
Production Systems and Information Engineering
Volume 10 (1), pp. 29–47
doi: 10.32968/psaie.2022.1.3.

DATA STRUCTURES AND SIMULATION ALGORITHMS FOR FLOW-SHOP SCHEDULING WITH RESOURCE AVAILABILITY CONSTRAINTS

Levente Fazekas
University of Miskolc, Hungary
Department of Information Engineering
aitflew@uni-miskolc.hu

Gyula Kulcsár
University of Miskolc, Hungary
Department of Information Engineering
iitkgy@uni-miskolc.hu

Abstract. Flow-shop scheduling problems are classic examples of multi-resource and multi-operation optimization problems. This paper demonstrates our model and simulation for solving Flow-shop scheduling optimization problems with resource availability constraints to minimize makespan. Working hours, transfer times and setup times are taken into account. Working hours could be interpreted in two ways. In the first option, operations must fit into a single available time slot. In the second option, the simulation could cut operations across multiple time slots. We propose a simulation algorithm that accounts for both scenarios and could be used with any permutation-based exact, heuristic, or search algorithm.

Keywords: Flow-shop, Constrained Flow-shop, Resource constraints, Scheduling, Optimization, Working hours, Transfer times, Setup times, Makespan

1. Introduction

Scheduling tasks are often within the scope of production information engineering. It is an interdisciplinary subject area in which the laws of the specific systems and processes of the field appear together with the rules, methods, and tools of applied informatics. Production information engineering is the applied informatics field that deals with modeling, planning, and controlling production systems and processes [1].

Scheduling is a form of decision-making that is essential in several areas. It deals with allocating limited available resources between the types of operations to be performed that can be optimized for one or more performance metrics. Depending on the situation, resources and activities can take several different forms. Resources can be hospital nurses, bus drivers, machines in a manufacturing system, processors, and mechanics in a workshop. Activities can include manufacturing operations, tasks of nurses in a hospital, running computer programs, car repair operations in a workshop.

We can also optimize for many different performance metrics. These can be to minimize lead time or the number of late jobs [2].

They have been scheduling in plant management, operations research, management, and computing for over fifty years. To date, a vast amount of knowledge has accumulated in this area. Well-functioning scheduling algorithms can significantly reduce the cost of manufacturing operations, allowing the manufacturer to remain competitive. The scheduling problems studied in the 1950s were relatively simple. Several algorithms for optimal results have been developed. The most prominent of which were developed by Jackson [3, 4], Johnson [5] and Smith [6]. Over time, the problems became more sophisticated, and researchers did not find algorithms that provided optimal solutions to them. Most researchers have tried to develop efficient branching and constraint (BB, B&B, or BnB) algorithms for these, which are exponentially running algorithms. With the development of complexity theory [7, 8, 9], researchers have recognized that these problems are inherently complex to solve. In the 1970s, many problems proved to be NP hard [10, 11, 12, 13].

In the 80s, scheduling took several directions in industry and academia. One trend was the development and analysis of approximation algorithms. Another direction was the growing attention to stochastic scheduling problems. From then on, research in the field of scheduling has accelerated.

2. The extended problem

Graham et. al. [14] introduced the $\alpha|\beta|\gamma$ formal classification scheme. The α field denotes the resource environment. β describes the characteristics and constraints regarding jobs. γ sets the objective functions, for which we optimize [15, 10].

The β field has the following default values:

- Each job can only be at one machine at a time.
- A machine can only process one operation at a time.
- All the machines are available all the time.
- Every job is available right away, the can be started at any time.

- The jobs are independent from each other. They can be finished in any order.
- The operations cannot be interrupted.
- The buffer size between machines is infinite.

Our scheduling problem can be defined as:

$$F, Cal_m|perm, s_{i,j,m}, T^r_{k,l}|C_{max}$$

The meaning of the symbols is the following:

- F – one way, multi-operation, shop level scheduling problem (Flow-shop [5])
- Cal_m – the resources are constrained to a set of working hours.
- $perm$ – the jobs cannot precede one another
- $s_{i,j,m}$ – the resources must be set up between jobs. The setup time is based on the order of jobs.
- $T^r_{k,l}$ – the transfer times between machines cannot be neglected. Transfer times are based on the relative position of resources.
- C_{max} – Minimizing makespan.

The general Flow-shop problem can be formulated as NJ number of jobs that have to be processed on NR number of machines in order. Thus each job consists of NR number of operations $p_{i,j}, i \in \{1, ..., NJ\}, j \in \{1, ..., NR\}$ where i is the index of the job and j is the index of the operation. Each machine can handle only one job at a time and one job can only be processed on one machine at once. We want to find a processing order s such that the time required to complete all operations (the makespan) is minimized $C_{max} \rightarrow min$.

In our more generalized problem, we take the transfer times between machines and setup times between jobs on each machine into account. In our problem the machines are not available at any time, they also have working time-intervals (time slots).

This problem and most of it's sub-problems are NP-hard [16, 17], therefore they cannot be solved with an algorithm that has a polynomial run time and guarantees an optimal solution [18, 15]. The maximal sub-problem that can be solved in polynomial time is

$$F2||C_{max}[5].$$

This problem can always be solved by Johnson's algorithm. The minimally NP-hard sub-problem is

$$F3||C_{max}[8],$$

which can be solved by the Extended Johnson's algorithm in special cases.

3. The data-model of the problem

To change the model of the regular flow-shop problem, we have to store the working hours, the transfer times, and the setup times.

For the working hours, we need to store the start and end times of every slot where the resource is available. There are multiple of these slots for every resource.

The setup times represent configuration and tool changes for every resource between each job. In our model, every resource contains a table. The key for each setup time value is a compound key. It contains the index of the ended and starting jobs. Usually, the setup time value is set to 0 between matching jobs.

A simple solution for storing transfer times is a global, shop-level table in which the key is the index of the two machines and the value is the transfer time between them. Another solution would be to slice this table and distribute it between the resources. In this case, the key becomes the resource index from which the workpiece is transferred.

4. The simulation

The problem's extension does not require any changes in implementing the objective function and the scheduling algorithms. Every time the problem changes, we only have to alter the simulation, and data-model [19]. Alongside simulating the Flow-shop problem, we have to consider the additional criteria we proposed in section 2.

For simulating working hours, we have two options. In the first option, the operations cannot be interrupted. We have to finish them in the available time slot where they can be processed from start to finish. In the second option, operations can be processed across multiple available time slots. They do not have to finish processing in the same slot they are started in. Here, the base schedule is used; there is no idle time, only the downtime is wedged in. Therefore if we take out the downtime, we get the original, non-interrupted schedule back.

Case 1. *Let the number of jobs be* $NJ = 4$*, the number of resources* $NR = 1$*, the set of processing times* $\vec{p} = \{2, 2, 2, 1\}$*, and the schedule set by the search algorithm is* $\vec{s} = \{1, 2, 3, 4\}$*.*

In this case, the Gantt diagram of the original, flow-shop case is:

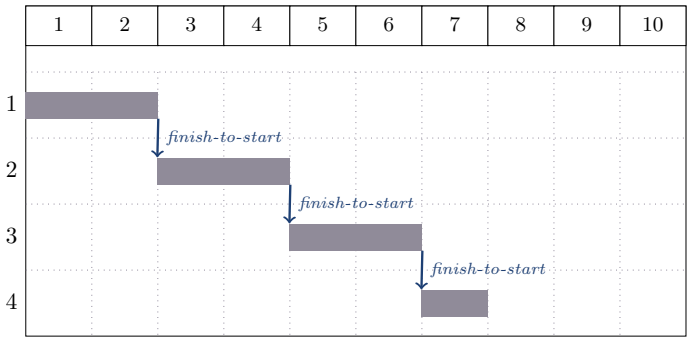

Figure 1. Base case

Changes introduced in the first, non-interrupting case:

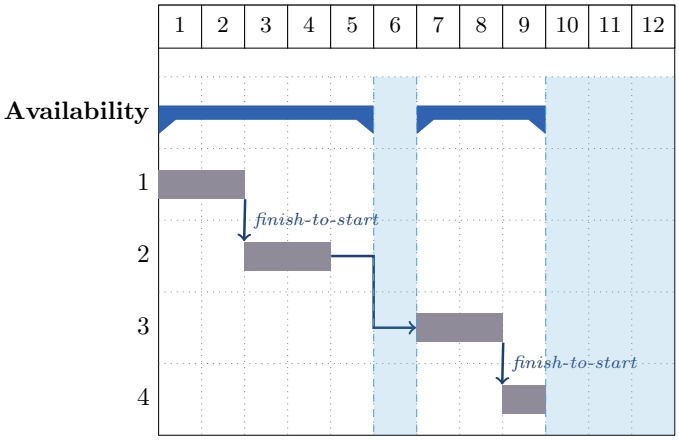

Figure 2. Non-interrupting case

Changes introduced in the second, interrupting case:

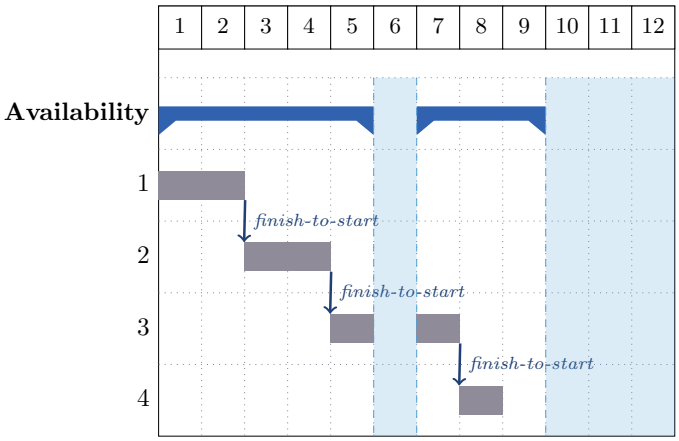

Figure 3. Interrupting case

When we take setup times into account, the time it takes to set up each machine is read from the setup table and added to the processing time of each operation. In our case, the first operation does not require any setup, so the setup time is 0. If we take the same case as before (case 1) and we define the setup table as:

Table 1. Setup table

Starting job	Finished job	time
1	1	0
1	2	1
1	3	2
1	4	1
2	1	3
2	2	0
2	3	2
2	4	2
3	1	1
3	2	2
3	3	0
3	4	2
4	1	1
4	2	1
4	3	2
4	4	0

Considering setup times, the Gantt of the base case (case 1) changes to:

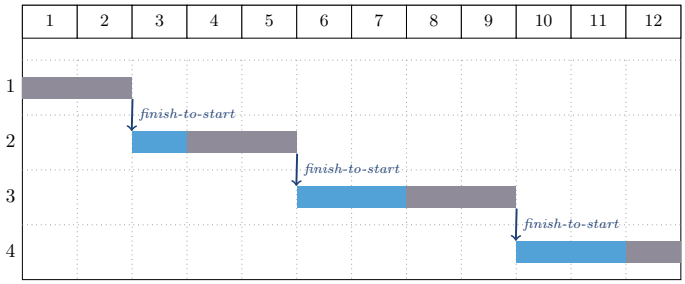

Figure 4. Setup times

The setup time from the first job to the second is 1 unit of time. Therefore the second job's processing time is $p_2^* = p_2 + 1 = 3$. If we continue throughout every job, we get the following set of processing times:

Table 2. Processing times with added setup times

i	p_i	setup	p_i^*
1	2	0	2
2	2	1	3
3	2	2	4
4	1	2	3

We also take into account the movement of the semi-finished product between the individual resources and machines. In this case, we add the delivery time to the start time of the work, which depends on the previous and current machine. In our case the transfer time before the first operation is 0.

Case 2. *Let the number of resources be* $NR = 4$*, the number of jobs* $NJ = 1$*, and the set of processing times* $\vec{p} = \{1, 3, 4, 1\}$.

In this case, the Gantt of the problem without taking transfer times into account:

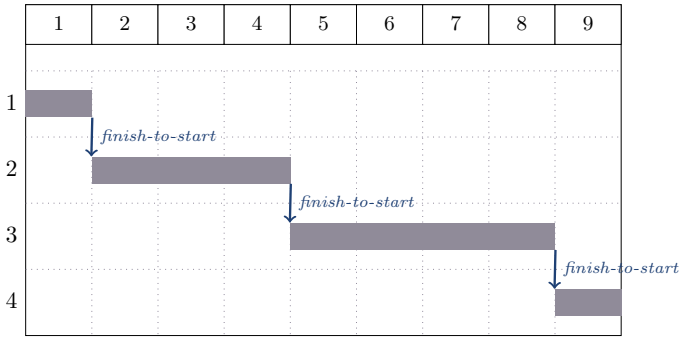

Figure 5. Gantt for case 2

Let transfer times be:

Table 3. Transfer times

Previous machine	Next machine	Time
1	1	0
1	2	1
1	3	2
1	4	1
2	1	3
2	2	0
2	3	2
2	4	2
3	1	1
3	2	2
3	3	0
3	4	2
4	1	1
4	2	1
4	3	2
4	4	0

If we take transfer times into account, the Gantt of the schedule is the following:

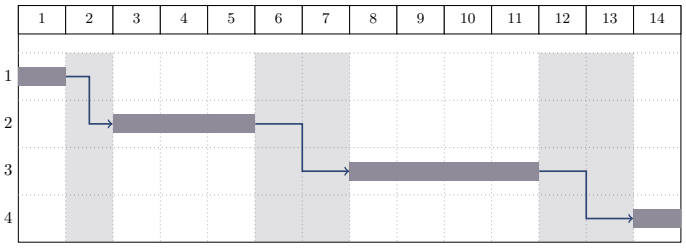

Figure 6. Transfer times

The table of the original and modified schedule:

Table 4. Schedule when considering transfer times

j	p_i	transfer	ST	ET	ST^*	ET^*
1	1	0	0	1	0	1
2	3	1	1	4	2	5
3	4	2	4	8	7	11
4	1	2	8	9	13	14

The algorithm for the simulation can be found in Algorithm 1. $variable.field$ denotes the field of a variable (struct or class). The list of parameters are:

Table 5. Parameters for the Simulation (algorithm 1)

Input parameters	
NJ	number of jobs
res	resource data
$res_r.cal$	time intervals
$res_r.trans_{r-1}$	transportation time of jobs from machine $r-1$ to machine r
$res_r.set_{k,l}$	setup time on machine r changing from job k to job l
NR	number of resources (machines)
s	schedule (the sequence of jobs to be executed)
t_0	start time of the execution
$mode$	mode of the job cutting (splitting)
Input/Output parameters	
job	calculated time data of the jobs
$job_j.start$	starting time of job j on machine r
$job_j.end$	finishing time of job j on machine r
$job_j.proc$	processing time of job j on machine r
Local Variables	
j	sequence index for jobs in schedule
r	index for resources (machines)

Algorithm 1 Simulation

```
for j ← 1, NJ do
    for r ← 1, NR do
        if j = 1 then                                          ▷ First job
            if r = 1 then                                      ▷ First machine
                jobs_{s_j}.start_r ← t_0
            else
                jobs_{s_j}.start_r ←
                    jobs_{s_j}.end_{r−1} +
                    res_r.trans_{r−1}
            end if
            jobs_{s_j}.end_r ←
                jobs_{s_j}.start_r +
                jobs_{s_j}.proc_r +
                res_r.set_{1,s_j}
        else
            if r = 1 then                                      ▷ First machine
                jobs_{s_j}.start_r ← jobs_{s_{j−1}}.start_r
            else
                jobs_{s_j}.start_r ← MAX(
                    jobs_{s_j}.end_{r−1} + res_r.trans_{r−1},
                    jobs_{s_{j−1}}.end_r)
            end if
            jobs_{s_j}.end_r ←
                jobs_{s_j}.start_r +
                jobs_{s_j}.proc_r +
                res_r.set_{s_{j−1},s_j}
        end if
        LoadToCalMode(jobs_{s_j}.start, jobs_{s_j}.end,        ▷ Fit job to time slot
            res, r, mode)
    end for
end for
```

Algorithm 2 contains the procedure where the jobs are fitted to an available time slot without interruption. Algorithm 3 contains the procedure where splitting jobs across multiple available time slots is permitted. If no available time slot is found, both algorithms append the job at the end and return with -1. Algorithm 4 contains the function that calls the previous two procedures according to the *mode*. If the *mode* is set to `true`, splitting jobs across multiple time slots is permitted. The list of parameters for all three algorithms (algorithm 2, 3, 4) are:

Table 6. Parameters for fitting jobs to time slots (algorithm 2, 3, 4)

Input parameters	
res	resource data
r	resource index
$res_r.cal_c$	availability time interval c
$res_r.cal_c.st$	start time of availability time interval c
$res_r.cal_c.et$	end time of availability time interval c
$res_r.ncal$	number of availability time intervals
$mode$	mode of the job cutting (splitting)
Input/Output parameters	
st	start time of the job
et	end time of the job
Local Variables	
fps	start of the first part
$found$	index of the time slot where the job is fitted
c	counter for time slots
$size$	size of job

Algorithm 2 Fitting a job to working time slots without cutting

```
function LoadToCalNoCut(st, et, res, r)
    found ← −1
    c ← 0
    size ← et − st
    while c < res_r.ncal do
        if st < res_r.cal_c.et then
            st ← max(st, res_r.cal_c.st)
            et ← st + size
            if et ≤ res_r.cal_c.et then                ▷ The job fits
                found ← c
                return found
            else                                       ▷ The job doesn't fit
                c ← c + 1
                if c ≥ res_r.ncal then                 ▷ No more slots available
                    st ← res_r.cal_{c−1}.et            ▷ Append to last slot
                    et ← st + size
                    return found
                end if
                continue to next iteration
            end if
        end if
        c ← c + 1
    end while
end function
```

Algorithm 3 Fitting a job to working time slots with cutting

```
function LoadToCalCut(st, et, res, r)
    fps ← −1
    found ← −1
    c ← 0
    size ← et − st
    while c < res_r.ncal do
        if st < res_r.cal_c.et then
            st ← max(st, res_r.cal_c.st)
            et ← st + size
            if fps = −1 then fps ← st
            end if
            if et ≤ res_r.cal_c.et then                    ▷ The job fits
                found ← c
                return found
            else                                           ▷ The job doesn't fit
                c ← c + 1
                if c ≥ res_r.ncal then                     ▷ No more slots available
                    et ← st + size
                    return found
                end if
                size ← size − res_r.cal_{c−1}.et − st      ▷ Remaining time
                continue to next iteration
            end if
        end if
        c ← c + 1
    end while
end function
```

Algorithm 4 Fitting a job to working time according to the mode

```
function LoadToCalCutMode(st, et, res, r, mode)
    if mode then                                           ▷ Cutting mode
        return LoadToCalCut(st, et, res, r)
    else
        return LoadToCalNoCut(st, et, res, r)
    end if
end function
```

Case 3. *Let the number of jobs be $NJ = 3$, the number of resources $NR = 2$, the matrix of processing times*

$$p_{i,j} = \begin{vmatrix} 10 & 12 \\ 20 & 5 \\ 22 & 15 \end{vmatrix},$$

and the schedule set by the search algorithm is $\vec{s} = \{1, 3, 2\}$.

The resulting schedule represented on a Gantt chart is the following:

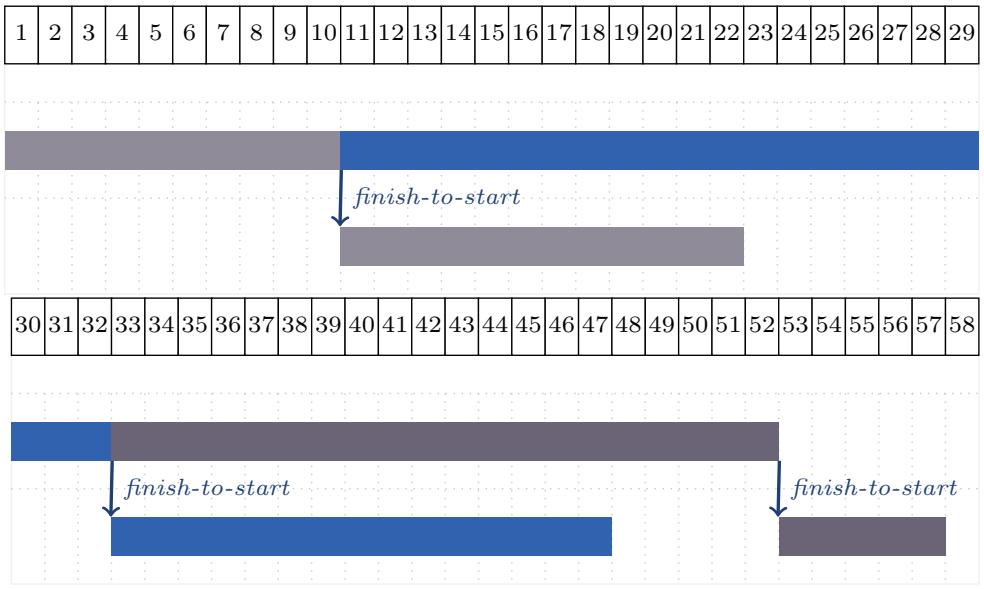

Figure 7. Schedule of case 3

Case 4. *Let the number of jobs, the number of resources, and the matrix of processing times along with the schedule be the same as in case 3. Let the transfer time between the two machines be $t = 3$. The table of setup times:*

Table 7. Setup times for case 4

machine	*from*	*to*	*value*
1	*1*	*3*	*2*
1	*3*	*2*	*5*
2	*1*	*3*	*4*
2	*3*	*2*	*3*

The operating time slots for the machines:

Table 8. Operating time slots for case 4

machine	*from*	*to*
1	*0*	*15*
1	*23*	*47*
1	*50*	*80*
2	*12*	*27*
2	*32*	*50*
2	*55*	*90*

If every operation must be processed in a single time slot, the resulting schedule is the following:

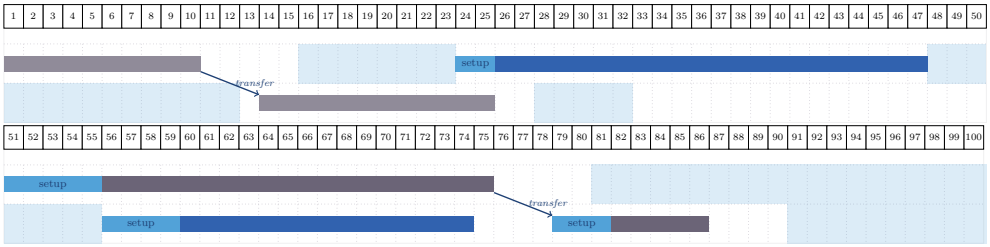

Figure 8. Schedule of case 4 with no cut

If we are allowed to execute an operation in multiple different time slots, the schedule changes to:

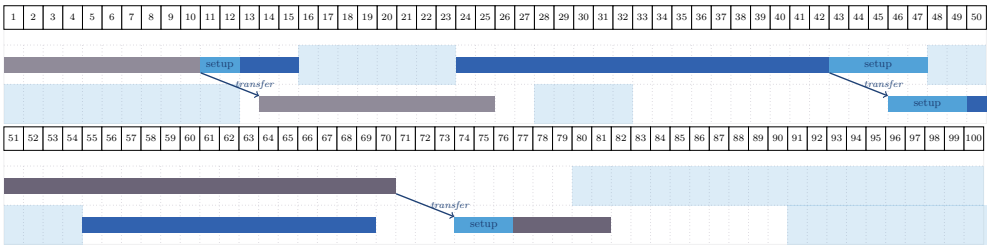

Figure 9. Schedule of case 4 with cut

In this section, we demonstrated the operation of the advanced simulation based on the extended data model. Experimental runs with our implemented

system have proven that the algorithms work correctly, efficiently, and meet the requirements of industrial applications.

This proposed extended simulation can support to solve the flow-shop scheduling problems, in which the resources (machines, workers and so on) are not available continuously, the transportation times of the workpieces must be considered, and the setup times of the resources cannot be ignored.

The main advantage of our proposed solution concept is that the search algorithms developed to handle traditional flow shop problems can still be used without change, because the proposed simulation completely hides the specific features of the real production systems. Our method also allows that the new meta-heuristic search algorithms will be used directly to solve the flow shop type scheduling problems by exploiting the potential of the proposed simulation.

5. Conclusion

Our research focused on scheduling work to be performed on time-limited resources. The classic one-way overtaking scheduling task type was extended to include material handling times and machine changeover times based on industry needs and to associate resource availability time intervals with resources and add them to the model as a constraint. This extension has significantly changed the characteristics of the underlying problem. The latest completion date was used as the optimization goal. In the basic model, the search for the best startup sequence was influenced only by the operation times of the jobs. In the case of the problem, we examined the changeover times depending on the work sequence, the material handling times between the machines, the interruptibility of the operations of the jobs, and the availability times of the machines that appeared.

We transform the extended problem to the classical one-way overtaking case using a simulation method, with the proposed simulation algorithms collectively encompassing the features of the extended problem. As a result, the extended problem has become manageable with any permutation search algorithm.

In addition to the fact that our solution method proved very efficient in this particular case, we also found that the proposed solution concept can be used for other scheduling tasks. The advantage of our solution is especially evident when resources are not available on time. Furthermore, the method can also be used for continuously available resources because we get back to the time axis of the original problem, which can be modeled so that each resource has a single availability time interval, according to which it is available from the initial time to infinity.

The results encouraged us to continue our research work, incorporate other specific industrial needs into the simulation, and solve different types of scheduling problems on a similar principle with a problem-transformation approach.

References

[1] Tibor, T.: Termelési rendszerek és folyamatok. *Miskolci Egyetemi Kiadó, Miskolc*.

[2] Kulcsár, G. and Erdélyi, F.: Multi-objective searching methods for solving scheduling and rescheduling problems. *microCAD 2007 International Scientific Conference*.

[3] Jackson, J. R.: Scheduling a production line to minimize maximum tardiness. *management science research project*.

[4] Jackson, J. R. et al.: An extension of johnson's results on job idt scheduling. *Naval Research Logistics Quarterly*, **3**(3), (1956), 201–203, URL https://doi.org/10.1002/nav.3800030307.

[5] Johnson, S. M.: Optimal two-and three-stage production schedules with setup times included. *Naval research logistics quarterly*, **1**(1), (1954), 61–68.

[6] Smith, W. E. et al.: Various optimizers for single-stage production. *Naval Research Logistics Quarterly*, **3**(1-2), (1956), 59–66, URL https://doi.org/10.1002/nav.3800030106.

[7] Cook, S. A.: The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, 1971, pp. 151–158, URL https://doi.org/10.1145/800157.805047.

[8] Garey, M. R., Johnson, D. S., and Sethi, R.: The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, **1**(2), (1976), 117–129, URL https://doi.org/10.1287/moor.1.2.117.

[9] Karp, R. M.: Reducibility among combinatorial problems. In *Complexity of computer computations*, pp. 85–103, Springer, 1972, URL https://doi.org/10.1007/978-1-4684-2001-2_9.

[10] Brucker, P.: Scheduling algorithms. *Journal-Operational Research Society*, **50**, (1999), 774–774.

[11] Lenstra, J. K. and Rinnooy Kan, A.: Complexity of scheduling under precedence constraints. *Operations Research*, **26**(1), (1978), 22–35, URL https://doi.org/10.1287/opre.26.1.22.

[12] Lenstra, J. K., Kan, A. R., and Brucker, P.: Complexity of machine scheduling problems. In *Annals of discrete mathematics*, vol. 1, pp. 343–362, Elsevier, 1977.

[13] Kreipl, S. and Pinedo, M.: Planning and scheduling in supply chains: an overview of issues in practice. *Production and Operations management*, **13**(1), (2004), 77–92, URL https://doi.org/10.1111/j.1937-5956.2004.tb00146.x.

[14] Graham, R. L., Lawler, E. L., Lenstra, J. K., and Kan, A. R.: Optimization and approximation in deterministic sequencing and scheduling: a survey. In *Annals of discrete mathematics*, vol. 5, pp. 287–326, Elsevier, 1979, URL https://doi.org/10.1016/s0167-5060(08)70356-x.

[15] Leung, J. Y.: *Handbook of scheduling: algorithms, models, and performance analysis*. CRC press, 2004.

[16] Lageweg, B., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A.: Computer aided complexity classification of deterministic scheduling problems. *Stichting Mathematisch Centrum. Mathematische Besliskunde*, **1**(BW 138/81).

[17] Lageweg, B., Lenstra, J. K., Lawler, E., and Kan, A. R.: Computer-aided complexity classification of combinational problems. *Communications of the ACM*, **25**(11), (1982), 817–822, URL https://doi.org/10.1145/358690.363066.

[18] Knust, S.: Complexity results for scheduling problems. http://www2.informatik.uni-osnabrueck.de/knust/class/ [2022-06-09], 2009.

[19] Kulcsar, G. and Erdélyi, F.: Modeling and solving of the extended flexible flow shop scheduling problem. *Production Systems and Information Engineering*, **3**, (2006), 121–139.