8.7 hours x 50 weeks = **€187,859** per year, only considering pressline energy and labour cost savings.

Table 10 Rescheduling with IP2/IP1

Week	# of Jobs	C_{max}(per.)		C_{max}(hr)		Elapsed Time(s)		Difference(hr)	ADOL%
		$IP1$	IP2/IP1 / $IP1$	Company	IP2/IP1	Company	IP2/IP1	IP2/IP1-Comp.	IP2/IP1-Comp.
1	61	29	29	145.0	116.0	-	95.0	-	-
	63	30	30	145.0	120.0	-	123.3	-	-
	65	31	31	145.0	124.0	-	356.8	21.0	14.5
					Total	100800	575.1		
2	60	30	30	137.5	120.0	-	103.9	-	-
	61	30	30	137.5	120.0	-	114.6	-	-
	63	30	31	137.5	124.0	-	186.1	-	-
	65	31	33	137.5	132.0	-	236.2	-	-
	66	31	32	137.5	128.0	-	265.2	-	-
	67	31	34	137.5	136.0	-	341.5	1.5	1.1
					Total	100800	1247.5		
3	57	30	30	137.5	120.0	-	90.0	-	-
	58	31	32	137.5	128.0	-	96.0	-	-
	59	31	33	137.5	132.0	-	209.2	-	-
	60	32	34	137.5	136.0	-	322.7	1.5	1.1
					Total	100800	717.9		
4	68	33	33	145.0	130.4	-	169.4	-	-
	69	34	34	145.0	136.0	-	134.9	9.0	6.2
					Total	100800	304.3		
5	59	31	31	137.5	124.0	-	100.9	-	-
	60	32	34	137.5	136.0	-	154.4	1.5	1.1
					Total	100800	255.3		
6	59	33	33	137.5	132.0	-	84.1	5.5	4.0
					Total	100800	84.1		
7	57	29	29	145.0	116.0	-	86.5	-	-
	59	31	31	145.0	124.0	-	102.9	-	-
	60	31	31	145.0	124.0	-	157.1	21.0	14.5
					Total	100800	346.5		
						Avg.	504.4	8.7	6.1

In summary, IP2/IP1 is able to generate fast, usable, high-quality real-world schedules, which means that Beyçelik can observe gains in many aspects with the use of IP2/IP1. First, it can reduce the cost of its pressline and can expect to save €187,859 per year. Second, Beyçelik can increase the capacity of its pressline and produce more products within the same time period limit. Third, it can save time and money by decreasing the weekly rescheduling time spent by its engineer from 28 hours to just minutes. Furthermore, in addition to these monetary benefits, there are non-monetary benefits such as increased customer satisfaction, ability to respond to market

changes and additional demand requests, improved scheduling capability, and ability to adapt to breakdowns and interruptions. We believe that our estimate is conservative and that the total savings could be much greater than €187,000.

6. Managerial Insights

In this section, we explore how much more production can be done with the same amount of (executed) production time if IP2/IP1 schedules are used. This analysis, in a sense, determines the potential production-capacity increase that becomes possible with IP2/IP1. We also analyze the relationship between demand and makespan.

6.1. Increased Demand Analysis

Our results show that, compared to the company-generated schedules, IP2/IP1 can create shorter schedules within minutes for the same amount of production. To investigate the potential growth of the pressline capacity using IP2/IP1, we repeat the same runs with increasing levels of demand.

We consider the company's historical schedules and increase the current demand by 5%, 10%, and 15% (e.g., 5% increment in demand is realized by multiplying 1.05 to the total duration of each job i, J_i). We present the IP2/IP1 model solutions for different demands in sequence as D_5%, D_10%, and D_15% in Table 11.

Table 11 Increased demand analysis

Week	# of Jobs	C_{max}(hr) Company	IP2/IP1	IP2/IP1 D_5%	IP2/IP1 D_10%	IP2/IP1 D_15%
1	63	145	124	128	132	136
2	65	137.5	124	128	128	140
3	60	137.5	124	132	136	136
4	69	145	136	136	144	152
5	60	137.5	128	136	140	148
6	59	137.5	132	140	140	148
7	59	145	120	128	136	140
	Avg.	140.7	126.9	132.6	136.6	142.9

From Table 11, we see that the company spends 140.7 hours per week (on average) to produce its demand. Also, we realize that even if the demand is increased by 10%, IP2/IP1 schedules can handle the increased demand (on average) with a smaller makespan (136.6 hours). There are, however, a few instances where the IP2/IP1 makespan for increased demand exceeds the company's current makespan values. For example, when demand increases by 5%, the company's scheduled makespan for week 6 is shorter than that of IP2/IP1. Similarly, for a 10% increase in demand, there are two instances (out of seven) with higher IP2/IP1 makespan values. Overall, when demand increases up to 15%, IP2/IP1 schedules are shorter than the company's in 14 of 21 instances. Another way

to think about these numbers is that, when using IP2/IP1, 78% (11/14) of the time, Beyçelik can increase its production (capacity) by 10%; and 66% (14/21) of the time, it can increase its production (capacity) by 15%. These capacity increases are significant and can translate into large financial savings, increased flexibility, and higher levels of competitiveness.

6.2. Relationship between demand and makespan

It is important to know the relationship between demand and makespan for planning purposes. With this information, the company can predict the change in makespan due to a potential change in demand and plan accordingly. Furthermore, the company can estimate how much capacity can be increased without exceeding the limit of makespan in a week, i.e., it can estimate the highest production capacity of the company. Therefore, we investigate whether there is a relationship between demand and makespan by changing the real-world demands in certain rates. We change the demands of seven weeks gradually up and down by 5%-25% and create 10 different instances (scenarios) for each of seven weeks. In order to observe the change in C_{max}, we solve these problems with the IP2/IP1 model and present the solutions with different demands as D_5%, D_10%, D_15%, D_20%, and D_25% for increase and decrease, in Tables 12 and 13, respectively.

Table 12 IP2/IP1 solutions of problems with increased demand

Week	# of Jobs	C_{max}(hr)					
		IP2/IP1	IP2/IP1 D_5%	IP2/IP1 D_10%	IP2/IP1 D_15%	IP2/IP1 D_20%	IP2/IP1 D_25%
1	63	124	128	132	136	144	148
2	65	124	128	128	140	140	148
3	60	124	132	136	136	144	148
4	69	136	136	144	152	156	160
5	60	128	136	140	148	152	156
6	59	132	140	140	148	152	156
7	59	120	128	136	136	140	144
	Avg.	126.9	132.6	136.6	142.3	144	148

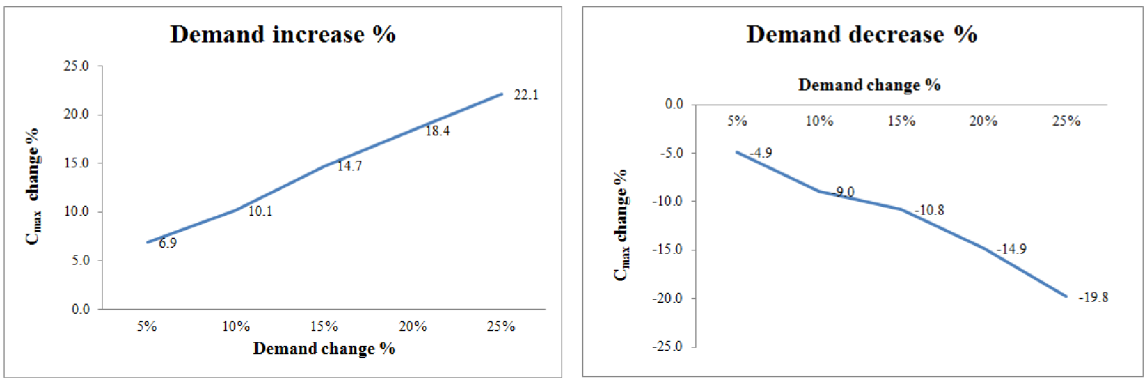
Table 13 IP2/IP1 solutions of problems with decreased demand

Week	# of Jobs	C_{max}(hr)					
		IP2/IP1	IP2/IP1 D_5%	IP2/IP1 D_10%	IP2/IP1 D_15%	IP2/IP1 D_20%	IP2/IP1 D_25%
1	63	124	120	116	116	108	100
2	65	124	116	112	108	104	96
3	60	124	120	112	108	104	96
4	69	136	128	120	120	116	108
5	60	128	120	116	116	108	108
6	59	132	124	120	116	112	104
7	59	120	116	112	108	104	100
	Avg.	126.9	120.6	115.4	113.1	108.0	101.7

To find a relationship between the change in the overall demand and the change in the makespan, we calculate the percentage change of C_{max} by comparing the new results with the IP2/IP1 solutions

for weeks with real-world demands, and we find the average C_{max} change for each demand change, as in Figure 6. Figure 6 shows that there is a positive and (almost) linear relationship between the change in demand and makespan.

Figure 6 **The relationship between demand and makespan**



7. Conclusion

We investigate a real-world scheduling problem in an automotive stamping company. Important parameters include the number of jobs per week, the number of operations for each job, the machines capable of performing each job, and the weekly demands. Considering the practices and conventions of the company, we develop an integer programming model and compare it with a constraint programming model from the literature (Caglar Gencosman et al. 2014). Our proposed method's solution quality is high, but it has a long run time and usually uses the full time limit to find a solution. Thus, to speed up its solution, we develop a second solution method by dividing the Stamping Scheduling Problem into two parts: (1) a job-machine assignment problem (master problem) and (2) a sequencing problem (sub-problem). The second method combines two integer programming models for the master problem and the sub-problem.

We model the production system in detail, taking care to include characteristics of the company's existing process (e.g., planned interruptions and breakdowns are taken into account in our model). These considerations enable company managers to feel confident that the system is modelled accurately and that the model closely represents Beyçelik's reality.

We compare the solutions of the models with the company's generated and executed schedules. Also, we use the second method to reschedule for any interruptions, such as a machine breakdown or a change in demand. Our results show that the second method is able to find schedules within minutes, and these revised schedules have a 6.1% shorter makespan (on average) than

the company's, which means that by using the second method, the company can decrease the production cost of its pressline by 6.1%.

Our additional demand analysis shows that, compared to its existing schedules, the company can process up to 15% more demand with the use of the second method. This increase uses only different scheduling tools; physical changes are not required. The company is currently producing its schedules manually, a process that takes about 28 hours per week. Our second method reduces the scheduling time of the company's current practice from days to minutes, which in turn decreases labour costs. (We did not include this latter cost in our cost-saving calculation.)

This paper provides successful modelling of a real-world scheduling problem. In our model, we are able to reduce the problem into two smaller ones and solve both of them efficiently, showing that the second method is able to generate high-quality schedules in a short time. This result not only provides savings to the company but also gives the company the ability to quickly reschedule its production runs whenever a breakdown of a machine or a fluctuation in demand occurs. The next step involves the implementation of the second method at Beyçelik. The manager of Beyçelik's R&D department has provided a letter of support to accompany this paper, indicating that the company was pleased with the results of our project and that Beyçelik plans to implement our study as a customized software program. We believe that the findings and models of this paper are sufficiently generic to be used in other companies in this industry and in other industries similar to it.

Acknowledgements

We would like to thank Beyçelik for giving us the opportunity to work on such an interesting problem. They not only gave us a detailed description of their operations and the corresponding data, but also provided us with a daily contact person for our questions and data needs. We had opportunities for site visits and countless information and data gatherings. We appreciate their continuous support and guidance throughout the project.

We are grateful to the departmental editor for his suggestions and insights about the content and presentation of the paper as well as the complexity proof. We express our appreciation to the senior editor for his constructive insights on the paper. We also would like to thank the referees for their helpful recommendations and comments. The paper has been significantly improved in terms of content and presentation as a result of the reviewing team's efforts and feedback.

References

Barlatt, A. Y., A. M. Cohn, O. Gusikhin. 2010. A hybridization of mathematical programming and dominance-driven enumeration for solving shift-selection and task-sequencing problems. *Computers & Operations Research,* **37**(7) 1298–1307.

Barlatt, A. Y., A. M. Cohn, O. Gusikhin, Y. Fradkin, R. Davidson, J. Batey. 2012. Ford motor company implements integrated planning and scheduling in a complex automotive manufacturing environment. *Interfaces,* **42**(5) 478–491.

Behnke, D., M. J. Geiger. 2012. Test instances for the flexible job shop scheduling problem with work centers. http://www.nbn-resolving.de/urn:nbn:de:gbv:705-opus-29827 (accessed date May 13, 2016).

Caglar Gencosman, B., H.C. Ozmutlu, H. Ozkan, M.A. Begen. 2014. Automotive stamping scheduling using mixed integer programming and constraint programming. *Industrial Engineering Non-Traditional Applications in International Settings*. CRC Press, New York, 37–68.

Camargo, V. C. B., F. M. B. Toledo, B. Almada-Lobo. 2014. Hops-hamming-oriented partition search for production planning in the spinning industry. *European Journal of Operational Research,* **234**(1) 266–277.

Cambazard, H., P. E. Hladik, A. M. Déplanche, N. Jussien, Y. Trinquet. 2004. Decomposition and learning for a hard real time task allocation problem. *Principles and Practice of Constraint Programming-CP 2004*. Springer, Berlin Heidelberg, 153–167.

Canto, S. P. 2008. Application of benders' decomposition to power plant preventive maintenance scheduling. *European Journal of Operational Research,* **184**(2) 759–777.

Chen, B., C. N. Potts, G. J. Woeginger. 1998. A review of machine scheduling: Complexity, algorithms and approximability. *Handbook of Combinatorial Optimization*. Kluwer Academic, Berlin, 1493–1641.

Coban, E., J. N. Hooker. 2010. Single-facility scheduling over long time horizons by logic-based benders decomposition. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. Springer, Berlin Heidelberg, 87–91.

Edis, E. B., I. Ozkarahan. 2012. Solution approaches for a real-life resource-constrained parallel machine scheduling problem. *The International Journal of Advanced Manufacturing Technology,* **58**(9-12) 1141–1153.

Fazel-Zarandi, M. M., J. C. Beck. 2009. Solving a location-allocation problem with logic-based benders' decomposition. *Principles and Practice of Constraint Programming-CP 2009*. Springer, Berlin Heidelberg, 344–351.

Garey, M. R., D. S. Johnson. 1979. *Computers And Intractability: A Guide to the Theory of NP-Completeness*. WH Freemann, New York.

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

Geismar, H. N., M. Dawande, B. P. S. Murthi, C. Sriskandarajah. 2015. Maximizing revenue through two-dimensional shelf-space allocation. *Production and Operations Management,* **24**(7) 1148–1163.

Gilmore, P.C., R. E. Gomory. 1965. Multistage cutting stock problems of two and more dimensions. *Operations Research,* **13**(1) 94–120.

Gonzalez, T., S. Sahni. 1978. Flowshop and jobshop schedules: complexity and approximation. *Operations Research,* **26**(1) 36–52.

Harjunkoski, I., I. E. Grossmann. 2002. Decomposition techniques for multistage scheduling problems using mixed-integer and constraint programming methods. *Computers & Chemical Engineering,* **26**(11) 1533–1552.

Heinz, S., J. C. Beck. 2012. Reconsidering mixed integer programming and mip-based hybrids for scheduling. *Integration of AI and OR Techniques in Contraint Programming for Combinatorial Optimization Problems*. Springer, Berlin Heidelberg, 211–227.

Hooker, J. N. 2005. Planning and scheduling to minimize tardiness. *Principles and Practice of Constraint Programming-CP 2005*. Springer, Berlin Heidelberg, 314–327.

Hooker, J. N. 2007. Planning and scheduling by logic-based benders decomposition. *Operations Research,* **55**(3) 588–602.

Hooker, J. N., H. Yan. 1995. Logic circuit verification by benders decomposition. *Principles and Practice of Constraint Programming: The Newport Papers,* 267–288.

Jain, V., I. E. Grossmann. 2001. Algorithms for hybrid milp/cp models for a class of optimization problems. *Journal on Computing,* **13**(4) 258–276.

Khayat, G. E., A. Langevin, D. Riopel. 2006. Integrated production and material handling scheduling using mathematical programming and constraint programming. *European Journal of Operational Research,* **175**(3) 1818–1832.

Lawler, E. L., J. K. Lenstra, A. H. G. Rinnooy Kan, D. B. Shmoys. 1993. Sequencing and scheduling: Algorithms and complexity. *Handbooks in Operations Research and Management Science: Logistics of Production and Inventory*. North Holland, Amsterdam, 445–522.

Lodi, A., S. Martello, D. Vigo. 1999. Approximation algorithms for the oriented two-dimensional bin packing problem. *European Journal of Operational Research,* **112**(1) 158–166.

Martello, S., M. Monaci, D. Vigo. 2003. An exact approach to the strip-packing problem. *Journal on Computing,* **15**(3) 310–319.

Mastrolilli, M., O. Svensson. 2009. Improved bounds for flow shop scheduling. *Automata, Languages and Programming*. Springer, Berlin Heidelberg, 677–688.

Mastrolilli, M., O. Svensson. 2011. Hardness of approximating flow and job shop scheduling problems. *Journal of the ACM,* **58**(5) 20.

Meyr, H., M. Mann. 2013. A decomposition approach for the general lotsizing and scheduling problem for parallel production lines. *European Journal of Operational Research,* **229**(3) 718–731.

Pinedo, M. 2012. *Scheduling: Theory, Algorithms, and Systems*. Springer, New York.

Relvas, S., S. N. B. Magatão, A. P. F. Barbosa-Póvoa, F. Neves. 2013. Integrated scheduling and inventory management of an oil products distribution system. *Omega,* **41**(6) 955–968.

Tran, T. T., J. C. Beck. 2012. Logic-based benders decomposition for alternative resource scheduling with sequence dependent setups. *In Proceedings of the 20th European Conference on Artificial Intelligence (ECAI)*. Montpellier, France.

Waldherr, S., S. Knust. 2015. Complexity results for flow shop problems with synchronous movement. *European Journal of Operational Research,* **242**(1) 34–44.

Wäscher, G., H. Haußner, H. Schumann. 2007. An improved typology of cutting and packing problems. *European Journal of Operational Research,* **183**(3) 1109–1130.

Xiaolu, L., B. Baocun, C. Yingwu, Y. Feng. 2014. Multi satellites scheduling algorithm based on task merging mechanism. *Applied Mathematics and Computation,* **230** 687–700.

Online Appendix for "Scheduling Methods for Efficient Stamping Operations at an Automotive Company"

A The Constraint Programming Model: CP1

We present the constraint programming model CP1 for the Stamping Scheduling Problem given in Caglar Gencosman et al. (2014) using the OPL language and the *Scheduler* module of constraint programming in ILOG CPLEX 12.5. (We give the additional notation needed for CP1 only.)

Table 14 Additional sets, parameters, and variables for CP1

Index:	
j	Index for operations.
Sets:	
Q_i	The set of operations for job i. Q_i={1,2,...,$Q_{i_{max}}$}.
$Alter$	$\{(i,m) \mid i \in I, m \in M$: A_{im}=1$\}$.
$Curr$	$\{(i,j) \mid i \in I, j \in Q_i\}$.
Decision Variables:	
y_i	interval variable for job i, *size* J_i.
e_{ij}	interval variable for operation j of job i *in* 0..$3600 * K_{max} * U$.
x_{im}	interval variable for alternative machines m of job i $(i,m) \in Alter$; *optional* in 0..3600*$K_{max} * U$.
g_m	$\sum_{(i,j) \in Curr}$ pulse(e_{ij},1), $\forall m \in M$.

$$\text{minimize}\, max_i(endOf(y_i)) \tag{19}$$

$$endBeforeStart(e_{ij}, e_{ij+1}) \;\; \forall i \in I, j \in Q_i \mid j+1 \leq Q_{i_{max}} \tag{20}$$

$$presenceOf(e_{ij}) \overset{yields}{\rightarrow} presenceOf(e_{ij+1}) \;\; \forall i \in I, j \in Q_i \mid j+1 \leq Q_{i_{max}} \tag{21}$$

$$span(y_i, all(j \in Q_i) e_{ij}) \;\; \forall i \in I \tag{22}$$

$$alternative(y_i, all(a \in Alter : a.i = i) x_a) \;\; \forall i \in I \tag{23}$$

$$noOverlap(all(a \in Alter : a.m = m) x_a) \;\; \forall m \in M \tag{24}$$

$$\begin{pmatrix} presenceOf(x_a) > 0 \\ AND \\ presenceOf(x_b) > 0 \end{pmatrix} \overset{yields}{\rightarrow} \begin{pmatrix} endOf(x_a) \leq startOf(x_b) \\ OR \\ endOf(x_b) \leq startOf(x_a) \end{pmatrix}$$

$$\forall a, b \in Alter \mid Q_{a.i_{max}} > 1 \, AND \, b.i \neq a.i \, AND \, b.m \geq a.m \, AND \, b.m \leq a.m + Q_{a.i_{max}} - 1 \tag{25}$$

$$\sum_{m \in M} g_m \leq M_{max} \tag{26}$$

The objective function of CP1 is to minimize the maximum completion time, represented with the *endOf* term in CP, as in expression (19). Constraint (20) ensures that the start time of operations must be sequential, and this is achieved with the *endBeforeStart* term. Constraint (21) connects the operations of a job together with the *presenceOf* construct. In constraint (22), the *span* expression says that a job must cover the start and end times of its operations. Constraint (23) guarantees that each job can be assigned only to one of its alternative machines. In constraint (24), the *noOverlap* term blocks the assignment of multiple jobs to one machine at the same time. Constraint (25) makes sure that if job a is assigned to machine m, then no other jobs can be assigned until the machine's required number of operations have been completed. Finally, the cumulative usage of machines is restricted by constraint (26) with the number of machines.

Caglar Gencosman et al. (2014) test a few search methods and evaluate their performances. The authors compare these methods considering solution points, number of branches, and choice points. Their results show that these methods do not perform any better than the default automatic search algorithm of the software.

B MinK_{max} Algorithm

We develop an algorithm (MinK_{max}) to find the smallest K_{max} value that makes IP1 feasible to determine the boundary of production horizon efficiently. (Note that K_{max} represents the last period that a job can start processing in our models, and, hence, it does not represent the makespan value.) The algorithm starts with two K_{max} estimates: one *lower bound (lb)* and one *upper bound (ub)*. Then, it chooses a K_{max} value based on these bounds and determines the feasibility of an instance by using an integer programming model IP_K_{max}. The IP_K_{max} model is almost identical to IP1, except for its objective. We run IP_K_{max} until the first feasible solution is found repetitively, and we update the *lb* and *ub* values of K_{max} at each iteration. (Since we are interested in the first feasible solution only, we set the objective of IP_K_{max} to a constant value, zero.) The IP_K_{max} model is given below.

$$\text{minimize } 0 \tag{27}$$

subject to

$$\sum_{i\in I} A_{im}x_{ikm} \leq 1 \quad \forall k \in K, m \in M \tag{28}$$

$$\sum_{k\in K}\sum_{m\in M} A_{im}x_{ikm} = 1 \quad \forall i \in I \tag{29}$$

$$A_{im}x_{ikm}(H_i + k - 1) \leq K_{max} \quad \forall i \in I, k \in K, m \in M \tag{30}$$

$$A_{tn}x_{tzn} \leq 1 - A_{im}x_{ikm} \quad \forall i,t \in I, \forall k,z \in K, \forall m,n \in M \mid t \neq i, k \leq z \leq k + H_i - 1, n \leq m + O_i - 1, m \leq n + O_t - 1 \tag{31}$$

$$x_{ikm} \in \{0,1\} \quad \forall i \in I, k \in K, m \in M \tag{32}$$

We use three variables in the MinK_{max} algorithm: the *upper bound (ub)*, the *lower bound (lb)*, and the *current value (cv)* of K_{max}. We first find the average number of operations (O_{avg}) and the average number of periods (J_{avg}) of all jobs in a given problem instance. In addition, we find the total number of jobs (I_{max}) and the total number of machines (M_{max}) in the problem instance.

We consider the smallest integers that are smaller than or equal to the O_{avg} and J_{avg}, and we define the default value of *lb* as $lb = (I_{max} * \lfloor O_{avg} * J_{avg} \rfloor)/M_{max}$. We next consider the smallest integers that are greater than or equal to the O_{avg} and J_{avg}, and we define the default value of *ub* as $ub = (I_{max} * \lceil O_{avg} * J_{avg} \rceil)/M_{max}$. Thus, we define the default values of variables based on the problem instance.

We provide the algorithm in Table 15. The algorithm starts with testing the *lb* and *ub* values. We need to guarantee at least one solution between these boundaries; hence, we define the first *While*

loop to test the current boundaries and to determine an appropriate ub value. If the algorithm finds a solution with the current ub value, the default value of cv is set as $cv = \lceil ub + lb/2 \rceil$, and the algorithm continues to find the best K_{max} value with the second *While* loop. On the other hand, if the algorithm could not find a solution with the current ub, the ub value is increased as $ub = \lceil ub * 1.25 \rceil$, the cv is set as $cv = ub$, and the problem is solved again. If the algorithm finds a solution after increasing the upper bound, the cv is updated as $cv = \lceil ub + lb/2 \rceil$, and the first *While* loop is terminated to continue with the second *While* loop. The algorithm continues to update the boundaries depending on the feasibility of the outcome, and the process is repeated until the gap between lb and ub is less than 2 ($ub - lb < 2$), or until the time limit of 900 seconds is reached.

Table 15 The MinK_{max} Algorithm

$lb = (I_{max} * \lfloor O_{avg} * J_{avg} \rfloor)/M_{max}$
$ub = (I_{max} * \lceil O_{avg} * J_{avg} \rceil)/M_{max}$
While (Inf=1) {
  Solve IP_K_{max}
  If the solution is feasible Then {
    $cv = \lceil ub + lb/2 \rceil$
    Inf=0 }
  else{
    $ub = \lceil ub * 1.25 \rceil$
    $cv = ub$ }
  If(Elapsed Time>900) Then {
    Stop = 1
    Inf = 0}
  }
While (Stop=0) {
  Solve IP_K_{max}
  If the solution is feasible Then {
    If($ub - lb < 2$) Then {Stop = 1}
    $ub = cv$
    $cv = \lceil (ub + lb)/2 \rceil$ }
  else{
    $lb = cv$
    $cv = \lceil (lb + ub)/2 \rceil$ }
  If(Elapsed Time>900) Then {Stop = 1}
  If($ub - lb < 2$) Then {Stop = 1}
  }
End

C Alternative Versions of Constraint (5)

We developed alternative versions of Constraint (5). (Recall that this constraint prevents overlapping of two different jobs.)

We first give a definition for a condition used in all versions and then give the different constraints.

C_0: Job t is different from job i, and job t starts before job i finishes.

- **Alternative 1 (A1):**

  — C_1: Job t cannot be scheduled if it starts at the same machine with job i,

  — C_2: Job t cannot be scheduled if it finishes before job i finishes,

  — C_3: Job t cannot be scheduled if it starts before job i starts.

$$\begin{array}{l} A_{tn}x_{tzn} \leq 1 - A_{im}x_{ikm} \;\; \forall i,t \in I, \forall k,z \in K, \forall m,n \in M \mid \\ C_0 \, AND \, (C_1 \, OR \, C_2 \, OR \, C_3) \;\; where \\ C_0 : t \neq i \, AND \, k \leq z \leq k + H_i - 1, \\ C_1 : m = n, \\ C_2 : m \leq n + O_t - 1 \leq m + O_i - 1, \\ C_3 : n \leq m + O_i - 1 < n + O_t - 1. \end{array} \tag{33}$$

- **Alternative 2 (A2):**

$$\begin{array}{l} A_{tn}x_{tzn} \leq 1 - A_{im}x_{ikm} \;\; \forall i,t \in I, \forall k,z \in K, \forall m,n \in M \mid \\ (C_0 \, AND \, C_1) \, OR (C_0 \, AND \, C_2) \, OR (C_0 \, AND \, C_3) \;\; where \\ C_0 : t \neq i \, AND \, k \leq z \leq k + H_i - 1, \\ C_1 : m = n, \\ C_2 : m \leq n + O_t - 1 \leq m + O_i - 1, \\ C_3 : n \leq m + O_i - 1 < n + O_t - 1. \end{array} \tag{34}$$

- **Alternative 3 (A3):**

  — D_1: Job t cannot finish during processing of job i,

  — D_2: Job t cannot start during processing of job i,

  — D_3: Job t cannot start before job i starts and it cannot finish after the completion time of job i.

$$\begin{array}{l} A_{tn}x_{tzn} \leq 1 - A_{im}x_{ikm} \;\; \forall i,t \in I, \forall k,z \in K, \forall m,n \in M \mid \\ (C_0 \, AND \, D_1) \, OR (C_0 \, AND \, D_2) \, OR (C_0 \, AND \, D_3) \;\; where \\ C_0 : t \neq i \, AND \, k \leq z \leq k + H_i - 1, \\ D_1 : m \leq n + O_t - 1 \leq m + O_i - 1, \\ D_2 : m \leq n \leq m + O_i - 1, \\ D_3 : n < m \, AND \, m + O_i - 1 < n + O_t - 1. \end{array} \tag{35}$$

- **Alternative 4 (A4):**

  — E_1: Job t cannot be scheduled if it starts before job i finishes,

  — E_2: Job t cannot be scheduled if it finishes after job i starts.

$$
\begin{aligned}
&A_{tn}x_{tzn} \leq 1 - A_{im}x_{ikm} \quad \forall i,t \in I, \forall k,z \in K, \forall m,n \in M \mid \\
&(C_0 \; AND \; E_1 \; AND \; E_2) \\
&C_0 : t \neq i \; AND \; k \leq z \leq k + H_i - 1, \\
&E_1 : n \leq m + O_i - 1, \\
&E_2 : m \leq n + O_t - 1.
\end{aligned} \tag{36}
$$

We ran IP1 with the four alternative versions of Constraint (5) with the same dataset. The results are given in Table 16. Results show that A4 is slighlty faster, and hence, we chose A4 for our models.

Table 16 Comparison of elapsed times with four alternative versions of constraint (5)

#of Jobs	Problem	K_{max} (per.)	C_{max}(per.)				Elapsed Time(s)			
			A1	A2	A3	A4	A1	A2	A3	A4
50	1	24	24	24	24	24	49.9	48.7	48.3	**41.4**
	2	26	26	26	26	26	56.4	56.0	54.1	**47.1**
	3	24	24	24	24	24	72.0	72.0	71.8	**64.3**
55	1	31	31	31	31	31	86.7	86.3	83.9	**69.0**
	2	28	28	28	28	28	230.2	231.5	229.3	**220.6**
	3	27	27	27	27	27	344.6	345.7	341.5	**333.1**
60	1	31	31	31	31	31	121.1	122.1	121.8	**104.4**
	2	29	29	29	29	29	538.6	537.0	537.8	**520.4**
	3	31	32	32	32	32	900.0	900.0	900.0	900.0
65	1	33	33	33	33	33	900.0	900.0	900.0	900.0
	2	32	33	33	33	33	900.0	900.0	900.0	900.0
	3	35	35	35	35	35	895.2	890.7	**870.5**	889.8
70	1	34	35	35	35	35	900.0	900.0	900.0	900.0
	2	37	38	38	38	38	900.0	900.0	900.0	900.0
	3	36	39	39	39	39	900.0	900.0	900.0	900.0
						Avg.	519.6	519.3	517.3	**512.7**

D The Proof of NP-hardness

For completeness, we restate the Theorems before giving their proofs.

Theorem 1 *Stamping Scheduling Problem is strongly NP-hard if the number of machines in the input is a variable.*

Proof: We choose the 3-Partition problem (Garey and Johnson (1979)) for our reduction.

3-Partition Instance: Given positive integers s, B and a set of integers $A=\{a_1,a_2,...a_{3s}\}$ with $\sum_{i=1}^{3s} a_i = sB$ and $B/4 < a_i < B/2$ for $1 \le i \le 3s$, does there exist a partition of A into three element sets $\{A_1,A_2,...,A_s\}$ such that $\sum_{a_j \in A_i} a_j = B, 1 \le i \le s$?

3-Partition Solution: A partition of A into three element sets $\{A_1,A_2,...,A_s\}$ such that $\sum_{a_j \in A_i} a_j = B$ for $1 \le i \le s$.

Given an arbitrary instance of 3-Partition, we construct the following instance of the Stamping Scheduling Problem.

- The number of machines is $M_{max} = 2s-1$.
- The number of jobs is $I_{max} = 4s$.
- The notation $J_i(m,o,p)$ indicates a job i $(1 \le i \le 4s)$ which starts on machine m, requiring o consecutive operations for processing time of p.
- We define three job types.
  - —Job α, a single job, with $J_\alpha(1,2s-1,B)$.
  - —Job β_j, $s-1$ jobs $(1 \le j \le s-1)$, with $J_{\beta_j}(2j,1,B)$.
  - —Job γ_j, $3s$ jobs $(1 \le j \le 3s)$, with $J_{\gamma_j}(x,1,a_j)$ where $x \in \{1,3,5,...,2s-1\}$; i.e., job γ_j requires one operation with processing duration of a_j and can be processed on any machine with an odd index.
- The makespan, T.

For the instance of the Stamping Scheduling Problem constructed above, we consider the following decision problem:

Decision Problem: Does there exist a feasible schedule σ with the makespan $T(\sigma) \le 2B$?

Let σ be a schedule of $4s$ jobs, where $\sigma=(\sigma_1,\sigma_2,...,\sigma_{4s})$, and the first component of σ denotes start time of job $J_\alpha(1,2s-1,B)$; the next $s-1$ entries in σ denotes start time of jobs $J_{\beta_j}(2j,1,B)$ $(j=1,2,...,s-1)$; the last $3s$ entry in σ denotes start time of jobs $J_{\gamma_j}(x,1,a_j)$ $(x \in \{1,3,5,...,2s-1\}$ and $j=1,2,...,3s)$.

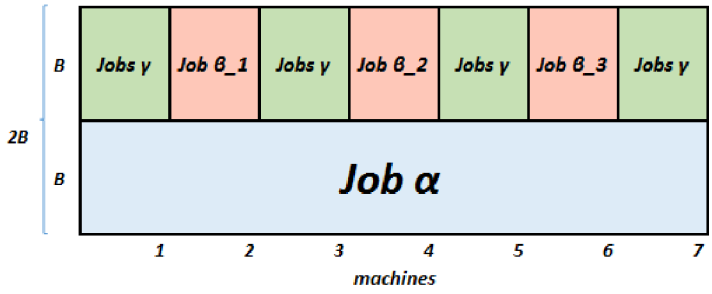
If part: Suppose there exists a partition of A into three element sets $\{A_1,A_2,...,A_s\}$ such that $\sum_{a_j \in A_i} a_j = B, 1 \le i \le s$. Assume that $A_1=\{a_1,a_2,a_3\}, A_2=\{a_4,a_5,a_6\},...,A_j=\{a_{3j-2},a_{3j-1},a_{3j}\},...,A_s=\{a_{3s-2},a_{3s-1},a_{3s}\}$. Note that $a_{3k-2}+a_{3k-1}+a_{3k}=B$ for $k=1,2,...,s$.

We generate schedule σ as follows:

- Job α, $J_\alpha(1, 2s-1, B)$, is scheduled to start at time zero on machine 1; i.e., $\sigma_1 = 0$.
- Job β_j, $J_{\beta_j}(2j, 1, B)$ $(1 \le j \le s-1)$, is scheduled to start at time B on machine $2j$; i.e, $\sigma_2 = \sigma_3 = ... = \sigma_s = B$.
- For γ-type of jobs we implement the following: schedule the jobs in A_i $(1 \le i \le s)$, that is jobs $\gamma_{3i-2}, \gamma_{3i-1}, \gamma_{3i}$, on machine $2i-1$ at times $(B, B + a_{3i-2}, B + a_{3i-2} + a_{3i-1})$; i.e., $\sigma_{s+1} = B, \sigma_{s+2} = B + a_1, \sigma_{s+3} = B + a_1 + a_2, \sigma_{s+4} = B, \sigma_{s+5} = B + a_4, \sigma_{s+6} = B + a_4 + a_5, ..., \sigma_{3i+s-2} = B, \sigma_{3i+s-1} = B + a_{3i-2}, \sigma_{3i+s} = B + a_{3i-2} + a_{3i-1}, ..., \sigma_{4s-2} = B, \sigma_{4s-1} = B + a_{3s-2}, \sigma_{4s} = B + a_{3s-2} + a_{3s-1}$.

The makespan of schedule σ can easily be verified to be $2B$ and hence $T(\sigma) \le 2B$. See Figure 7 with an instance of $s = 4$.

Figure 7 **If Part: Schedule σ correspond to 3-Partition solution for $s = 4$**

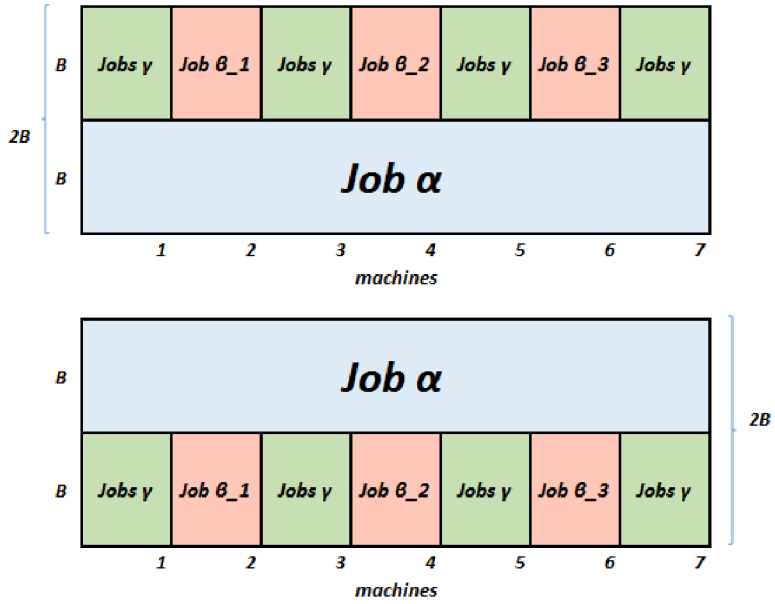


Only if part: Suppose there exists a schedule σ to the decision problem with $T(\sigma) \le 2B$. We need to prove that there exists a solution to the 3-Partition problem.

If there is a feasible schedule with a makespan less or equal to $2B$, then the makespan value must be exactly $2B$ since processing of job α and any of the β-type jobs would yield a makespan greater or equal to $2B$. Hence, the makespan value must be $2B$. Due to α and β jobs there are two possibilities for the schedule σ. Job α is either scheduled to start processing at time zero on machine 1 or it starts processing at time B on machine 1. See Figure 8 with an instance of $s = 4$.

Suppose job α starts processing at time zero on machine 1, then job β_j $(1 \le j \le s-1)$ must be scheduled to start at time B on machine $2j$, i.e, $J_{\beta_j}(2j, 1, B)$ $(1 \le j \le s-1)$. This leaves space of B processing time on each of the machines $1, 3, ..., 2s-1$ (i.e., on s machines) for $3s$ many γ-type of jobs. We know that the total processing time of all γ-type of jobs is sB; i.e., $\sum_{i=1}^{3s} a_i = sB$. Since we have a feasible schedule with a makespan of $2B$, there cannot be any idle time on any of the machines $1, 3, ..., 2s-1$; otherwise, idle time on any machine would imply some overtime in another machine and, thus, the makespan will be strictly greater than $2B$. Therefore, $3s$ many γ-type jobs must be scheduled on s machines without any idle time for a processing time of B. Since each γ

Figure 8 **Only If Part: Two possible schedules presenting a solution to the 3-Partition problem with $s=4$**



type job size is strictly greater than $B/4$, the number of jobs processed on machines $1, 3, ..., 2s-1$ can not be four or greater. Similarly, since each γ job size is strictly less than $B/2$, the number of jobs processed on machines $1, 3, ..., 2s-1$ cannot be two or less. Therefore, we must have precisely three γ jobs on each machine $1, 3, ..., 2s-1$ with a total processing time of B. Thus, a solution to the 3-Partition problem exists.

If job α does not start processing at time zero, then it starts at time B. In this case, β and γ type jobs are scheduled at time zero. The proof for this case is similar to the above case and omitted.

Theorem 2 *Stamping Scheduling Problem is NP-hard in the ordinary sense if the number of machines is fixed.*

Proof: We choose the 2-Partition problem (Garey and Johnson 1979) for our reduction.

2-Partition Instance: Given positive integers s, B and a set of integers $A=\{a_1, a_2, ...a_s\}$ with $\sum_{i=1}^{s} a_i = 2B$, does there exist a partition of A into two subsets A_1 and A_2 such that $A_1 \bigcap A_2 = \emptyset$ and $\sum_{a_j \in A_1} a_j = \sum_{a_k \in A_2} a_k = B$?

Solution: Find a partition of A into two subsets A_1 and A_2 such that $A_1 \bigcap A_2 = \emptyset$ and $\sum_{a_j \in A_1} a_j = \sum_{a_k \in A_2} a_k = B$.

Given an arbitrary instance of 2-Partition, we construct the following instance of the Stamping Scheduling Problem.

- The number of machines is $n=5$. Thus the machine set is $\{M_1, M_2, ..., M_5\}$.
- The number of jobs is $I = s+7$.

- The notation $J_j(m,o,p)$ indicates a job j $(1 \le j \le s+7)$ which starts on machine m, requiring o consecutive operations for processing time of p. We define the jobs as below.
- Seven jobs: $J_1(1,1,3B)$, $J_2(1,3,B)$, $J_3(5,1,3B)$, $J_4(3,3,B)$, $J_5(2,1,2B)$, $J_6(2,3,B)$, and $J_7(3,1,B)$.
- Job a_i, s jobs $(1 \le j \le s)$, with $J_{a_i}(4,1,a_i)$.
- The makespan, T.

For the instance of the Stamping Scheduling Problem constructed above, we consider the following decision problem:

Decision Problem: Does there exist a feasible schedule σ with the makespan $T(\sigma) \le 4B$?

Let σ be a schedule of $s+7$ jobs, where $\sigma = (\sigma_1, \sigma_2, ..., \sigma_{s+7})$, and the first 7 entry in σ denotes the start time of jobs $J_1, J_2, ..., J_7$, the next s entry in σ denotes start time of jobs $J_{a_i}(4,1,a_i)$, $(i = 1,2,...,s)$.

If part: Suppose there exists a partition of A into two subsets A_1, and A_2 such that $A_1 \bigcap A_2 = \emptyset$ and $\sum_{a_j \in A_1} a_j = \sum_{a_k \in A_2} a_k = B$ where $A_1 = \{a_1, a_2, ..., a_r\}$, $A_2 = \{a_{r+1}, a_{r+2}, ..., a_s\}$ (Thus, $|A_1| = r$ and $|A_2| = s - r$). We propose schedule σ as follows:

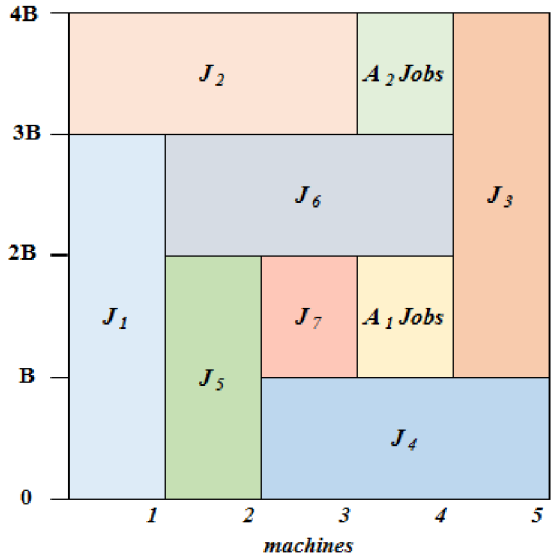
- Jobs J_1, J_4, and J_5 are scheduled to start at zero; i.e., $\sigma_1 = \sigma_4 = \sigma_5 = 0$.
- Jobs J_3 and J_7 are scheduled to start at B; i.e., $\sigma_3 = \sigma_7 = B$.
- Job J_6 is scheduled to start at $2B$; i.e., $\sigma_6 = 2B$.
- Job J_2 is scheduled to start at $3B$; i.e., $\sigma_2 = 3B$.
- For J_a-type jobs we implement the following: schedule the jobs in A_1, that is jobs $a_1, a_2, ..., a_r$, at times $(B, B + a_1, B + a_1 + a_2, ..., B + a_1 + a_2 + ... + a_{r-1})$. Next, schedule the jobs in A_2, that is jobs $a_{r+1}, a_{r+2}, ..., a_s$, at times $(3B, 3B + a_{r+1}, 3B + a_{r+1} + a_{r+2}, ..., 3B + a_{r+1} + a_{r+2} + ... + a_{s-1})$.

In above schedule σ, it is easy to verify that $T(\sigma) = 4B$ (see Figure 9).

Only if part: Suppose there exists a schedule σ to the decision problem with $T(\sigma) \le 4B$. We need to prove that there exists a solution to the 2-Partition problem. To prove this, we follow the steps below:

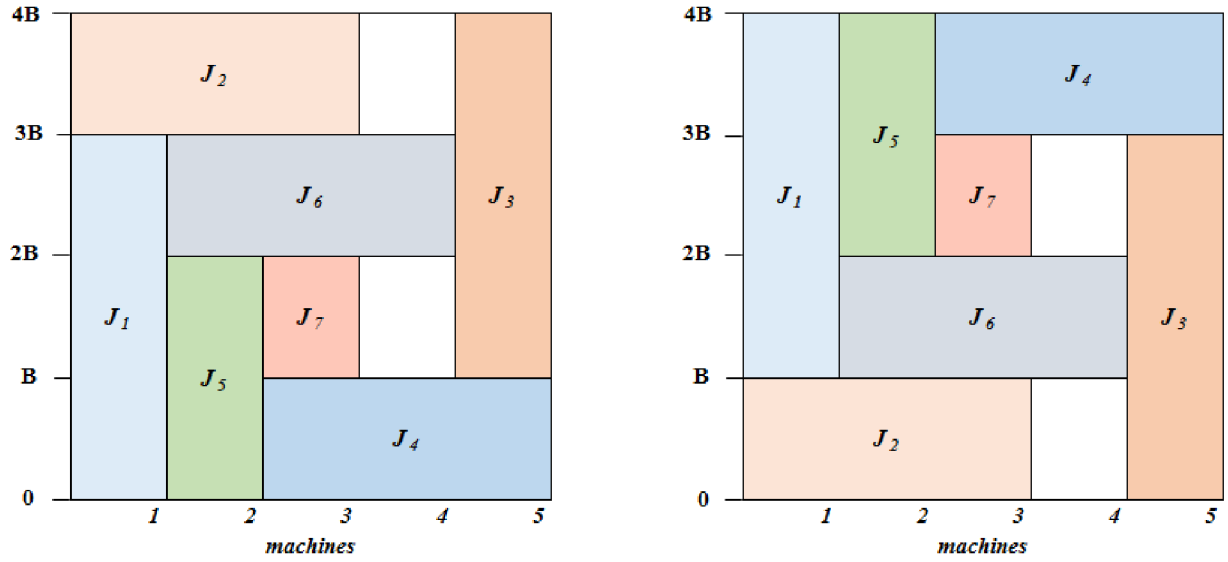
- Assume that $T(\sigma) \le 4B$, then jobs $J_1, J_2, ..., J_7$ must be scheduled in σ as either one of two cases in Figure 10. Thus, $T(\sigma)$ must be equal to $4B$.
- The two possible schedules in Figure 10 create two slots on machine 4 each having B time length.
- Since $T(\sigma) = 4B$ and the sum of the processing time for all J_a-type jobs is $2B$, the two time slots must be filled without any idle time on machine 4 (any idle time on either of the two time slots on machine 4 would result in $T(\sigma) > 4B$).

Figure 9 **If Part: Schedule σ correspond to 2-Partition solution for $n=5$**



- Therefore, the jobs fit in each time slot in each possible schedule must have the total length B. This proves that there exists a solution to the 2-Partition problem.

Figure 10 **Only If Part: Two possible schedules of jobs $J_1, J_2, ..., J_7$ in σ when $T(\sigma) \leq 4B$, where $n=5$**



E Additional Computational Experiments

To investigate the limits of IP1 and IP2/IP1 models, we perform additional experiments with larger problem instances. We generate new instances with the number of jobs varying from 75 to 150 by using the frequencies of parameters' values in Table 7. We limit the MinK_{max} algorithm with 900 seconds, and we terminate the IP1 and IP2/IP1 models after 20 hours. When the MinK_{max} algorithm could not find any solution within the time limits (i.e., for the last three problems with 150 jobs), we use the $IP2_{Cmax}$ value, which can be found in seconds by IP2/IP1.

We compare the performances of models in Table 17. IP1 runs for 8.4 hours for all problems, on average, and it cannot prove the optimality of three problems within 20 hours. On the other hand, the IP2/IP1 model is able to find the optimal solutions ($\underline{IP1}$ and $\overline{IP1}$ values - bounds - are equal) of problems within 1.4 hours on average, which is approximately six times faster than IP1, on average.

Table 17 Comparison of IP1 and IP2/IP1 with randomly generated large instances

# of Jobs	Problem	$MinK_{max}$(per.)		C_{max}(per.)		Elapsed Time(s)		Gap%	
		K_{max}	Elapsed Time	IP1	IP2/IP1	IP1	IP2/IP1	IP1	IP2/IP1
75	1	74	900.0	74	74	72000.0	80.2	1.4	-
	2	82	900.0	82	82	720.4	451.3	-	-
	3	62	819.5	62	62	813.9	390.1	-	-
100	1	113	900.0	111	111	1041.2	879.5	-	-
	2	111	900.0	102	102	12110.6	4475.9	-	-
	3	93	900.0	87	87	72000.0	1329.2	1.2	-
125	1	110	900.0	94	94	15794.6	6731.5	-	-
	2	145	900.0	130	130	72000.0	4266.2	27.7	-
	3	137	900.0	124	124	21858.0	8869.3	-	-
150	1	-	900.0	108	108	63363.8	17725.6	-	-
	2	-	900.0	137	137	17480.8	9922.9	-	-
	3	-	900.0	133	133	12072.1	6814.7	-	-
					Avg.	30104.6	5161.4		