Computer Science and Information Systems 18(1):349–378 https://doi.org/10.2298/CSIS201009058S

# A JSSP Solution for Production Planning Optimization Combining Industrial Engineering and Evolutionary Algorithms

Sašo Sršen and Marjan Mernik

University of Maribor
Faculty of Electrical Engineering and Computer Science
Koroška cesta 46, 2000 Maribor, Slovenia
saso.srsen@student.um.si, marjan.mernik@um.si

**Abstract.** A Job Shop Scheduling Problem (JSSP), where *p* processes and *n* jobs should be processed on *m* machines so that the total completion time is minimal, is a well-known problem with many industrial applications. Many researchers focus on the JSSP classification and algorithms that address the different JSSP classes. In this research work, the production times, a very well-known theme covered in Industrial Engineering (IE), are integrated into an Evolutionary Algorithm (EA) to present a solution for real-world manufacturing JSSP problems solving. Since a drawback of classical IE is a manual determination of the technological times, an Internet of Things (IoT) architecture is proposed as a possible solution.

**Keywords:** JSSP, Genetic Algorithms, Evolutionary Algorithms, Industrial Engineering, Internet of Things

## 1. Introduction

The production planning/scheduling problem has been known for a very long time. Very often, we can find it under the term "Job Shop Scheduling Problem" (JSSP). The term first appeared in the 1950s, more specifically around 1954 [1]. JSSPs are generally known to belong to the group of the so-called non-deterministic problems, bound by polynomial-time hardness (NP or non-deterministic polynomial-time hardness). In practice this means that the time to calculate the optimal solution increases exponentially with the problem's size. JSSP is still considered to be one of the most challenging problems in terms of computation complexity today.

In the early research, several analytical techniques, like a branch and bound and heuristic approaches, have been proposed to solve the problem and deliver an optimal or near-optimal solution. With problem size growth (number of machines, jobs, processes), those approaches were not able to deliver the expected results anymore. Hence, more recently, the studies turned to other techniques, like simulation, Artificial Intelligence (AI) [38], and Evolutionary Algorithms (EAs) [26]. EAs are population-based search algorithms, which mimic concepts from biological evolution, such as survival of the fittest, crossover, and mutation. EAs are known to have a remarkable balance between exploration and exploitation [22] [51], which is needed to search an enormous space of all possible solutions efficiently [29], [21] [32]. Early examples of EAs are Genetic Algorithms (GAs), Evolutionary Strategies (ES), and Genetic Programming (GP) [26]. While recently, state-of-the-art metaheuristics are variants of Differential Evolution (DE) (e.g., jDE [20], SHADE [39]), and CMA-ES (IPOP-CMA-ES [7]). Algorithms that mimic problem-solving from nature are nowadays flourishing, and belong to a wider group of Swarm Intelligence (e.g., ABC [31] [34] [25]), or Computational Intelligence (e.g., TLBO [36], [23]). They are suitable for solving complex real-world problems [28], [37], [30], where the search space is simply too big to check all possible solutions.

Since 1985, when Davis [2] proposed the first GA based solution for the JSSP problem, a lot of research has been done to address the scheduling problem. The studies that followed developed different constraints, representations, and algorithms to classify, differentiate, and solve Shop Scheduling Problems (SSPs). In the recent review [9] fourteen classes of JSSP have been identified, based on their main characteristics: Job arrival process, inventory policy, duration time processing, and job attributes, as shown in Figure 1. We will explain the main characteristics of those 14 types in Section 2.2.

Although all of them are trying to solve the JSSP by arranging an optimal schedule with different goals (e.g., minimizing the makespan [4], completion time, the lateness of the due date, tardiness, throughput time), they still rely on a universal unit of measurement: Time. If we want to apply any JSSP solution in the real world, we need to clarify what kind of times should be used where and how to measure them successfully. Here, we rely on the relatively old knowledge from Industrial Engineering (IE) to define, measure and classify the production times precisely, as shown in Figure 1. That way, a "communication channel" between the real world and JSSP domain can be made. According to [8,10] three different approaches for production time determination exist: the actual data approach, the plan data approach, and the hybrid approach. In order to schedule production, any approach can be used, usually resulting in the implementation of a global standard methodology for time determination [8,10,11] (e.g. MTM, REFA, MOST, Work factor). If a manufacturing company product diversity is taken into account, the existence of a production time data database is a necessity and is usually covered by an "Enterprise Resource Planning" (ERP) System. Production management can access the data to schedule the production, but the traditional scheduling approaches (especially ad-hoc) very often don't achieve the desired results, possibly causing significant production efficiency decrease.

In this paper, we want to take advantage of the principles and methods found in IE [8,10,11] and combine them with an Evolutionary Algorithm (EA) JSSP approach in order to optimize and simplify the manufacturing scheduling process for production management by introducing a solution in the form of a tool.

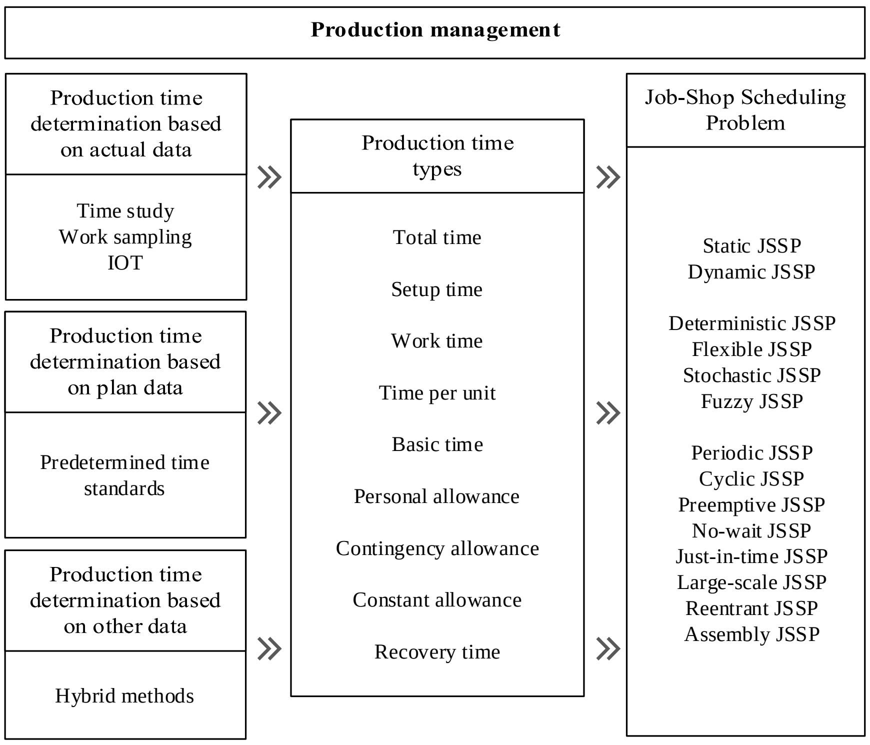


**Fig. 1.** Production time types and determination approaches, different JSSP classes

The main contributions of this research work are:

- Extending the JSSP with the time parameters found in manufacturing as defined by IE,
- Introducing an EA solution for a static flexible deterministic JSSP in the form of a tool,
- Proposing an IoT architecture to mitigate manual determination of technological times,
- Providing a use case with real-world data.

The rest of the paper is organized as follows. Section 2 covers the problem explanation and the time determination possibilities as per IE. In Section 3, we explain the proposed approach. Section 4 displays the solution use case example, problems, and results. Section 5 concludes with a summary of this work, adding some possible future research possibilities.

## 2. The proposed approach

### 2.1. JSSP description

How can we explain what the basic JSSP is? Informally, the problem could be described as follows: We have a set of jobs and a set of machines. Each job consists of a sequence of continuously performed processes for a specific time on a particular machine. Each machine can complete only one process at a time. The "schedule" represents the occupancy of machines with processes at specific time intervals. The key problem for this situation is creating a schedule where the finish time of the final process in the schedule is minimal. In general, the problem could be described formally as follows. Let the finite set M represent the set of all machines, and the finite set J represent the set of all jobs:

$$M = \{M_1, M_2, M_3, \dots M_m\} \tag{1}$$

and

$$J = \{J_1, J_2, J_3, \dots J_n\}. \tag{2}$$

If each job needs to be processed on all machines, but only once on each machine, the set representing the job sequence per machine could be written as a matrix $x$ of size $m\ x\ n$. For example:

$$x = \begin{pmatrix} 4 & 2 & 3 & 1 \\ 2 & 1 & 4 & 3 \\ 1 & 2 & 3 & 4 \end{pmatrix}. \tag{3}$$

Each row in the matrix represents a job order for the machine $M_1, M_2, M_3, \dots M_m$. The above matrix can therefore be read as: Machine processes on the machine $M_1$ will be performed in the sequence: $J_4, J_2, J_3, J_1$ processes, on the machine $M_2$ will be performed in the sequence: $J_2, J_1, J_4, J_3$, and processes on the machine $M_3$ will be performed in the sequence: $J_1, J_2, J_3, J_4$ . We can quickly conclude that the matrix $x$ is only one element of a broader set (let's call it, for example, the set X) of all possible combinatorial variants, i.e., $x \in$ X.

If we want to search for an optimal schedule, we need a general estimation function $f$ that can calculate the exact "value" for each matrix $x \in$ X:

$$f: \mathrm{X} \rightarrow [0, +\infty] \tag{4}$$

or, if we look more precisely, for each element of the matrix:

$$f_{ij}: J \times M \rightarrow [0, +\infty] \ . \tag{5}$$

Throughout the paper, we will use the index $i$ for jobs and index $j$ for machines. Because JSSP algorithms are used to optimize the time required, a common output of the estimation function represents the total execution time (also called timespan or makespan). Other possible function outputs include, and are not limited to, flow time (total weighted completion times) and lateness or tardiness (with a due date) [9]. The $f_{ij}$ function, therefore, calculates the total execution time of the job $J_i$ on the machine $M_j$. The JSSP solution is, therefore, a matrix $x \in$ X, where the makespan $f(x)$ for completing all the tasks (or jobs) is minimal, or that there is no known $y \in$ X where $f(x) > f(y)$ . In other words, the solution of JSSP is to find a schedule where:

- simultaneous processing of multiple jobs on the same machine is not possible,
- the same job cannot be processed simultaneously on multiple machines,

- each operation for an individual job occupies each machine for a specific time T, and
- the makespan is minimal.

## 2.2. Related work, JSSP classes and types

Although we have limited ourselves to the JSSP (Figure 1), we should first explain that, in general, three basic Shop Scheduling Problems (SSP) types exist [3]:
- Flow Shop Scheduling Problem (FSSP),
- Job Shop Scheduling Problem (JSSP), and
- Open Shop Scheduling Problem (OSSP).

The problem that needs to be solved for all the above-mentioned Shop Scheduling Problems is the same; the only difference is their limitations. In the case of FSSP, each job has exactly the same number of machine processes, and the sequence of machine processes for each job is predefined and the same, for example:

$$\begin{aligned} J_1 &: M_1 \rightarrow M_2 \rightarrow M_3 \rightarrow M_4 \\ J_2 &: M_1 \rightarrow M_2 \rightarrow M_3 \rightarrow M_4 \\ J_3 &: M_1 \rightarrow M_2 \rightarrow M_3 \rightarrow M_4. \end{aligned} \tag{6}$$

The possible sequences are usually limited by the technological process, and could be written for every job as: $J_i$: $M_1 \rightarrow M_2 \rightarrow \dots \rightarrow M_j$ , where $1 \le j \le m$ and $1 \le i \le n$. As already explained, the solution to the problem lies in finding a machine sequence where the estimation function (makespan) for completing all jobs is minimal.

In contrast to FSSP, the JSSP machine sequences are also limited by technology and known in advance, but they can vary, for instance:

$$\begin{aligned} J_1 &: M_1 \rightarrow M_2 \rightarrow M_3 \rightarrow M_4 \\ J_2 &: M_2 \rightarrow M_1 \rightarrow M_4 \rightarrow M_3 \\ J_3 &: M_1 \rightarrow M_2 \rightarrow M_4 \rightarrow M_3. \end{aligned} \tag{7}$$

Thus, for each job, we can write the technological machine sequence for a specific job $J_i$ as $M_{i1} \rightarrow M_{i2} \rightarrow M_{i3} \dots \rightarrow M_{i4}$, where $1 \le j \le m$ and $1 \le i \le n$. It should be emphasized that, in the case of simple JSSPs, we assume that the number of processes for each job is equal to $m$ or the number of machines (i.e., each job "travels" through all machines exactly once). In real life, however, often there is a situation where this number is less than $m$, meaning that each job does not need to be processed by all machines. Another case is where the number of processes exceeds $m$, resulting in repeating the machine process on an operation multiple times.

In the case of OSSP, the sequences of machine processes aren't predefined. It is often assumed that the number of machine processes for a job is equal to $m$, meaning that all machine processes must be completed for each job. We have to emphasize that OSSP occurrence is extremely rare in the real world.

If we look at JSSP, we can see many different types in Figure 1. We can classify them further by different criteria: job arrival criteria, time parameter criteria, and other criteria.

Using job arrival criteria two types of JSSP can be defined:
- Static JSSP, and
- Dynamic JSSP.

For static JSSPs, a finite number of jobs are ready for processing on a finite number of machines at the time zero [40]. An unexpected event occurrence is not possible.

Dynamic JSSPs are similar, except the job occurrence is random [3]. In both cases, the order of precedence of operations and processing times are predefined.

Using time parameter criteria, several types of JSSP can be defined:

- Deterministic JSSP,
- Flexible JSSP,
- Stochastic JSSP, and
- Fuzzy JSSP.

If the processing time for every operation of job $j$ on every machine $m$ is known in advance and the operation sequence order is predefined, we can classify it as a deterministic JSSP (also called a crisp JSSP) [42]. The flexible JSSP extends the deterministic JSSP by allowing a machine operation to be processed by one machine out of a set of machines, thereby adding the problem of assigning each operation to a specific machine (routing) [41]. Stochastic JSSPs introduce parameters dealing with probability conditions, for instance, machine breakdown or processing time [16]. Since real-world JSSP times often don't have deterministic value, fuzzy values (processing times, due date, ranking) have been incorporated into JSSP, hence the name fuzzy JSSP [43].

Using other criteria, further types of JSSP can be defined:

- Periodic JSSP,
- Cyclic JSSP,
- Preemptive JSSP,
- No-wait JSSP,
- Just-in-time JSSP,
- Large-scale JSSP,
- Reentrant JSSP, and
- Assembly JSSP.

The periodic JSSP is an iterative version of the JSSP where a batch of size $n$ of each job is processed iteratively [44]. The cyclic JSSP deals with a set of process operations that cycle an indefinite number of times by minimizing the period length [45]. In case the algorithm allows the interruption of an operation during processing on a specific machine and to continue at a later time, we're talking about pre-emptive JSSP [46]. The no-wait JSSP introduces the no-wait constraint between two sequential operations by delaying the job starting time at the first machine operation [47]. The just-in-time JSSP is solving the earliness-tardiness problem of jobs by penalizing both options [48]. The large-scale JSSP approach can be used when huge numbers of machines and jobs are required [49]. The Reentrant JSSP extends a deterministic JSSP, where a job operation may be repeated multiple times [50]. The assembly JSSP extends the JSSP by appending an assembly stage and introducing lot streaming (LS) thereby splitting the job into smaller batches and taking away job independence.

Many other subtypes exist, many of them extending the basic types with different constraints and Objective Functions, for instance, machine blocking constraints [17]. Recent research even covers the so-called "low-carbon" JSSPs by pursuing the goal to minimize the sum of completion time cost and energy consumption cost [18].

## 2.3. Time in Industrial Engineering (IE)

Since we'll be using production time as a parameter for the JSSP solution, we must take a look at how IE is determining and structuring production time. According to REFA [8], similar to Seifermann [10], when we need to determine a time for specific work or work part on an operational level, different approaches exist, as shown in Figure 2:

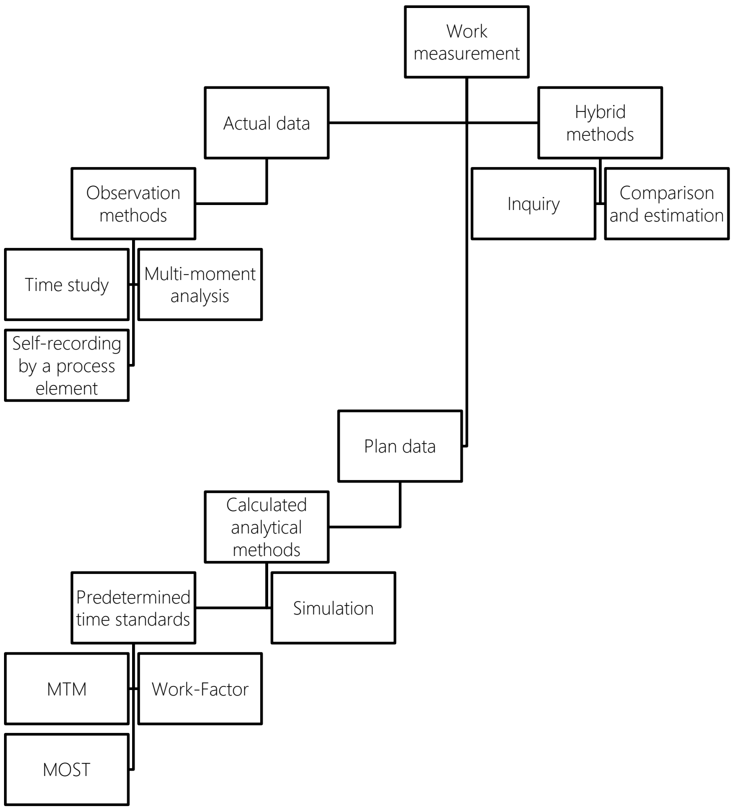


**Fig. 2.** Overview of different IE methods for time determination

The actual data approach requires the presence of an analyst in the workplace for work observation and measurement. In contrast, the plan data approach just requires a detailed work process analysis for work time determination. The hybrid approach combines techniques from the mentioned ones.

We should emphasize that the times that need to be determined for JSSP use are usually called "target times" ("Sollzeit") [8] or norm times. They often represent the foundation for different manufacturing divisions, like production planning and management, costing, controlling, and remuneration. Some of the mentioned divisions require another type of time called "actual time" ("Istzeit") [8] for their work, that represents the actual spent amount of time that has been used to complete a specific job.

According to REFA [8][14], the following applies:

$$T = t_r + t_a \,. \tag{8}$$

The total target time $T$ is divided into setup time ($t_r$, "Rüstzeit") and work time ($t_a$, "Arbeitszeit"). The setup (also called changeover) times are quantity independent, and can be defined as:

- Fixed, for a specific job/machine, and
- Variable or sequence-dependent, for a specific job/machine.

In serial production, work time $t_a$ can be written as:

$$T = t_r + \mathrm{m} * t_e \ . \tag{9}$$

Whereas quantity m ("Menge") stands for the total quantity of products required for a specific operation and/or job, depending on the level being used. The variable called $t_e$ ("Einzelzeit") stands for time per unit, and defines the target time required to manufacture/process one unit of the product (liter, kg, meter, piece). The setup time is not quantity dependent, as displayed in Figure 3:

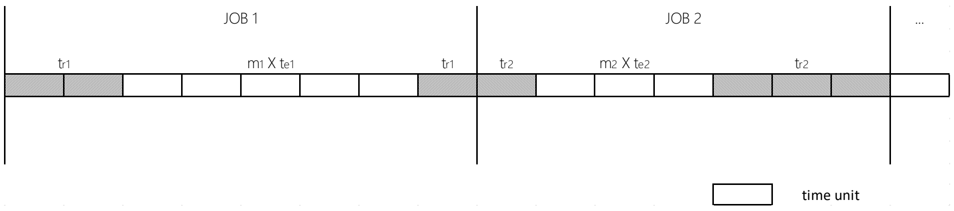


**Fig. 3.** Production timeline example

Each time we swap the product (or change the job) on a machine, the setup time usually occurs at the beginning and/or at the end of the process. The processes that require setup times can start before the previous process step in the job has finished. As shown in the example in Figure 4, the setup time $t_r$ is one unit long. The total time T for job 3 on machine 3 is actually two units, but since machine 3 is IDLE before job 3 on machine 4 is finished, the setup on machine 3 can start.

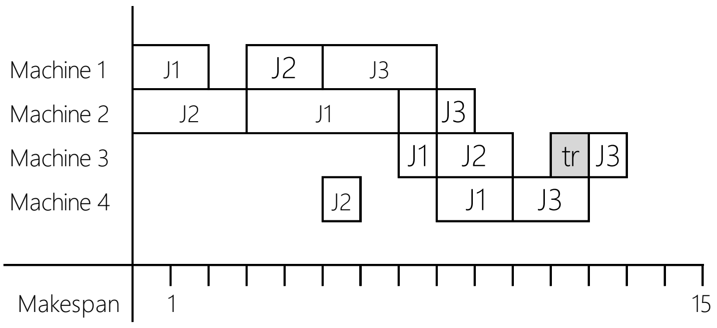


**Fig. 4.** Setup time

Time per unit $t_e$ gets divided further into three parts, namely:

$$T = t_r + \mathrm{m} * (t_g + t_v + t_{er}) \ . \tag{10}$$

The first variable in the equation, basic time $t_g$ ("Grundzeit") represents the time of a bare machine or manual work or a combination of them. The second variable, $t_v$ ("Verteilzeit") or allowances represent a percentage of smaller interruptions/disturbances in the work process that occur randomly that gets added to the basic time [15]. Those disturbances can be divided further into contingency allowances $t_{vsv}$, personal allowances $t_{vp}$ and special/constant allowances $t_{vsk}$. Contingency allowances cover short stochastic process delays like machine breakdown or raw material shortage, for example. Personal allowances cover personal needs like toilets or drinking fluids. Special or constant allowances are usually given per work period, and cover the time needed for work that isn't bound directly to any work order, like cleaning at the end of the shift or tooling at the beginning of the work. Allowances can be defined per product/process/machine, but, usually, they are defined on a higher level, for example, a group of workplaces or a sector, or even for the entire production plant. The third variable, $t_{er}$ ("Erholungszeit") or relaxation allowances, occur only in harsh work conditions (heat, radioactive environment, etc.) and, similar to the allowances, raise the basic time by a certain percent to compensate for the delegated breaks.

A specific approach for determination can be used for every time component mentioned. To determine basic times, usually an observation methodology is used, like time study, or a predetermined time system (MTM, Work factor or MOST).

**Time study**

A time study approach requires the use of a chronometer, and is completed by an analyst, who is present while the work is being executed. The process can be divided into four phases:

- Work (place) analysis and phases definition,
- On-site work measurement (using a chronometer), performance rating,
- Time study analysis, and
- Reporting and data updating.

The measurement usually requires multiple cycles (or samples) to get reliable data.

**Predetermined time systems**

In contrast, predetermined time systems instead use motion study as the foundation [11]. There is no need for on-site presence if you are able to break down the work into single standardized motion elements, the building blocks of a predetermined time system.

Only basic time is determined by summing times for all identified motion elements using matrices on cards, as defined by the Standard (MTM, MOST or Work factor). Table 1 displays the standard MTM-1 motions. By combining basic motion elements (MTM-1, for example), higher-level motion elements can be defined (MTM-UAS or MTM-MEK, for example).

**Table 1.** MTM-1 standard motion elements [13]

| **Hand/arm motion elements** | **Eye motion elements** | **Body, leg and foot motions** |
|---|---|---|
| Reach (R) | Eye travel (ET) | Foot motion (FM) |
| Grasp (G) | Eye focus (EF) | Leg motion (LM) |
| Release (RL) | | Sidestep (SS) |
| Move (M) | | Turn body (TB) |
| Position (P) | | Walk (W) |
| Apply pressure (AP) | | Bend (B), Arise from bend (AB) |
| Disengage (D) | | Stoop (S), Arise from stoop (AS) |
| Turn (T) | | Kneel on one knee (KOK), Arise from kneeling on one knee (AKOK) |
| | | Kneel on both knees (KBK), Arise from kneeling on both knees (AKBK) |
| | | Sit (SIT), Stand (STD) |

**Allowances**
Work sampling (Multimoment analysis) or long-term time study (LTTS) can be used for determining allowances. Since we're determining stochastic events` frequency and duration, a much larger sample size is required compared to the classical time study.

If we want to extend the technology matrix (e.g. from Tables 3 or 4) with the time definitions, as explained earlier in this Section, we should define the time for each cell in the technology matrix as:

$$T_{ik} = t_{r_{ik}} + \mathrm{m} * (t_{g_{ik}} + t_{v_{ik}} + t_{er_{ik}}) \ . \qquad (11)$$

where $1 \leq i \leq m$ and $1 \leq k \leq p$.

## 3. Solution implementation

According to Section 2.2, the presented approach could be classified as a static flexible JSSP solution, meaning that we have all the jobs ready at time zero. All the processing times are predefined and for each operation, and the solution can choose a machine out of a set of machines. The number of processes $p$ is not necessarily equal to $m$ machines, with the meaning:

- If $p < m$: The job doesn't have to visit every machine, and
- If $\mathrm{p} > m$: The job re-enters a specific machine as defined by the technology matrix.

**Chromosome representation**
When implementing an EA solution, the first problem is the representation of an individual in the form of a chromosome [26]. We chose to use the unpartitioned permutation with m-repetitions representation [5][12]. Every job $J$ (out of $n$ jobs) consists of $p$ processes that we have to schedule on $m$ machines, thus giving us a two-row chromosome with the size of $n * p$ elements $N$, as shown in Figure 5:

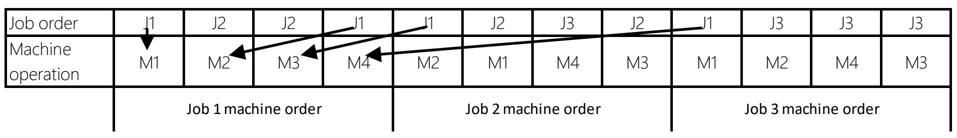


**Fig. 5.** Chromosome representation

In this formulation, each job $J_j$ appears exactly $p$ times (number of defined processes) in the first row of the chromosome, while the second row specifies the process sequence for a specific job, consisting of $p$ elements (machines). When scanning the job order from left to right, each job iteration increases the machine operation index for that job by 1 (Figure 5). Permutation, in this case, only means a change in the order in which the job processes will be performed, as shown in Figure 6:

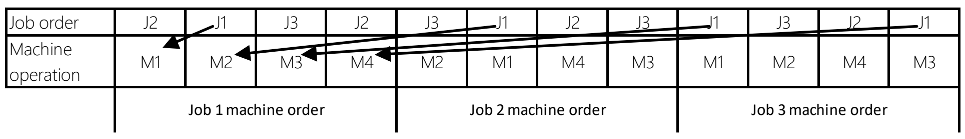


**Fig. 6.** Schedule permutation example

**Table 2.** Predefined machine order with times` example

| **Operation** / **Job** | **1.** | **2.** | **3.** | **4.** |
|---|---|---|---|---|
| **J1** | M1, 2 | M2, 4 | M3, 1 | M4, 2 |
| **J2** | M2, 3 | M1, 2 | M4, 1 | M3, 2 |
| **J3** | M1, 3 | M2, 1 | M4, 2 | M3, 1 |

Usually, gene J1 with occurrence index 1 in the chromosome means that the processing (if possible) must begin on machine 1, the next occurrence of this index implies that the processing must begin on machine 2, the next occurrence on machine 3, etc. as defined by the Job 1 machine order.

A job/machine is usually defined as a fixed Table value (Table 2), meaning that the processing for job 1 on machine 1 lasts two units, on machine 2 four units, etc. Since we classified the solution as flexible, rather than using Table 2, we defined another Table, named the technology matrix, for each job $J$ consisting of $p$ processes on $m$ machines. Table 3 shows an example of a technology matrix filled with times for job J1 from Table 2. The Table itself introduces the flexibility by letting the solution choose between different machines for the same process, in case we have multiple machines available, as shown in Table 4. Process 1 for Job 1 can be completed on machine 1 lasting two units, or on machine 3 lasting three units, and process 2 can be completed on machine 2 or 3 (4 and 5 time units). In table 3, we also demonstrated the option where the number of processes exceeds the number of machines $(p > m)$. If $p > m$ for any job, the chromosome size raises to $n * \max(p)$. In case the number of processes for a specific job is lower than the number of machines $(p < m)$, the unused processes simply get a value of 0 for any machine.

**Table 3.** Technology matrix for job J1 with fixed machines and times as per the example

| | M1 | M2 | M3 | M4 |
|---|---|---|---|---|
| P1 | 2 | X | X | X |
| P2 | X | 4 | X | X |
| P3 | X | X | 1 | X |
| P4 | X | X | X | 2 |

**Table 4:** Technology matrix for job J1 with machine options and $p > m$:

| | M1 | M2 | M3 | M4 |
|---|---|---|---|---|
| P1 | 2 | X | 3 | X |
| P2 | X | 4 | 5 | X |
| P3 | X | X | 1 | X |
| P4 | X | X | X | 2 |
| P5 | X | 3 | X | X |

Because the 2nd row of the chromosome representation from Figure 5 still can't change (static JSSP), we can calculate the search space for the chromosome upper part for the JSSP representation as:

$$\frac{(p\ x\ n)!}{(p!)^n}. \qquad (12)$$

Formula (12) is based on permutations with repetition and because the number of processes is the same for all jobs, the denominator has the exponent $n$.

If we use the formula in our example from Table 1, the search space is limited by 34,650 different chromosomes. If we extend the static JSSP to flexible JSSP, where we define a matrix with different machine options (routing) for each job process, we can extend formula (12) to:

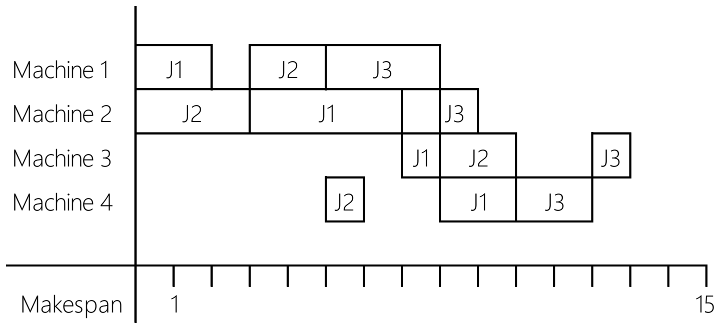
$$\frac{(p\ x\ n)!}{(p!)^n}\ x \prod_{i=1}^{n} \prod_{k=1}^{p} count(possible\ machines)_{(ik)} \qquad (13)$$

meaning that the number of feasible chromosomes will grow because now even the 2nd row of the chromosome (machine sequence) can change accordingly. By using formula (13), our example search space (let's presume the technology matrix from Table 4 is the same for all 3 jobs) grows to 2,217,600 different chromosomes. If we presume that all technology matrices are full (worst-case scenario, there are no infeasible solutions, every process can be completed on every machine), we can define the search space as:

$$\frac{(p\ x\ n)!}{(p!)^n}\ x\ m^{p\ x\ n}. \qquad (14)$$

This gives us a search space of 581,330,534,400 different chromosomes for our example.

The chromosome phenotype [26] can be represented by using a Gantt chart (Figure 7):



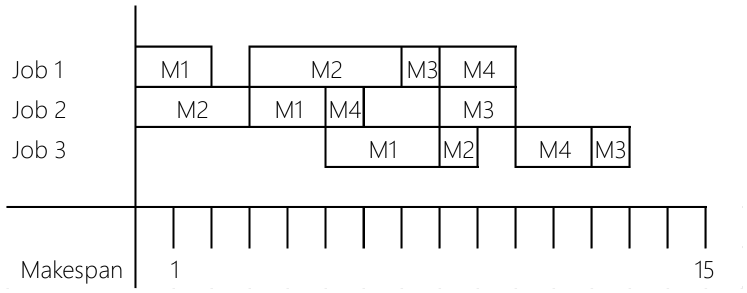


**Fig. 7.** Phenotype representations of Table 2 using a Gantt diagram *(machine/job)* and *(job/machine)* representation

Or, if we want to use the matrix representation (3), we could write:

$$x = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 1 & 3 \\ 1 & 2 & 3 \\ 2 & 1 & 3 \end{pmatrix}. \tag{15}$$

If a job needs fewer processes than $p$ to finish, the excessive processes get machine processing times with a duration of 0. If a process can not be completed on a specific machine as per the technology matrix (that chromosome represents an infeasible solution), that individual gets a "bad" fitness value which may, possibly, drive it out of the population.

## Initial population

In the current solution a random schedule population is generated with defined parameters: Number of jobs, number of machines, number of processes. A technology matrix with the size of $p\ x\ m$ is required for each job $J_i$.

## Selection operator(s)

Two selection operators have been implemented: Tournament selection and roulette wheel selection [26]. The tournament selection picks a group of a specific size randomly out of the population and orders the group according to the chromosomes' fitness values. The best individual is selected as one of the parents. The roulette wheel uses the individual fitness value (makespan) and normalizes it to 1 by dividing it by the total fitness of all individuals in the group, thus defining the probability of selection.

**Crossover operators**

Four different crossover operators have been implemented: Single point, two point, uniform, and ordered crossover operators [26]. For single point crossover, a random index $k$ is selected between 1/4 and 3/4 of the chromosome size $N$. Then the first $k$ genes are copied from parent 1 and the rest from parent 2, starting at gene $k + 1$, considering every job can only occur $p$ times in every chromosome, skipping the gene otherwise. When the copy index reaches $N$, we continue at the beginning of the 2nd parent chromosome (Figure 8).

The two-point genetic operator defines two indexes, the starting index $k$ and ending index $l$. They both get selected randomly, then the genes between index $k$ and $l$ get copied from parent 2. The next step is to copy the genes from parent 1, where the occurrence count of a specific job in the gene of the chromosome doesn't exceed the number of processes $p$. Finally, the empty spaces are filled with copies of genes from the second parent, but in an order in which they appear in the second parent after the ending index $l$ (Figure 9).

The uniform operator works very simply; it's flipping a coin for every gene, to decide whether the offspring will contain the gene from parent 1 or 2, starting at the beginning of the chromosome and counting the occurrence of each job in the gene. When the job occurrence count for the chosen job reaches $p$, we try to insert the gene from the other parent, and if the occurrence count of that gene hasn't reached $p$, we copy it to the offspring. Otherwise, we skip the gene. After the index reaches the chromosome size $N$, we copy the remaining missing genes from parent 2, starting with the first gene.

Parent 1

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Job order | J1 | J2 | J2 | J1 | J1 | J2 | J3 | J2 | J1 | J3 | J3 | J3 |
| Machine operation | M1 | M2 | M3 | M4 | M2 | M1 | M4 | M3 | M1 | M2 | M4 | M3 |
| | Job 1 machine order | | | | Job 2 machine order | | | | Job 3 machine order | | | |

Parent 2

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Job order | J2 | J1 | J3 | J2 | J3 | J1 | J2 | J3 | J1 | J3 | J2 | J1 |
| Machine operation | M1 | M2 | M3 | M4 | M2 | M1 | M4 | M3 | M1 | M2 | M4 | M3 |
| | Job 1 machine order | | | | Job 2 machine order | | | | Job 3 machine order | | | |

Offspring

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Job order | J1 | J2 | J2 | J1 | J3 | J1 | J2 | J3 | J1 | J3 | J2 | J3 |
| Machine operation | M1 | M2 | M3 | M4 | M2 | M1 | M4 | M3 | M1 | M2 | M4 | M3 |
| | Job 1 machine order | | | | Job 2 machine order | | | | Job 3 machine order | | | |

**Fig. 8.** Single point crossover operator (Random position k = 4, chromosome size N = 12)

Parent 1

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Job order | J1 | J2 | J2 | J1 | J1 | J2 | J3 | J2 | J1 | J3 | J3 | J3 |
| Machine operation | M1 | M2 | M3 | M4 | M2 | M1 | M4 | M3 | M1 | M2 | M4 | M3 |
| | Job 1 machine order | | | | Job 2 machine order | | | | Job 3 machine order | | | |

Parent 2

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Job order | J2 | J1 | J3 | J2 | J3 | J1 | J2 | J3 | J1 | J3 | J2 | J1 |
| Machine operation | M1 | M2 | M3 | M4 | M2 | M1 | M4 | M3 | M1 | M2 | M4 | M3 |
| | Job 1 machine order | | | | Job 2 machine order | | | | Job 3 machine order | | | |

Offspring

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Job order | J1 | J2 | J2 | J1 | J3 | J1 | J2 | J3 | J1 | J3 | J3 | J2 |
| Machine operation | M1 | M2 | M3 | M4 | M2 | M1 | M4 | M3 | M1 | M2 | M4 | M4 |
| | Job 1 machine order | | | | Job 2 machine order | | | | Job 3 machine order | | | |

**Fig. 9.** Two-point crossover operator (Random *position k = 4, l = 9, chromosome size N = 12*)

The ordered crossover operator (OX, Figure 10) is very similar to the two-point genetic operator, except that, after copying the genes between the starting index $k$ and ending index $l$, the rest is copied from the first parent, starting at the index $l$, and skipping values where the occurrence count of a specific gene exceeds the number of processes $p$.

Parent 1

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Job order | J1 | J2 | J2 | J1 | J1 | J2 | J3 | J2 | J1 | J3 | J3 | J3 |
| Machine operation | M1 | M2 | M3 | M4 | M2 | M1 | M4 | M3 | M1 | M2 | M4 | M3 |
| | Job 1 machine order | | | | Job 2 machine order | | | | Job 3 machine order | | | |

Parent 2

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Job order | J2 | J1 | J3 | J2 | J3 | J1 | J2 | J3 | J1 | J3 | J2 | J1 |
| Machine operation | M1 | M2 | M3 | M4 | M2 | M1 | M4 | M3 | M1 | M2 | M4 | M3 |
| | Job 1 machine order | | | | Job 2 machine order | | | | Job 3 machine order | | | |

Offspring

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Job order | J2 | J2 | J1 | J2 | J3 | J1 | J2 | J3 | J1 | J3 | J3 | J1 |
| Machine operation | M2 | M3 | M4 | M1 | M2 | M1 | M4 | M3 | M1 | M2 | M4 | M1 |
| | Job 1 machine order | | | | Job 2 machine order | | | | Job 3 machine order | | | |

**Fig. 10.** Ordered crossover operator (OX)

## Mutation operators

Two widespread mutation operators have been implemented: Exchange values and change values (Figure 11). The change value operator first selects a gene in the chromosome randomly, then changes the machine in the gene to a random machine $n$, where $n \in M$:

Original

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Job order | J1 | J2 | J2 | J1 | J1 | J2 | J3 | J2 | J1 | J3 | J3 | J3 |
| Machine operation | M1 | M2 | M3 | M4 | M2 | M1 | M4 | M3 | M1 | M2 | M4 | M3 |
| | Job 1 machine order | | | | Job 2 machine order | | | | Job 3 machine order | | | |

Mutated chromosome

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Job order | J1 | J2 | J2 | J1 | J1 | J2 | J3 | J2 | J1 | J3 | J3 | J3 |
| Machine operation | M1 | M2 | M1 | M4 | M2 | M1 | M4 | M3 | M1 | M2 | M4 | M3 |
| | Job 1 machine order | | | | Job 2 machine order | | | | Job 3 machine order | | | |

**Fig. 11.** Change value mutation operator

The change value operator cannot affect the job order because of the occurrence limitation. The exchange value operator selects two random genes and switches the job order and machine values (Figure 12).

Original

| Job order | J1 | J2 | J2 | J1 | J1 | J2 | J3 | J2 | J1 | J3 | J3 | J3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Machine operation | M1 | M2 | M3 | M4 | M2 | M1 | M4 | M3 | M1 | M2 | M4 | M3 |
| | Job 1 machine order | | | | Job 2 machine order | | | | Job 3 machine order | | | |

Mutated chromosome

| Job order | J1 | J2 | J1 | J1 | J1 | J2 | J3 | J2 | J2 | J3 | J3 | J3 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Machine operation | M1 | M2 | M1 | M4 | M2 | M1 | M4 | M3 | M3 | M2 | M4 | M3 |
| | Job 1 machine order | | | | Job 2 machine order | | | | Job 3 machine order | | | |

**Fig. 12.** Exchange values mutation operator

## Fitness function

While building the solution, we focused on a static deterministic production, meaning that the process times are known, and the jobs are ready at time zero. As already mentioned, we chose makespan as the fitness function for the implementation. Makespan can be written as $f_{max}$ and represents the time when the last operation process is completed [6]:

$$f_{max} = \max(f_1, f_2, \dots, f_n) \tag{16}$$

where:

$$f_j = \sum_{k=1}^{p} \left(W_{jk} + T_{mj(k)}\right). \tag{17}$$

Variable $f_j$ stands for job $j$ completion time, $W_{jk}$ stands for waiting (or IDLE) time of job $j$ at sequence $k$, and $T_{mj(k)}$ stands for the processing time for job $j$ on machine $m$ at sequence $k$.

## Algorithm

The pseudocode of the proposed GA is shown in Figure 13. At first, a random population of a predefined size is generated and makespans are calculated, the best one noted. Then, in a predefined loop of size MaxIterations, we use the chosen selection to select a group of chromosomes and perform a crossover with the two best parents in the group. Afterward, we apply the mutation operator to the two children. The two worst individuals in the group get substituted by the two children, the best makespan gets checked. We apply another mutation on a random chromosome in the population and recheck if the best makespan changed. If 1/3 of the population went through the loop and the best makespan hasn't changed, we inject a % of fresh random chromosomes into the population.

The algorithm can be improved further using Long Term Memory Assistance (LTMA) [24], where duplicate solutions are identified. As such, time-consuming fitness evaluation is spared.

All the algorithm parameters can be changed by the user directly in the tool, like population size, selection group size, mutation probability pm, terminal condition MaxIteration, selection type, crossover-type, mutation type, and % of random chromosome injection if the fitness function didn't evaluate any better solution for 1/3 of the MaxIteration. The crossover probability pc is set to 1.

```
createInitialPopulation(population size)
calculateMakeSpans()
while (I < MaxIteration)
{
     doSelection(group size)
     doCrossover(best parent1, best parent2)
     doMutation(best child1, best child2)
     replace(worst parent1, worst parent2) with (best
     child1, best child2)
     checkBestMakeSpanChanged(best child1, best child2)
     doMutation(random chromosome)
     checkBestMakeSpanChanged(mutated chromosome)
     if (bestMakeSpan has not changed after 1/3 of
     MaxIteration)
     {
        injectNewRandomChromosomes(a % of population size)
        reset MakeSpanChanged counter
     }
}
```

**Fig. 13.** Genetic algorithm pseudocode

## 3.1. IoT

A potential approach in determining production times could also be by implementing a solution based on the Internet of Things (IoT) technology [35][19]. The goal of using IoT is to minimize or completely eliminate the need for human intervention in actual and target time data gathering, e.g., using IR scanners or terminals.

A sample IoT architecture for such purpose can be seen in Figure 14. Because the use case in the next Section was completed in a shoe factory, we will explain the working principle for the latter. The workers are using trolleys to transport upper shoes from one machine to another. Currently, each trolley receives a unique job (work order) barcode, so the worker can scan this barcode and a barcode on the machine to signal the beginning or end of a work process. This way, the ERP system can track the job completion status at the process level. However, tracking time and status in the explained way requires a high amount of discipline among the workers, meaning that any delay or mistake (e.g., forgetting to register at the beginning, or at the end of the process) in the registration can potentially produce unexplainable errors in the data interpretation. IoT use minimizes the registration mistake possibility.

Figure 14 explains a different approach proposal. The trolley must "know" which job it is carrying, so the Production Manager must somehow provide the ERP system with that information before the trolley is launched (e.g. by using a barcode or Radio-Frequency Identification (RFID) technology). The trolley must be equipped with a small computer, System on Chip (SoC), ESP8266 used in the example, and different sensors. In our case, the trolley is also equipped with two load cells, an RFID scanner, an accelerometer, and a gyroscope. When the worker drives the trolley around the shop, the accelerometer and gyroscope sensors detect movement and send the movement time data across WiFi to the Message Queueing Telemetry Transport (MQTT) broker using the MQTT protocol. The time resolution and data amount require the use of a No-SQL

Database (e.g., MongoDB) to store the streamed data. When the worker stops in front of a specific machine to start working on a job, the trolley has to stand still in a specific place for a certain amount of time. RFID tags on the reserved trolley position should be used to bind the machine ID to the current job ID. Again, the time data should be sent to the broker as soon as the sensor recognizes the RFID tag, and also when it leaves the reserved position. While the worker is completing the job on the machine, the two load cells stream the trolley weight data to the MQTT broker. Because only streaming the time data to a database still wouldn't provide input to the JSSP solution, an intelligent service is required to match and identify production events described above and allocate the time data to a specific process and define the type of time.

Using IoT as a means of determining production times opens a new perspective, not only for the JSSP solutions:

- Automatic target times` actualization based on chronological time data: Basic times, setup times and allowances in the ERP and/or other services like a JSSP tool,
- Delegation of short-term delayable allowance events (e.g., filling containers with very fine material, non-urgent cutting tool change, personal needs, etc.),
- Scheduling progress monitoring & dynamic re-launch of the JSSP search in case of unforeseen schedule deviations (e.g. longer machine breakdowns, unexplainable long delays, etc.).

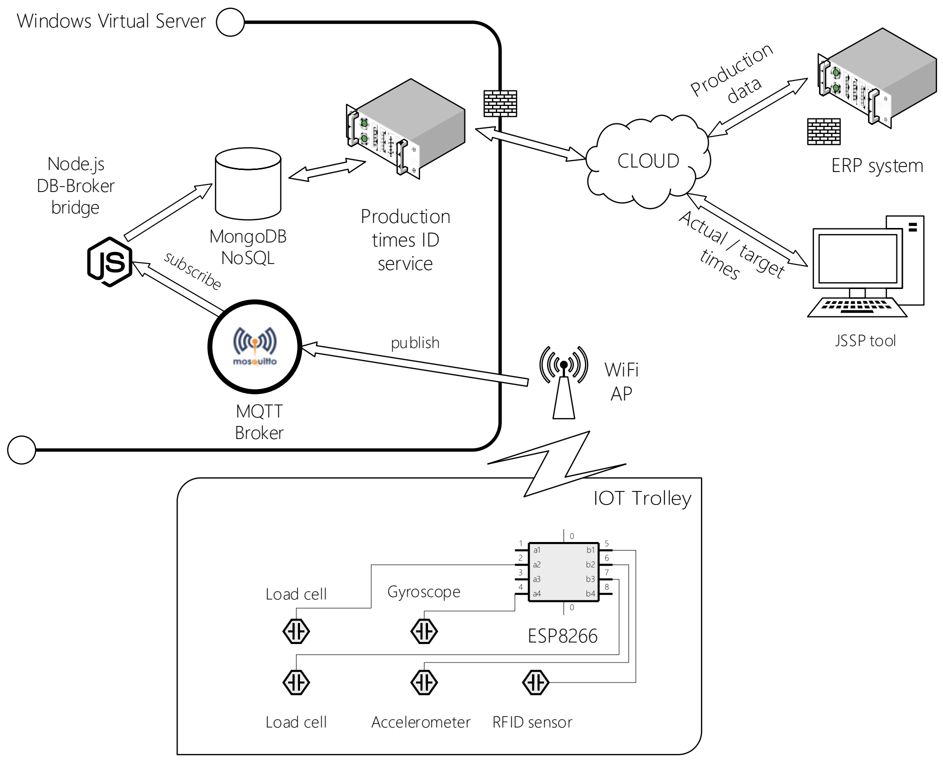


**Fig. 14.** An IoT production time determination architecture proposal

## 4. Use case

The implementation was made with a specific goal in mind, to optimize daily production scheduling/job launch for a shoe company. The shoe company has many production sectors, but for the use case, we focused on only a specific one, the upper shoe production sector. The problem that occurred is that the company was not able to define a daily job schedule with a predictable outcome. Additionally, the information about jobs and quantities for the upcoming day are usually defined at the end of the shift. Usually, the Production Managers were the ones delegating the work according to their past experience.

Papers and journals often use the term "machines" in JSSP, but, in practice, we can generalize the meaning of "resource." This small alteration is beneficial, since, in general, the term is a useful definition for manual workplaces as well as machines or machine types or groups; the principles described in Section 3.3 are valid for all. If we look at the parameters in Table 5 for a specific production request at the beginning of the week we got from the company:

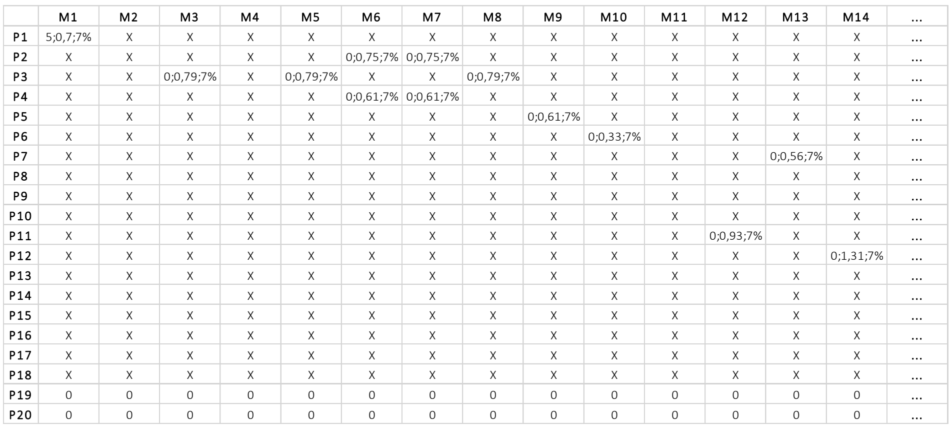
**Table 5.** JSSP / time parameters

No. of different resources $m$ … **38**
Max. no. of processes $p_{max}$ … **20**
No. of jobs/shift… **7**
Basic times $t_g$ … defined per resource / process / job

Setup times $t_r$ … fixed per resource, only M1 and M38 needing 5 minutes
Allowances $t_v$ … **7% (**fixed for all resources / jobs)
Relaxation allowances $t_{er}$ … 0% (normal working conditions)

| Job | m (pcs required) |
|---|---|
| 1 | 20 |
| 2 | 280 |
| 3 | 140 |
| 4 | 140 |
| 5 | 150 |
| 6 | 50 |
| 7 | 100 |
| sum | 880 |

Immediately, we can see the dimensions of the real-world problem. The total number of pieces (pairs) of shoes was 880, the task was split into seven jobs. Each job technology matrix contained 760 different matrix cells (possible process times) per job, meaning that each job contained a maximum of twenty processes that could be performed on 38 resources, as shown in Figure 15:



| | M1 | M2 | M3 | M4 | M5 | M6 | M7 | M8 | M9 | M10 | M11 | M12 | M13 | M14 | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| P1 | 5;0,7;7% | X | X | X | X | X | X | X | X | X | X | X | X | X | ... |
| P2 | X | X | X | X | X | 0;0,75;7% | 0;0,75;7% | X | X | X | X | X | X | X | ... |
| P3 | X | X | 0;0,79;7% | X | 0;0,79;7% | X | X | 0;0,79;7% | X | X | X | X | X | X | ... |
| P4 | X | X | X | X | X | 0;0,61;7% | 0;0,61;7% | X | X | X | X | X | X | X | ... |
| P5 | X | X | X | X | X | X | X | X | 0;0,61;7% | X | X | X | X | X | ... |
| P6 | X | X | X | X | X | X | X | X | X | 0;0,33;7% | X | X | X | X | ... |
| P7 | X | X | X | X | X | X | X | X | X | X | X | X | 0;0,56;7% | X | ... |
| P8 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | ... |
| P9 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | ... |
| P10 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | ... |
| P11 | X | X | X | X | X | X | X | X | X | X | X | 0;0,93;7% | X | X | ... |
| P12 | X | X | X | X | X | X | X | X | X | X | X | X | X | 0;1,31;7% | ... |
| P13 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | ... |
| P14 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | ... |
| P15 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | ... |
| P16 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | ... |
| P17 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | ... |
| P18 | X | X | X | X | X | X | X | X | X | X | X | X | X | X | ... |
| P19 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... |
| P20 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... |

**Fig. 15.** Job 1 technology matrix for the first 20 machines as defined by the product technology

For each row (or process), we calculated the time $m * t_e$ that is needed for a specific resource, in order to complete the job quantity m. Figure 15 displays the technology matrix containing the required time data: $t_r, t_g$ and $t_v$, where the times are delimited with a semi-column. If multiple cells are filled in the same row, the process can be completed with either of the resources, as explained in Section 2. For manual work, the times were usually the same, or very similar, for every possible resource, because the resource location is not very time relevant. The letter X in a cell means that it is impossible for the process to be completed on this resource. Since Job 1 in Figure 15 only needs 18 processes to finish, processes 19 and 20 get a value 0 for every machine. The company used LTTS for allowances` determination, and used a fixed value for all resources. Figures 16 and 17 display the actual tool window where all the parameters can be set, and the obtained solution is visualized [33].

To calculate the search space, we used the formula (12), so the upper chromosome part can be evaluated as:

$$\frac{(p\ x\ n)!}{(p!)^n} = \frac{(20\ x\ 7)!}{(20!)^7} \cong 2{,}67\ x 10^{112}$$

and the lower part to (without infeasible solutions):

$$\prod_{i=1}^{n} \prod_{k=1}^{p} \cong\ 7{,}54\ x\ 10^{30}$$

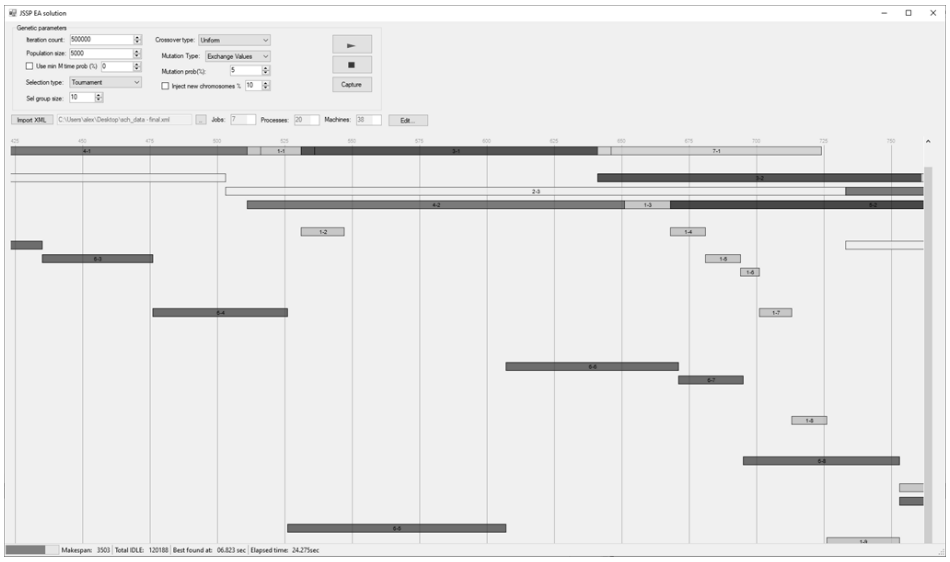
meaning a search space size without any infeasible solutions is roughly around $2{,}01\ x\ 10^{143}$ different chromosomes, or around $1{,}01\ x\ 10^{253}$ including the infeasible solutions.

The settings we used for GA were chosen experimentally in order to find the optimal solution as fast as possible:

**Table 6.** Used genetic parameters and their settings

| | |
|---|---|
| Iteration count | 500,000 |
| Population size | 5,000 |
| Crossover type | Uniform (pc = 0.5) |
| Selection type | Tournament |
| Tournament size | 10 |
| Mutation type | Exchange values |
| Mutation probability | 5% |

The algorithm has been run 20 times in sequence using the parameters from Table 6. 95% (19/20) of the time, the algorithm was able to find the optimal makespan (3,503 minutes) in around 30-35 seconds on the I7-7800X 6 core computer (12 logical processors) with 16GB of RAM.


JSSP EA solution

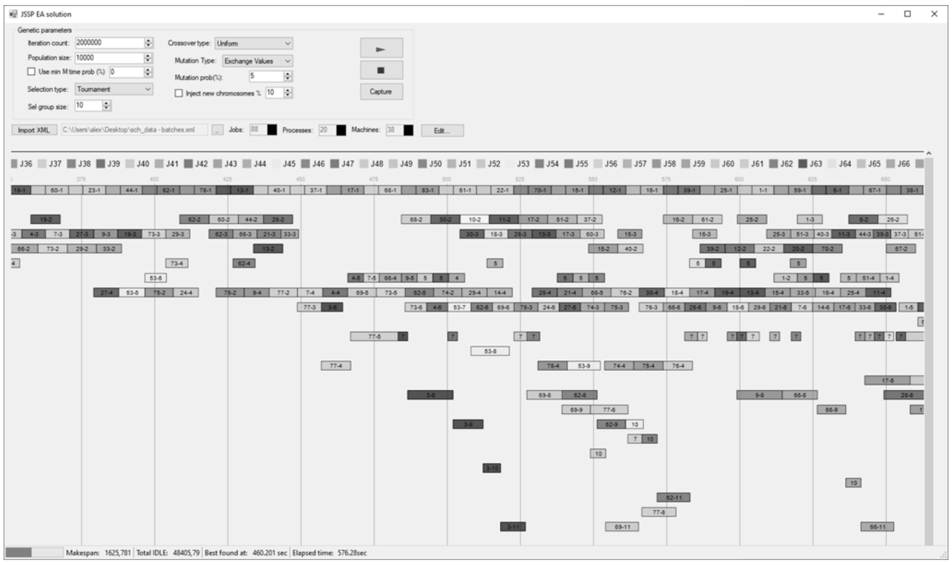
Genetic parameters

Iteration count: 500000
Population size: 5000
Use min M time prob (%) 0
Selection type: Tournament
Sel group size: 10
Crossover type: Uniform
Mutation Type: Exchange Values
Mutation prob(%): 5
Inject new chromosomes % 10

Import XML C:\Users\alex\Desktop\sch_data - final.xml Jobs: 7 Processes: 20 Machines: 38

Makespan: 3503 | Total IDLE: 120188 | Best found at: 06.823 sec | Elapsed time: 24.275sec

**Fig. 16.** App window example without lot streaming


JSSP EA solution

Genetic parameters

Iteration count: 2000000
Population size: 10000
Use min M time prob (%) 0
Selection type: Tournament
Sel group size: 10
Crossover type: Uniform
Mutation Type: Exchange Values
Mutation prob(%): 5
Inject new chromosomes % 10

Import XML C:\Users\alex\Desktop\sch_data - batches.xml Jobs: 88 Processes: 20 Machines: 38

Makespan: 1625,781 | Total IDLE: 48405,79 | Best found at: 460.201 sec | Elapsed time: 576.28sec

Capture
Edit...
J36 J37 J38 J39 J40 J41 J42 J43 J44 J45 J46 J47 J48 J49 J50 J51 J52 J53 J54 J55 J56 J57 J58 J59 J60 J61 J62 J63 J64 J65 J66

**Fig. 17.** App window example using lot streaming

Table 7 shows the data gathered across the 20 runs. Figure 18 shows a 3d histogram displaying the discrete IDLE time occurrence per machine.

**Table 7.** Results of running the algorithm 20 times

| | **Best found at [s]** | **Elapsed time [s]** | **Total IDLE time [min]** | **Makespan [min]** |
|---|---|---|---|---|
| **Run 1** | 7,367 | 30,707 | 120369 | 3503 |
| **Run 2** | 4,532 | 30,979 | 120384 | 3503 |
| **Run 3** | 6,311 | 30,694 | 120379 | 3503 |
| **Run 4** | 6,91 | 31,228 | 120305 | 3503 |
| **Run 5** | 4,425 | 30,874 | 120305 | 3503 |
| **Run 6** | 25,999 | 30,84 | 120379 | 3506 |
| **Run 7** | 5,987 | 35,819 | 120196 | 3503 |
| **Run 8** | 6,469 | 35,394 | 120308 | 3503 |
| **Run 9** | 7,946 | 35,564 | 120389 | 3503 |
| **Run 10** | 8,427 | 35,429 | 120379 | 3503 |
| **Run 11** | 9,1 | 34,821 | 120359 | 3503 |
| **Run 12** | 7,852 | 34,854 | 120374 | 3503 |
| **Run 13** | 4,72 | 34,581 | 120318 | 3503 |
| **Run 14** | 10,76 | 34,057 | 120342 | 3503 |
| **Run 15** | 8,272 | 34,643 | 120364 | 3503 |
| **Run 16** | 5,286 | 34,831 | 120285 | 3503 |
| **Run 17** | 10,051 | 34,49 | 120152 | 3503 |
| **Run 18** | 7,093 | 34,797 | 120374 | 3503 |
| **Run 19** | 4,017 | 34,763 | 120374 | 3503 |
| **Run 20** | 8,819 | 35,921 | 120384 | 3503 |

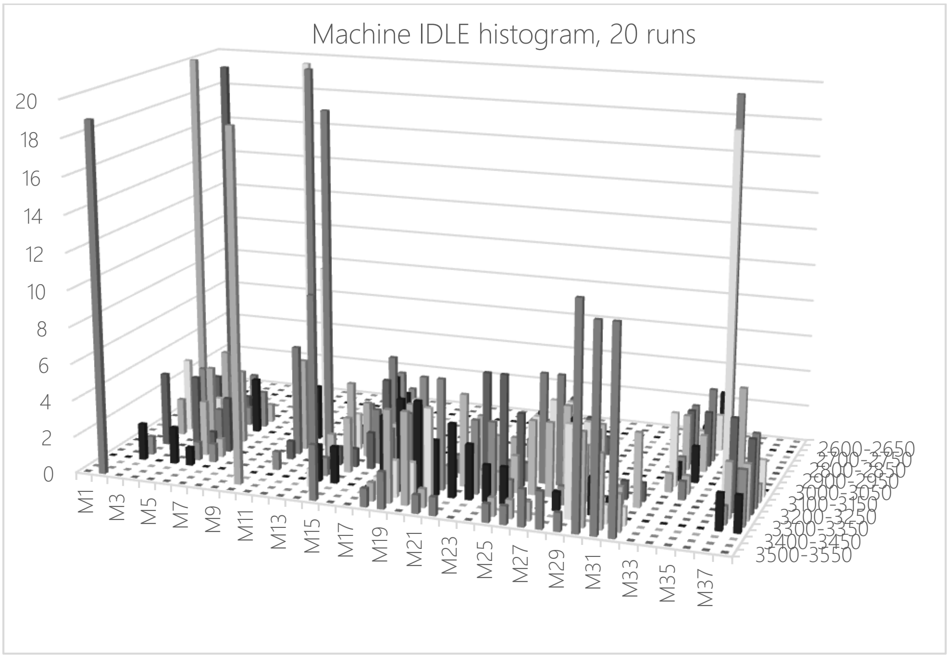


**Fig. 18.** IDLE time histogram per resource

The IDLE interval for all resources lay between [2583,3586] so the histogram IDLE intervals are defined between [2600,3550], with a step of 50. The graph shows that the machine IDLE times` occurrences are mainly concentrated in the upper bound of the IDLE interval, closer to the makespan (3000 < IDLE < makespan), except for maybe M33 and M34, meaning all the other resources are badly utilized. Ideally, the IDLE time occurrence count should be as close to 0 as possible.

Because the shoe company production was lot-organized, they split the entire job (order) quantity into smaller lots. The lot size was standardized and consisted of 10 pairs. This kind of approach is called lot streaming [4]. By splitting each job into sub-jobs of 10, the EA job parameter stretches from 7 to 88, remarkably increasing the search space (formula 12):

$$\frac{(p\ x\ n)!}{(p!)^n} = \frac{(20\ x\ 88)!}{(20!)^{88}} \cong 6{,}12\ x10^{4949}$$

$$\prod_{i=1}^{n} \prod_{k=1}^{p} count(possible\ machines)_{(ik)} \cong \ 3{,}47\ x\ 10^{357}$$

meaning we have around $2{,}12\ x\ 10^{5307}$ different chromosomes without infeasible solutions, or $2{,}32\ x\ 10^{6711}$ including all the infeasible solutions. Because of the increase the algorithm wasn't able to find the optimal solution with the EA parameters described in the use case. Even by doubling the initial population and quadrupling the iteration count, the algorithm success rate dropped significantly. The best-found makespan was 1,504, reducing the result found with the non-lot approach by 57%.

**Table 8.** Comparison of IDLE time/utilization between lot streaming and conventional scheduling

| | **Utilization** | | **IDLE time** | |
|---|---|---|---|---|
| | **8 jobs** | **88 jobs** | **8 jobs** | **88 jobs** |
| **M1** | 20,75% | 75,04% | 79,25% | 24,96% |
| **M2** | 0,00% | 0,00% | 100,00% | 100,00% |
| **M3** | 10,76% | 26,86% | 89,24% | 73,14% |
| **M4** | 18,01% | 41,94% | 81,99% | 58,06% |
| **M5** | 12,15% | 26,78% | 87,85% | 73,22% |
| **M6** | 9,89% | 22,72% | 90,11% | 77,28% |
| **M7** | 10,78% | 25,40% | 89,23% | 74,60% |
| **M8** | 22,36% | 51,78% | 77,64% | 48,22% |
| **M9** | 18,84% | 43,87% | 81,16% | 56,13% |
| **M10** | 0,20% | 0,47% | 99,80% | 99,53% |
| **M11** | 10,31% | 21,74% | 89,70% | 78,26% |
| **M12** | 10,90% | 25,92% | 89,10% | 74,08% |
| **M13** | 9,11% | 21,20% | 90,90% | 78,80% |
| **M14** | 11,37% | 27,39% | 88,63% | 72,61% |
| **M15** | 9,17% | 21,71% | 90,84% | 78,29% |
| **M16** | 12,18% | 29,73% | 87,82% | 70,27% |
| **M17** | 10,45% | 22,98% | 89,55% | 77,02% |
| **M18** | 5,40% | 8,28% | 94,60% | 91,72% |
| **M19** | 3,61% | 7,72% | 96,39% | 92,28% |
| **M20** | 3,49% | 8,79% | 96,51% | 91,21% |
| **M21** | 4,11% | 12,08% | 95,90% | 87,92% |
| **M22** | 7,71% | 20,39% | 92,29% | 79,61% |
| **M23** | 7,00% | 17,25% | 93,00% | 82,75% |
| **M24** | 8,60% | 17,88% | 91,40% | 82,12% |
| **M25** | 7,99% | 15,16% | 92,01% | 84,84% |
| **M26** | 9,73% | 24,28% | 90,27% | 75,72% |
| **M27** | 8,48% | 18,13% | 91,52% | 81,87% |
| **M28** | 9,61% | 21,09% | 90,40% | 78,91% |
| **M29** | 4,65% | 12,56% | 95,35% | 87,44% |
| **M30** | 0,61% | 4,63% | 99,40% | 95,37% |
| **M31** | 1,15% | 8,20% | 98,86% | 91,80% |
| **M32** | 1,94% | 4,34% | 98,06% | 95,66% |
| **M33** | 17,31% | 48,08% | 82,69% | 51,92% |
| **M34** | 19,03% | 36,54% | 80,98% | 63,46% |
| **M35** | 19,01% | 44,27% | 80,99% | 55,73% |
| **M36** | 12,25% | 28,52% | 87,76% | 71,48% |
| **M37** | 8,26% | 23,84% | 91,74% | 76,16% |
| **M38** | 7,75% | 22,76% | 92,26% | 77,24% |
| **AVG** | **9,60%** | **23,43%** | **90,40%** | **76,57%** |

Table 8 displays the utilization data comparison between using the lot streaming (best found) and not using the lot streaming (conventional scheduling). The average machine utilization rate rose from 9,60% to 23,43%.

We must emphasize the importance of setup times $t_r$ when using lot streaming. They play a significant role when searching for the optimal schedule. If they occur a maximum of $p$ times per job $j$ in conventional planning, the number of occurrences multiplies by $j\ /\ lot\ size$ when using lot streaming. This can cause the utilization rate to rise, but can also cause the makespan to increase.

Since the production was using lot streaming, we must compare the computed makespan of 1504 minutes with real-world data. The 88 carts were finished in around 3,5 shifts, resulting in 1680 minutes, meaning an efficiency increase of around 10,5% could be achieved by scheduling alone, confirming that using JSSP solutions in real-work can prove beneficial. Currently, daily scheduling is still left to the product engineers. Because of the size and complexity of the scheduling problem, no traditional tool can be used, so they launch the products daily simply by experience alone.

## 5. Conclusion

Implementing an EA solution for a specific Job Shop Scheduling Problem and combining it with Industrial Engineering knowledge proved to deliver good results. Many issues found in other types of JSSP were addressed, like shorter stochastic events and setup times, for example. Using the typical time components found in manufacturing brought the user domain and the science domain closer together. However, any schedule provided by a schedule optimization system like the one proposed can only be used as a guideline. The tool introduced in the article can be used at the end of the day for next-day scheduling by product engineers to search for the best option on how and when to launch the planned products but still must be extended by experience parameters that aren't included in the algorithm like specific worker skills or machine experience, for instance. Any longer stochastic event that can occur (and is not covered by allowances) would render the provided schedule unusable. The user can still rerun the optimization with the current situation parameters, but that could prove time-consuming. The proposed schedule could require rerouting the resources in a completely different way. Another drawback of classical Industrial Engineering is that all the time components necessary still demand a lot of work from a time analyst before all the technological times are determined. On the plus side of Industrial Engineering, we can find many manufacturing companies and even ERP systems using the time definitions mentioned. Another plus side is that any company target times should be as accurate as possible, otherwise causing inaccurate planning and cost calculations, providing an excellent fundament for JSSP. Future work should extend the solution with the proposed IoT architecture to measure and classify production times in real-time, and deliver the required data for the JSSP solution that would be run dynamically automatically.

## References

1. Johnson, S. M: Optimal Two and Three-Stage Production Schedules with Setup Times Included. Naval Research Logistic Quarterly, Vol. 1, No. 1. (1954)

2. Davis, L.: Job Shop Scheduling with Genetic Algorithms., Proc. of 1st Int. Conf. on Genetic Algorithms, Lawrence Erlbaus Associates, p. 136-140. (1985)
3. Werner, F.: Genetic algorithms for shop scheduling problems: a survey. Preprint Series. 11. 1-66. (2011)
4. Yigit, T., Birogul, S., Elmas, C.: Lot streaming based job-shop scheduling problem using hybrid genetic algorithm. Scientific research and essays. 6. 2873-2887. 10.5897/SRE10.152. (2011)
5. Bierwirth C.: A generalized permutation approach to job shop scheduling with genetic algorithms. Operations-Research-Spektrum, June 1995, Volume 17, Issue 2-3, pp 87-92. (1995)
6. Omar, M., Baharum, A., Hasan, A. Y.: A job-shop scheduling problem (JSSP) using genetic algorithm (GA). Proceedings of the 2nd IMT-GT Regional Conference on Mathematics, Statistics and Applications Universiti Sains Malaysia, Pennang, June 13-15. (2006)
7. Auger, A., Hansen, N.: A restart CMA evolution strategy with increasing population size. IEEE Congress on Evolutionary Computation, 1769-1776. (2005)
8. REFA Verband für Arbeitsgestaltung: REFA Methodenlehre der Betriebsorganisation: Datenermittlung. München: Carl Hanser Verlag. (1997)
9. Abdolrazzagh-Nezhad, M., Abdullah, S.: Job Shop Scheduling: Classification, Constraints and Objective Functions. World Academy of Science, Engineering and Technology International Journal of Computer and Information Engineering Vol:11, No:4, p. 423-428. (2017)
10. Seifermann S., Böllhoff J., Metternich J., Bellaghnach A.: Evaluation of Work Measurement Concepts for a Cellular Manufacturing Reference Line to enable Low Cost Automation for Lean Machining, Procedia CIRP 17 p. 588 – 593. (2014)
11. Maynard H, Stegemerten G, Schwab J.: Methods-Time Measurement. New York: McGraw-Hill. (1948)
12. Bierwirth C., Mattfeld D.C., Kopfer H.: On permutation representations for scheduling problems. In: Voigt HM., Ebeling W., Rechenberg I., Schwefel HP. (eds) Parallel Problem Solving from Nature — PPSN IV. PPSN 1996. Lecture Notes in Computer Science, vol 1141. Springer, Berlin, Heidelberg. (1996)
13. Bokranz R, Landau K. Produktivitätsmanagement von Arbeitssystemen. MTM-Handbuch (Productivity Management of Work Systems. MTMHandbook). Stuttgart: Schäffer-Poeschel. (2006)
14. Bundesministerium des Innern/Bundesverwaltungsamt (Hrsg.): Handbuch für Organisationsuntersuchungen und Personalbedarfsermittlung. (2018)
15. Poeschel, F.: Verteilzeit. In: Landau, Kurt (Hrsg.): Lexikon Arbeitsgestaltung : Best Practise im Arbeitsprozess. Stuttgart: Genter. (2007)
16. Sotskov, Y.N.; Matsveichuk, N.M.; Hatsura,V.D.: Schedule Execution for Two-Machine Job-Shop to Minimize Makespan with Uncertain Processing Times. Mathematics 2020, 8, 1314. (2020)
17. Sauvey, C.; Trabelsi, W.; Sauer, N. Mathematical Model and Evaluation Function for Conflict-Free Warranted Makespan Minimization of Mixed Blocking Constraint Job-Shop Problems. Mathematics 2020, 8, 121. (2020)
18. Luan, F.; Cai, Z.; Wu, S.; Liu, S.Q.; He, Y. Optimizing the Low-Carbon Flexible Job Shop Scheduling Problem with Discrete Whale Optimization Algorithm. Mathematics 2019, 7, 688. (2019)
19. Bak, N., Chang, B-M., N., Choi, K.: Smart Block: A visual block language and its programming environment for IoT. Journal of Computer Languages 60, 100999. (2020)
20. Brest, J., Greiner, S., Bošković, B., Mernik, M., Žumer, V.: Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems. IEEE Transactions on Evolutionary Computation 10(6), 646-657. (2006)
21. Črepinšek, M., Mernik, M., Žumer, V.: Extracting grammar from programs: brute force approach. ACM SIGPLAN Notices 40(4), 29-38. (2005)

22. Črepinšek, M., Liu, S.-H., Mernik, M.: Exploration and exploitation in evolutionary algorithms: A survey. ACM Computing Surveys 45(3): 35:1-35:33. (2013)
23. Črepinšek, M., Liu, S.-H., Mernik, L., Mernik, M.: Is a comparison of results meaningful from the inexact replications of computational experiments? Soft Computing 20, 223-235. (2016)
24. Črepinšek, M., Liu, S.-H., Mernik, M., Ravber, M.: Long term memory assistance for evolutionary algorithms, Mathematics 7 (11). (2019)
25. Du, Z., Chen, K.: Enhanced Artificial Bee Colony with Novel Search Strategy and Dynamic Parameter. Computer Science and Information Systems 16(3), 939–957. (2019)
26. Eiben, A,.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer, Heidelberg. (2003)
28. Hrnčič, D., Mernik, M., Bryant, B. R., Javed, F.: A memetic grammar inference algorithm for language learning. Applied Soft Computing 12 (3), 1006-1020. (2012)
29. Javed, F., Bryant, B. R., Črepinšek, M., Mernik, M., Sprague, A.: Context-free grammar induction using genetic programming, in: Proceedings of the 42nd Annual Southeast Regional Conference, ACM-SE 42, 404-405. (2004)
30. Jesenik, M., Mernik, M., Trlep, M.: Determination of a hysteresis model parameters with the use of different evolutionary methods for an innovative hysteresis model. Mathematics 8 (2). (2020)
31. Karaboga, D., Basturk, B.: On the performance of artificial bee colony (ABC) algorithm. Applied Soft Computing 8(1), 687-697. (2008)
32. Kovačević, Ž., Mernik, M., Ravber, M., Črepinšek, M.: From grammar inference to semantic inference-an evolutionary approach. Mathematics 8 (5). (2020)
33. Mei, H., Ma, Y., Wei, Y., Chen, W.: The design space of construction tools for information visualization: A survey. Journal of Visual Languages & Computing 44, 120 – 132. (2018)
34. Mernik, M., Liu, S.-H., Karaboga, D., Črepinšek, M.: On clarifying misconceptions when comparing variants of the artificial bee colony algorithm by offering a new implementation. Information Sciences 291, 115-127. (2015)
35. Grammatikis, P.I.R, Sarigiannidis, P.G., Moscholios, I.D.: Securing the Internet of Things: Challenges, threats and solutions. Internet of Things 5, 41-70. (2019)
36. Rao, V. R., Savsani, V., Vakharia, P. D.: Teaching-learning-based optimization: An optimization method for continuous non-linear large scale problems. Information Sciences 183, 1-15. (2012)
37. Rathee, A., Chhabra, J. K.: A multi-objective search based approach to identify reusable software components. Journal of Computer Languages 52, 26-43. (2019)
38. Russell, S., Norvig, P.: Artificial intelligence: a modern approach. Prentice Hall. (2002)
39. Tanabe, R., Fukunaga, A.: Success-history based parameter adaptation for differential evolution. In: IEEE Congress on Evolutionary Computation, pp. 71–78. (2013)
40. Qiu, X., & Lau, H. Y.. An AIS-based hybrid algorithm for static job shop scheduling problem. Journal of Intelligent Manufacturing, 25(3), 489-503. (2014)
41. Zhang, G., Gao, L., & Shi, Y.: An effective genetic algorithm for the flexible job-shop scheduling problem. Expert Systems with Applications, 38(4), 3563-3573. (2011)
42. Pinedo, M. L. (Ed.).: Scheduling, Theory, Algorithm and Systems. New York: Springer. (2012)
43. Kuroda, M., & Wang, Z.: Fuzzy job shop scheduling. International Journal of Production Economics, 44(1), 45-51. (1996)
44. Ahmad, F., & Khan, S. A.:Module-based architecture for a periodic job-shop scheduling problem. Computers & Mathematics with Applications, 64(1), 1-10. (2012)
45. Brucker, P., Burke, E. K., & Groenemeyer, S.:A mixed integer programming model for the cyclic job-shop problem with transportation. Discrete applied mathematics, 160(13-14), 1924-1935. (2012)
46. Ebadi, A., & Moslehi, G.: Mathematical models for preemptive shop scheduling problems. Computers & Operations Research, 39(7), 1605-1614. (2012)

47. Schuster, C. J., & Framinan, J. M.: Approximative procedures for no-wait job shop scheduling. Operations Research Letters, 31(4), 308-318. (2003)
48. Baptiste, P., Flamini, M., & Sourd, F.: Lagrangian bounds for just-in-time job-shop scheduling. Computers & Operations Research, 35(3), 906-915. (2008)
49. Zhang, R., & Wu, C.: A hybrid approach to large-scale job shop scheduling. Applied intelligence, 32(1), 47-59. (2010)
50. Topaloglu, S., & Kilincli, G.: A modified shifting bottleneck heuristic for the reentrant job shop scheduling problem with makespan minimization. The International Journal of Advanced Manufacturing Technology, 44(7), 781-794. (2009)
51. Jerebic, J., Mernik, M., Liu, S-H., Ravber M., Baketarić, M., Mernik, L., Črepinšek, M.: A novel direct measure of exploration and exploitation based on attraction basins. Expert Systems with Applications, Volume 167, 114353. (2021)

**Sašo Sršen** has graduated from the Faculty of Electrical Engineering and Computer Science, University of Maribor, in 2014. Currently, he is a Ph.D. student and co-owner of the company PISK, a licensed REFA/MTM company. He is working as a consultant/teacher for production companies. His research interests include domain-specific languages, evolutionary algorithms, Industry 4.0, predictive analytics, IOT, and simulation.

**Marjan Mernik** received the M.Sc. and Ph.D. degrees in computer science from the University of Maribor in 1994 and 1998, respectively. He is currently a professor at the University of Maribor, Faculty of Electrical Engineering and Computer Science. He was a visiting professor at the University of Alabama at Birmingham, Department of Computer and Information Sciences. His research interests include programming languages, compilers, domain-specific (modeling) languages, grammar-based systems, grammatical inference, and evolutionary computations. He is a member of the IEEE, ACM, and EAPLS. Dr. Mernik is the Editor-In-Chief of the Journal of Computer Languages, as well as Associate Editors of the Applied Soft Computing journal, Information Sciences journal, and Swarm and Evolutionary Computation journal. He is being named a Highly Cited Researcher for years 2017 and 2018. More information about his work is available at https://lpm.feri.um.si/en/members/mernik/

*Received: October 09, 2020; Accepted: December 30, 2020*