14th Annual International Conference on Industrial Engineering and Operations Management Dubai United Arab Emirates (UAE), February 12-14, 2024

Publisher: IEOM Society International, USA DOI: 10.46254/AN14.20240528
Published: February 12, 2024

A Digitalization Framework for Real-time Large-scale Production Scheduling

Neha Kadu, Shankar Prawesh and Avijit Khanra
Department of Management Sciences,
Indian Institute of Technology, Kanpur
India
nkadu@iitk.ac.in

Naoyuki Fujiwara, Yuichi Koga and Yosuke Watanabe
Digital Innovation Headquarters,
: 0360074 Mitsubishi Heavy Industries, Ltd., India

Abstract

In this work, we propose a novel framework for the development of real-time large-scale production scheduling. The problem that we study originated from the Digital Innovation Headquarters division of Mitsubishi Heavy Industries, Ltd. (MHI), which faced the challenge of developing a large-scale just-in-time (JIT) job-shop scheduling software that could schedule up to a quarter million jobs and 6.2 million operations within an hour. We designed the algorithm for scheduling and used list data structure for storing the resource availability. We used Ousterhout matrix as a benchmark, which is popularly used for slot scheduling and shows that our approach significantly improves the runtime. Our study highlights the implementation of data structures in Job-shop scheduling problems.

Keywords

Job-shop Scheduling, Digitalization, Frame work, Real-time Large-scale.

1.Introduction

Information and communication technologies (ICT) such as, Enterprise Resource Planning (ERP), Manufacturing Execution Systems (MES), Electronic Data Interchange (EDI), and more recently Internet of Things (IoT) have enabled us to collect detailed manufacturing, maintenance and operations related data from disparate sources such as sensors, embedded software feeds, and human interventions (Mithas et al., 2022). This digital transformation of the manufacturing ecosystem has greatly enabled the efficient use of input resources (Choi et al., 2022) and the adoption of lean manufacturing practices (Shah & Ward, 2003). In particular, the digitalization of manufacturing and services has generated various opportunities to improve the functioning of job shop scheduling problems (JSSP) to implement just-in-time (JIT) production (Ashton James, 1989). And it is imperative for the shop floor managers to embrace this digital transformation to develop the digital intelligence strategies to meet the objectives of lean manufacturing (Mithas et al., 2022).

Recently, Digital Innovation Headquarters division of a Japanese multinational company[1] faced the challenge of developing a large-scale JIT job-shop scheduling platform that could schedule up to a quarter million jobs and 6.2 million operations within an hour. These jobs were generated from various manufacturing plants located at multiple locations across the globe. Each job consists of a set of linearly ordered operations with a fixed precedence relation

[1] The name has been anonymized for peer review.

among them. Overall, the objective was to develop a just-in-time schedule honoring different resources, time and precedence constraints which vary substantially across each job.

This problem is unique and differs from a typical JSSP in many ways. First, because of its scale, i.e., despite a high mix of jobs, we also have their high-volumes. Second, the definition of resource, which is not limited to a machine, but can also refer a worker, land, or transportation. Because of this broader definition of resource, each resource can exhibit different working shifts. Third, the scheduling horizon spans over a few years therefore, it necessitates honoring the constraints related to working days, daily shifts, limited quantity of a resource, and release and due dates of a job. In addition to the problem characteristics highlighted above, shop floor managers often make up for delays by changing the order of operations, and they have limited time to spare for generating new schedules. Therefore, we need a production planning approach that generates a valid schedule honoring all constraints in a short or real time. All these challenges call for the use of carefully designed scheduling algorithms and data structure to process the data generated from diverse sources.

Job-shop problem is an extensively researched area, and in a typical JSSP, n-jobs are processed on m-machines such that each job has a specific precedence relation for processing its operations, each operation can be processed by only one machine at a time, each machine can process only one operation at a time and preemption of operations is not allowed (Pinedo 2012). JSSP can have different objectives such as minimization of makespan (time to finish the last job), tardiness (lateness of a job) or lead time. It is a complex combinatorial optimization problem, and in general most JSSP are NP-hard (Pinedo 2012). JSSP usually exhibits high-mix, low-volume characteristics i.e., each job follows a substantially different path for the processing of its operations to manufacture the end product, and the quantity to be produced is small. There are well established test instances (van Hoorn, 2018) exhibiting above characteristics of JSSP and early efforts had been to develop good local search heuristics (Balas & Vazacopoulos, 1998; Vaessens et al., 1996) to solve the selected test instances. Subsequently the use of machine learning (Ferreira et al., 2022; Zhang et al., 2019), and more recently quantum computing (Kurowski et al., 2023) have also gained prominence.

Another class of heuristics that is popular in scheduling is dispatching rule (Ferreira et al., 2022). A dispatching rule determines the priority of a job based on some characteristics of the job and the current state of the shop floor, and the job/operation with highest priority is selected for servicing. Unlike the search heuristics, dispatching rules have much less computational complexity, hence they are widely used in practice. In the prior research, dispatching rules and their modifications have been used to schedule up to 10^5 operations (E. C. Teppan, 2018). Giffler and Thompson algorithm is often used with dispatching rules to generate an active schedule (Sha & Hsu, 2006). However, it assumes that the quantity of a resource is one, for a job a particular resource is used by only one of its operations, and a resource is continuously available. Overall, all these studies lack the complexities of the large-scale real-life production scheduling that we highlighted earlier and therefore their direct application is not useful in our context.

In service industry, such as restaurant the impact of just-in-time scheduling on workers while considering their work shift has also received attention (Kamalahmadi et al. 2021). Especially, in the context of job-shop scheduling, Yau and Shi (2009) considered working shift and bill of materials constraints in a much simplistic setup.

The Use of ICT technologies to implement and improve JIT practices has also been investigated in IS research. For instance, by conducting a field study and using firm level data, Srinivasan et al. (1994) found that EDI technology facilitates the coordination of JIT material flows between different trading parties. In other related studies Banker et al. (2006) used dynamic capability theory to show that plant information systems have significant impact on developing JIT production practices, whereas Rai et al. (2006) highlight the role of IT infrastructure integration within and across its boundaries for physical flow integration.

The above discussion on extant research reveals that in general there is lack of a framework that helps us leverage the data collected from different sources to develop a good schedule for large-scale JSSP in real-time. While discussing the emerging challenges in operations management from AI and Industry 4.0 technologies Mithas et al. (2022) also recognize this: "*Mathematical models in OM (e.g., …, and scheduling models) are built on underlying assumptions that reflect the technological reality of the time when the model was created. As manufacturing and information technologies evolve, these assumptions and model setups may need to be revised to make these models relevant in the era of Industry 4.0 technologies."* In this context, the access to unique and detailed data for a job and its operations, resources, and the other real-life constraints allows us to propose a digitalization framework for real-time large-scale scheduling of high mix, high volume jobs. While presenting the solution, we highlight the major data preparation and

scheduling challenges that arise in this process and design a set of novel algorithms for calendar timestamp generation, resource update and scheduling honoring shift breaks. Overall, this work makes the following major contributions to the product digitalization, operations management, and algorithm design literature.

2. Objectives

1. To develop a novel real-time scheduling framework for JSSP honoring a resource's calendar and its quantity. In particular, our algorithm elegantly overcomes the computationally demanding task of finding a valid working shift of a resource with limited quantity.
2. The scheduling framework that enables the seamless integration of large-scale scheduling algorithms with shopfloor software for resource monitoring and job updates.
3. Finally, develop a framework that also provides the foundation for the application of AI/ML for the automation of large-scale job shop scheduling using dispatching rules.

3. Problem Description

We now describe the unique characteristics of the JSSP studied here. A snapshot of the data used to determine different scheduling constraints is mentioned in Figure 1. We use tuples to refer to a specific panel within the referenced figure[2]. Each job that enters the shop floor is assigned a unique job-id and its release and due dates are noted. Figure (1,1) shows the sample data entries for a job.

Each job can have multiple operations and each operation has the job specific unique id, see Figure (1,2) where each row represents the operation specific details. In this figure column 'operation' shows all operations corresponding to job 1. Operations within a job follow a predetermined order and the precedence constraints take the form of a chain. The first and the last operations within a job are marked with 'start' and 'finish' flags, respectively. See column 'position' in Figure (1,2) which presents the processing order of the operations in job 1. The entry sequence of each operation follows the processing sequence of operations in the job. For example, in job 1, the processing of operations must follow the following sequence: $13\ (start) \rightarrow 14 \rightarrow 15 \rightarrow 3 \rightarrow 35 \rightarrow 5 \rightarrow 8 \rightarrow 9 \rightarrow 10\ (finish)$. The column 'resource' mentions the resource required to process the corresponding operation and 'duration' represents the processing time in minutes. For illustration, consider operation 13 of job 1 which requires processing of 20 minutes on resource 3.

While assigning an operation to the designated resource, we must consider its quantity and working shifts. For instance, the quantity of resource 3 used by operation 10, is 12 and it uses calendar 1 (see Figure (1,3)). Note that in some cases, the quantity of resources is entered as ∞, because the availability for these resources always exceeds the requirement. Resources with quantity ∞ are marked as Type-1, whereas the resources with limited quantities are marked as Type-2.

Each calendar, identified by its ID, has predetermined working days (Figure (1,4)) and can be used to determine the time horizon for scheduling. Further, corresponding to each working day we can have different work shifts identified by its shift ID (Figure (1,5)). We continue with the example of resource 3, which uses calendar 1, and on date 7 January 2021 it uses shift 1 whose time slots are: 8:00-12:00 and 13:00-17:00.

There are some resources without shift break, and their 'start_time' and 'finish_time' both are marked as 0:00. Resources with limited quantity and without shift break are marked as Type-20, whereas resources with large availability and without break are marked as Type-10. Type-10 resources are always available for processing and typically, they are used for routine services such as transportation.

Some scheduling constraints are also illustrated in Figure 2. The successor operation in a job can only be processed after completing the processing of current operation (Figure (2,1)). All operations are non-preemptive i.e., once the processing of an operation has started, it must be completed before assigning another operation to the same resource. On a resource, an operation cannot be processed during the shift break (Figure (2,2)). If the processing requirement of an operation exceeds the time available in the current shift of the desired resource, then its processing must resume after the shift break, and it continues until completion. Scheduling on a resource must use only working days available for the resource (Figure (2,3)). For example, weekends and public holidays may not be used for some resources.

[2] For example, tuple (1, 1) refers to Figure 1 and panel 1, respectively. Panel number is indicated at the bottom of each panel.

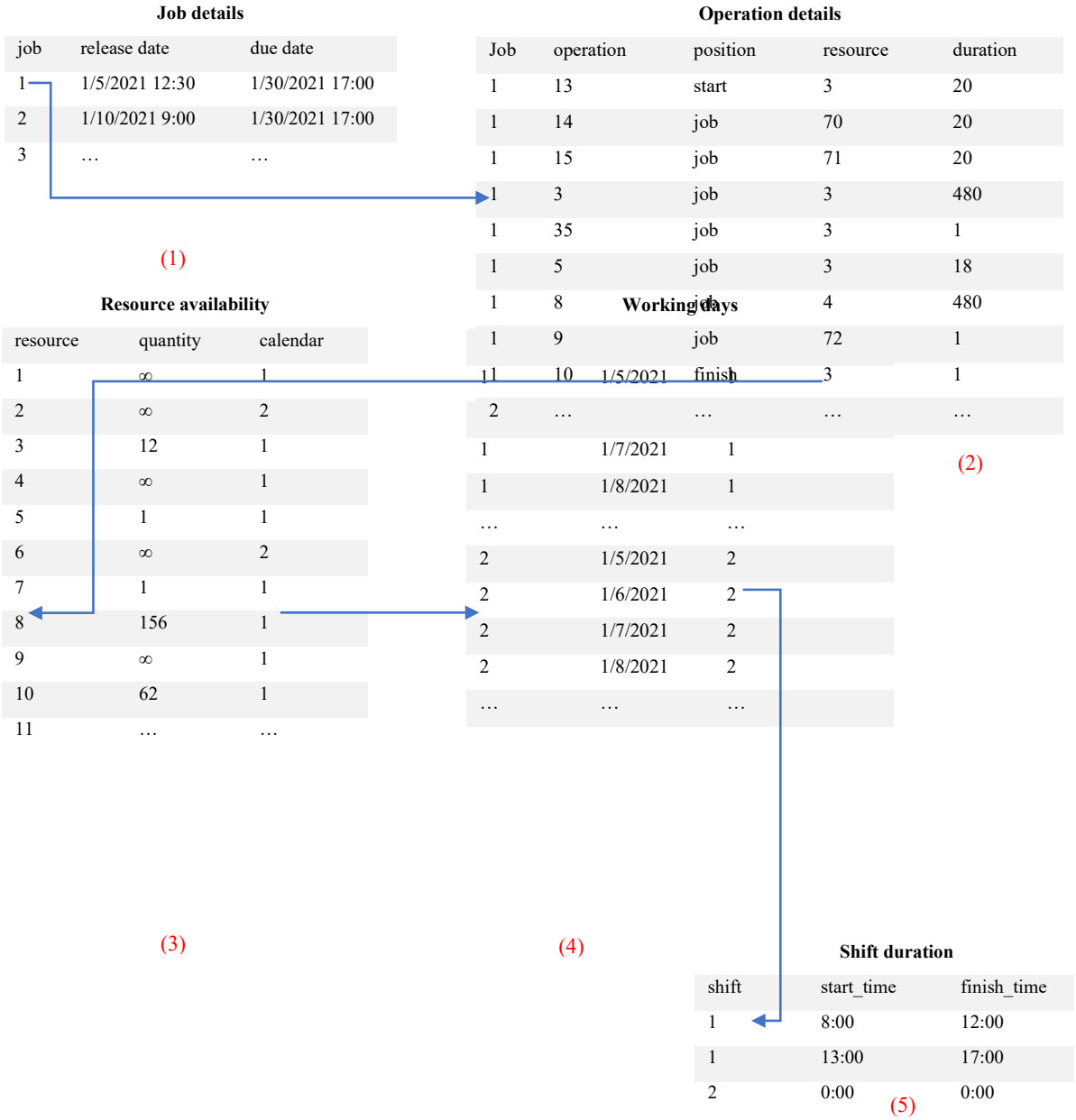

Figure 1. Production data on job processing and resource availability constraints

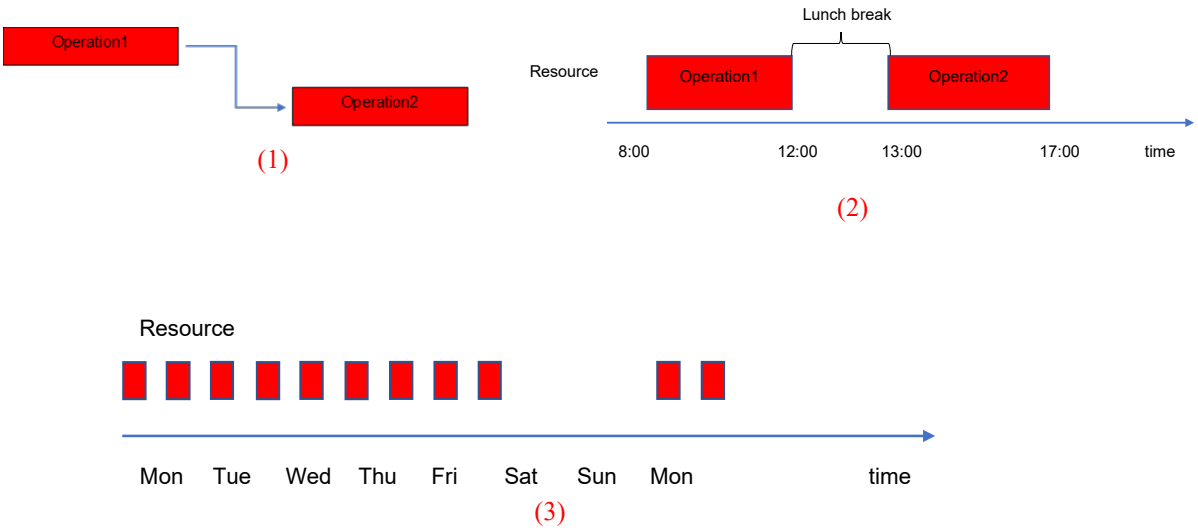

Figure 2. Illustration of operation processing and resource shift constraints

Note that unavailability of a resource due to shift break has a different characteristic than unavailability due to pre-scheduled operation on the resource. We highlight this difference using a specialized example in Figure 3 which presents a snapshot of two operations scheduled on Type-2 resource with quantity 1.

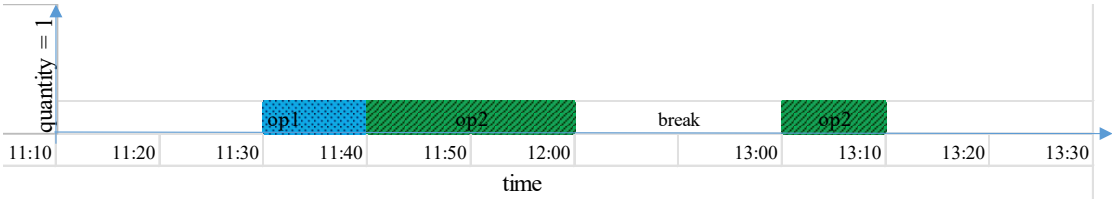

Figure 3. Unavailability of a resource due to limited quantity and shift break

Suppose operation, op2, requires processing of thirty minutes on the resource and op1 has already been scheduled for processing during the time interval of 11:30-11:40. The resource is only available for twenty minutes beginning at 11:10, therefore we cannot start processing op2 at 11:10. Whereas if the processing of op2 starts at 11:40, then again, we encounter a shift break of one hour after twenty minutes. However, the remaining processing for op2 can be finished after the break. We differentiate the unavailability of a resource due its available quantity becoming zero during a time interval and due to the shift breaks using different flags in the timestamp generation algorithm that we discuss in the following section.

4. Data Preparation and Scheduling Algorithms

Scheduling algorithms are usually designed to minimize objective(s) functions such as, makespan, lead time, tardiness etc. In our context, we would like to generate just-in-time (JIT) schedule which minimizes earliness, tardiness, and lead-time, i.e., to the extent possible a job should finish near its due date. However, as we highlight in Algorithm 3, generating a valid schedule in our context gives rise to new challenges related to design and execution of data structure used for storing and searching for valid timeslots on a resource with limited quantity. Therefore, we limit our discussion to the set of algorithms used for job shops with shift and resource quantity constraints and defer the discussion on selecting a good dispatching rule for JIT schedule for future work.

To store and access the information relating to resource availability, we design a timestamp generation algorithm (Algorithm 1). As more and more operations are allocated on a limited resource, this algorithm updates the availability of the resource using quantity, calendar, processing time, and shift details.

Algorithm 1: Timestamp generation for a resource

Input: Resource availability and working days data
Output: Timestamps for all resources
Step-1: $T \leftarrow$ current time for generating schedule
Step-2: **if** the resource is of Type-2
Set q ← 'quantity'
else
set q ← 1
end if
Set c ← 'calendar' of the resource, D_c ← dates corresponding to the calendar 'c'
Step-3: Set $t_l \leftarrow 0$
Step-4: **for** each date in D_c
Identify its shift: $x \leftarrow$ '$shift$'
Identify all tuples $(start\_time, finish\_time)$ of x, do the following for each such pair
a. Set $t_{xs} \leftarrow$ '$start\_time$' and $t_{xf} \leftarrow$ '$finish\_time$'
b. **if** $t_{xs} > t_l$
i. Add t_{xs} as a timestamp with q as the availability
c. **else if** $t_{xs} = t_l$
i. Delete timestamp t_l and its availability. Note that $t_{xs} \not< t_l$
d. Add t_{xf} as a timestamp with -1 as its availability and set $t_l \leftarrow t_{xf}$
end for
Step-5: Delete all timestamps $< T$ (if any)
Step-6: Identify the first timestamp, denoted by t_1.
if $T < t_1$
if availability of t_1 is -1
Add T as a timestamp with availability q
else if: availability of t_1 is q
Add T as a timestamp with availability -1
end if
Step-7: Replace the last timestamp t_l by $t_l + 100$ years with its availability unchanged, i.e., -1

Step 1 in Algorithm 1 takes input from the user, the time (T), at which a new schedule is generated for the job shop. Step 2 is used to note the quantity of limited resources. For those resources whose availability exceed the requirement we only need flags to determine the beginning of a shift break, therefore we use $q = 1$, and $q = -1$ to denote the beginning and break of a shift, respectively. Steps 3-4 are used to store working shifts corresponding to all working days in D_c for the resource. The check in step 4.c is used to create timestamps for the resources without shift break. Step 5 removes timestamps before T, and step 6 adds T and its corresponding availability depending upon whether it falls during a working shift or a shift break. Step 7 is used for protection against insufficient working days in calendar. Algorithm 1 is run for each resource to update its availability beyond the scheduling time T. Let us denote the number of days in calendar D_c as $|D_c|$. Then for a resource with b shift breaks per day, initially we create $|D_c|(2b + 2)$ timestamps. The timestamps created for a resource must be stored to enable honouring shift constraints for scheduling. We use a customized implementation of map data structure and Ousterhout matrix to update the resource usage and to honour the shift constraints. We now discuss the scheduling framework using a dispatching rule.

Algorithm 2: Job prioritization using critical ratio

Input: an empty map Ω
Output: Complete Schedule, and resource usage update
Step-1: **for** each job:
a. Get J_0, the set of unscheduled operations of the job. Operations in J_0 appear from the first to the last in accordance with the precedence relations.

b. Compute $(d - \tilde{r})/\sum_{j \in J_0}(p_j)$ where d is due date and $\tilde{r} = \max\{r, T\}$, where r is release date of the job and p_j denotes the processing time of operation-j of the job.
c. Store the job id as key and the above metric as value in Ω.

end for

Step-2: Sort pairs in Ω in the ascending order of the metric.
Step-3: **for** each element in Ω
a. Run *Algorithm-3* with other relevant data files.
b. For each operation $j \in J_0$, run *Algorithm-4* with inputs: current job, S_j (output of *Algorithm-3*), and other relevant data files.

end for

Algorithm 2 uses critical ratio as dispatching rule to determine the priority of a job. Step 1 computes the critical ratio of each job, and Step 2 sorts this metric. In Step 3, Algorithm 3 schedules operations of the current job, from first to last, so that the job completes at the earliest, whereas Algorithm 4 updates the data files for resource usage.

Algorithm 3: Scheduling a job

Input: Relevant data files
Output: S_j, the schedule for each operation in the job
Step-1: Set: the release time of job-i, $x \leftarrow r_i$; $J_0 \leftarrow$ the unscheduled operations of job-i
Step-2: Create list S_j for each $j \in J_0$
Step-3 **for** each $j \in J_0$, do the following:
(i) Get its processing time p, and the resource requirements
(ii) **if** the resource is Type-1, do the following
(a) Find the smallest timestamp $> x$, denote it by t
(b) **if** availability of t, $q_t = -1$,
set $u \leftarrow x$, $v \leftarrow \min\{u + p, t\}$
else
set $u \leftarrow t$, $v \leftarrow \min\{u + p, t^+\}$, t^+ is the next timestamp of t
(c) Add (u, v) to S_j and set $p \leftarrow p - (v - u)$, $x \leftarrow v$
(d) **if** $p > 0$
set $u \leftarrow v^+$, $v \leftarrow \min\{u + p, u^+\}$, resume from Step-3.(ii).(c)
end if
end if
(iii) **if** the resource is Type-10
add $(x, x + p)$ to the list S_j and set $x \leftarrow x + p$
end if
(iv) **if** the resource is Type-2 or 20, set $p_R \leftarrow p$ and do the following
(a) Find the highest timestamp $\leq x$, denote it by w
(b) **if** $q_w \geq 1$
set $u \leftarrow x$
else
Find t, the first timestamp after w with availability ≥ 1. Set $u \leftarrow t, w \leftarrow t$
(c) Find the first timestamp after w with availability < 1, denote it by t
(d) Set $v \leftarrow \min\{u + p_R, t\}$, add (u, v) to S_j. Set $p_R \leftarrow p_R - (v - u)$, $x \leftarrow v$
(e) **if** $p_R > 0$
if $q_v = -1$ and $q_{v^+} \geq 1$,
set $u \leftarrow v^+$, $w \leftarrow v^+$, resume from Step-3.(iv).(c)
else
clear S_j, set $p_R \leftarrow p$, $w \leftarrow v$, resume from Step-3.(iv).(b)
end if
Step-4: Set $y \leftarrow x$ (completion time of the job)

Algorithm 3 schedules a job. For the Type-1 resource, the complexity of step-3.(ii).(a) is $O(ts_c)$ for the list of timestamps where $ts_c = |\mathrm{D_c}|(2b+2)$, and O(.) refers to Big-O notation. The complexity of finding next greater element (NGE) using stacks for a list of size, ts_c, is $O(ts_c)$. Therefore, the complexity of step-3.(ii).(b) is $O(ts_c)$. If the shift length per day in hours for calendar 'c' is l_c, then step-3.(ii).(c-d) can be completed in at most $\left(1+\left\lceil\frac{p}{l_c}\right\rceil\right)$ iterations. Therefore, the complexity of scheduling on Type-1 resource is $O\left(\frac{p}{l_c} * ts_c\right)$. Whereas, for the Type-10 resource, the complexity of scheduling is of constant order.

For a Type-2 resource, its availability must be updated after scheduling an operation. Therefore, timestamps for a resource does not remain constant at the initial value of ts_c rather it changes as more operations are scheduled on the resource. We differentiate the updated timestamps from the initial ts_c using $ts_c{}^*, (ts_c \le ts_c{}^* \le ts_c + 2m^*)$, where m^* is the number of operations to be scheduled on the resource. Step-3.(iv).(b-e) are executed in a loop and therefore these steps dominate in determining the overall complexity.

For the ease of exposition, we present the worst-case analysis which helps us to show the polynomial dependence of this algorithm on the number of current timestamps for a limited resource. Step-3.(iv).(b) has linear dependence on $ts_c{}^*$, whereas Step-3.(iv).(c) has linear dependence on l_c. The minimum processing requirement for an operation is one minute, therefore at most $\left(\frac{l_c}{1}\right)$ searches are required. Therefore, the complexity of Step-3.(iv).(b-d) is $O(ts_c{}^*)$.

As more and more operations are scheduled on a resource, the 'else' condition is repeatedly executed in Step-3.(iv).(e). If the resource is extremely occupied for the given calendar dates, then $p_R > 0$ will hold for almost all timestamps, and hence the incomplete schedules for the operation are deleted and execution of Step-3.(iv).(e) is of order $O(ts_c{}^*)$. Therefore, the overall complexity of Step-3.(iv).(b-e) is a polynomial of degree two for the current timestamps, i.e. $O\left(ts_c{}^{*2}\right)$. For a large-scale scheduling these steps govern the total computational requirements.

Algorithm 4: Updating the resource usage

Input: current job, S_j (output of *Algorithm*-3), and other relevant data files.
Output: update resource availability
Step-1: Get S_j, and the resource details of the operation j, i.e., resource ID, and resource type
Step-2**: if** the resource is Type-2/20 do the following
(i) Get the first entry of S_j, and set $t_S \leftarrow (u,)$, the first entry in tuple (u, v)
(ii) Get the last entry of S_j, and set $t_E \leftarrow (, v)$, the second entry in tuple (u, v)
(iii) Identify the highest timestamp $\le t_S$, denote it by t
(iv) **if** $t_S \neq t$
add t_S as a timestamp and set $q_{t_S} \leftarrow q_t$
end if
(v) Repeat Step 2.(iii-iv) for t_E
(vi) **for** all timestamps $t \in [t_S, t_E)$,
reset availability as: $q_t \leftarrow \max\{-1, q_t - 1\}$
end for
(vii) **if** $q_{t_S} = q_{t_S^-}$, t_S^- is the timestamp immediately before t_S
Remove t_S as a timestamp along with its availability
end if
(viii) Repeat Step2.(vii) for t_E
end if

Algorithm 4 updates availability of resources after scheduling a job. Step 1 in Algorithm 4 gets S_j, i.e., the schedule of the operation j, the resource ID and its type. We do not need to update the quantity of Type-1/10 resources, but scheduling on Type-2/20 resources entail that their availability must be updated. Steps 2.(i-v), insert new timestamps, if required, for scheduling the operation. Step 2.(vi) updates the quantity of the resource after scheduling the operation. Steps 2.(vii-viii) remove redundant entries corresponding to the resource availability. The complexity of step 2.(iii-viii) is $O(ts_c{}^*)$, hence the overall complexity of Algorithm 4 is $O(ts_c{}^*)$.

We use map data structure to store the details of a limited resource. The resource ID is used as a key, and the corresponding value is the list of timestamps generated using Algorithm 1. We also store the resource's quantity corresponding to its timestamp.

4.1 Ousterhout Matrix for Shift Scheduling

Ousterhout matrix is widely used for slot scheduling (Aggarwal & Sarangi, 2013). In this approach time is a discrete quantity (rows) and it is divided into discrete quanta (columns) called slots. We adapt the concept of slot scheduling to store the availability of a resource at different timeslots and use it as one of the benchmarks. Usually, an entry in the matrix takes Boolean value indicating whether the resource is free or busy. However, in our case each resource can have multiple quantities, therefore cell entries represent the available quantity of the resource or a shift break.

In our context, the processing request on a resource can be as small as one minute. Therefore, we design Ousterhout matrix that shows resource availability for every minute. The calendar days span over almost two years, hence we need approx. $2 \times 365 \times 24 \times 60 \approx 10^6$ cells for each resource which consumes too much memory for the data structure and results in programming error.

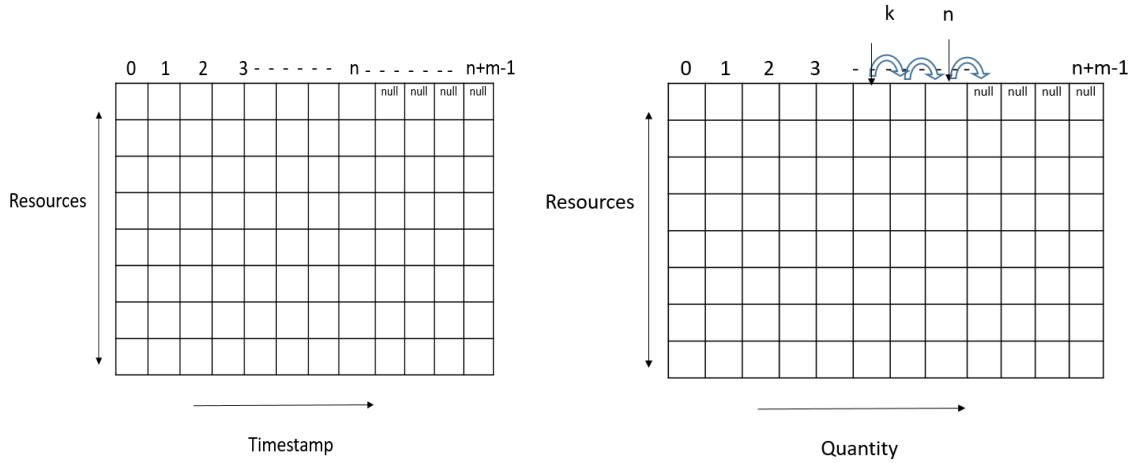

Figure 4. Ousterhout matrix for storing resource timestamps (left panel) and quantity (right panel)

Therefore, we adapted the Ousterhout matrix for our context, and we call it Timestamp matrix (left panel, Figure 4). Each row represents a resource, and a column represents the timestamps generated in Algorithm-1, and each cell stores a timestamp. This matrix has $n + m$ columns, where $n = ts_c$, i.e., the initial number of timestamps for a resource and m is additional timestamps added during the execution of the algorithm. The cell values corresponding to additional timestamps are initially 'null'.

Since resources can have different quantities, we maintain another matrix of same dimension (right panel, Figure 4) to store quantity corresponding to a timestamp. In order to update the resource availability at k^{th} position, the entries of both the matrices beyond the column k must be shifted to the adjacent cell as shown in Figure 4 (right panel). As the number of timestamps are large, shifting the entire matrix is computationally expensive. The complexity analysis of updating the Ousterhout matrix is mentioned in Technical Appendix[3].

4.2 Dataset and Results

We use a subset of job shop instances to illustrate the main findings. The job shop instance that we study (Table 1) here has 6000 and 100000 jobs and operations, respectively.

[3] It has been omitted here due to lack of space.

Table 1. Job Shop Instance Details

Attribute	Values
Number of jobs	6000
Total number of operations[4]	100000
Resources with unlimited quantity	2200
Resources with limited quantity	600
Scheduling horizon	5 January 2015 - 31 December 2016
Working shifts per day	8:00 -12:00; 13:00-17:00
Initial number of timestamps for Type-2 resources	~2100
Maximum number of new timestamps added for a limited resource after scheduling	~700
Operations requiring processing on a limited resource	20,167

For the computational experiments, we used a computer with following specifications: Intel®, core™ i7-6500U, CPU @ 2.50 GHz and 8 GB RAM, Windows 10, 64 bits. The total runtime for the Ousterhout matrix and our scheduling algorithms were 6 and 0.6 minutes, respectively. Our approach leads to almost 10 times reduction in runtime for this test case. As we move towards scheduling millions of operations in real-time this gain will have significant implications. As highlighted earlier, the major challenge in generating an implementable scheduling algorithm is finding a free slot for scheduling on a limited resource quickly while honoring all constraints. We are developing a new set of algorithms for this specific task to reduce the polynomial dependence of Algorithm 3 on the timestamps, i.e., $O(ts_c{}^{*2})$, and intend to share these findings in near future.

4.3 Implications and Conclusion

In this research, we develop a set of algorithms for large-scale JSSP which considers many real-life constraints such as, daily working shifts, release and due dates for a job, multiple quantity of a resource, and calendar dates. Incorporating all these constraints in a real-time scheduling algorithm calls for developing a novel scheduling framework that simplifies the process for finding the valid timeslot and updating the quantity of each limited resource. We achieve it by developing algorithms for maintaining resource availability over the given time horizon, job prioritization, scheduling, and resource update. We also highlight the role of careful selection of data structures for storing and retrieving different information related to real-life scheduling. We selected Ousterhout Matrix as a benchmark and show that our approach significantly improves the runtime significantly. These all are novel contributions of this work, and our scheduling framework creates new opportunities for designing and selecting better dispatching rules for real-time large-scale scheduling.

The importance of JIT scheduling is well recognized in manufacturing (Srinivasan et al., 1994). The advent of Industry 4.0, and the digitalization of manufacturing processes has enabled real-time interaction between digital and physical resources through "smart connections", and manufacturing and production related data collection from disparate sources such as sensors, information and communication technologies, and monitoring instrumentation (Choi et al., 2022). However, the opportunities created by the digitalization of manufacturing processes using these technologies have not been leveraged to a large extent to improve the JIT manufacturing practices[5]. In this context, our research makes a novel contribution by developing a scheduling framework for real-life large-scale JSSP.

In the current work we only use one dispatching rule, namely critical ratio, to demonstrate the promise of our framework in generating real-time scheduling. However, production scheduling must be designed to respond to real-

[4] We used 100,000 operations here and have planned to share the results for larger instances in the conference presentation.

[5] https://www.lean.org/the-lean-post/articles/why-lean-fails-in-job-shops-and-what-to-do-to-succeed/

time deviations in terms of new job arrivals, shop floor condition etc. In this context, machine learning and AI techniques are increasingly used for job-shop scheduling (Choi et al., 2022). Our work could be extended to switch the dispatching rules used for scheduling under different shop conditions. This selection can be made using a machine learning model that identifies a best dispatching given shop, job, and external constraints such as, scheduling horizon, due and release dates of a job (Parente et al., 2020). This is a direct extension of the current work, which we are working on and plan to share these results soon.

The processing of an operation may require multiple resources, perhaps with different working shifts. Also, in real-life a schedule must consider the interruptions due to maintenance or unavailability of a worker. These are also the possible extensions that we are working on in the ongoing research.

References

Aggarwal, P., Sarangi, S. R., Lock-Free and Wait-Free Slot Scheduling Algorithms. *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, 961–972, 2013.

Ashton James, C. F., Time to Reform Job Shop Manufacturing. *Harvard Business Review, 1989*.

Banker RD, Bardhan IR, Chang H, Lin S., Plant information systems, manufacturing capabilities, and plant performance. *MIS Quarterly*: 315–337, 2006.

Choi, T., Kumar, S., Yue, X., Chan, H., Disruptive Technologies and Operations Management in the Industry 4.0 Era and Beyond. *Production and Operations Management*, *31*(1), 9–31, 2022.

E. C. Teppan, Dispatching rules revisited - a large scale job shop scheduling experiment. *IEEE Symposium Series on Computational Intelligence:* 561–568, 2018.

Ferreira, C., Figueira, G., Amorim, P., Effective and interpretable dispatching rules for dynamic job shops via guided empirical learning. *Omega*, *111*, 102643, 2022.

Yau H., Shi L., Nested partitions for the large-scale extended job shop scheduling problem. *Annals of Operations Research*, *168*, 23–39, 2009.

Kamalahmadi M, Yu Q, Zhou YP., Call to Duty: Just-in-Time Scheduling in a Restaurant Chain. *Management Science* 67(11):6751–6781, 2021.

Kurowski, K., Pecyna, T., Slysz, M., Różycki, R., Waligóra, G., Węglarz, J., Application of quantum approximate optimization algorithm to job shop scheduling problem. *European Journal of Operational Research*, *310*(2), 518–528, 2023.

Mithas, S., Chen, Z.-L., Saldanha, T. J., De Oliveira Silveira, A. , How will artificial intelligence and Industry 4.0 emerging technologies transform operations management? *Production and Operations Management*, *31*(12), 4475–4487, 2022.

Parente, M., Figueira, G., Amorim, P., Marques, A., Production scheduling in the context of Industry 4.0: Review and trends. *International Journal of Production Research*, *58*(17), 5401–5431, 2020.

Pinedo ML., *Scheduling: Theory, Algorithms and Systems* (Springer, New York), 2012.

Rai A, Patnayakuni R, Seth N., Firm performance impacts of digitally enabled supply chain integration capabilities. *MIS Quarterly*: 225–246, 2006.

Sha, D. Y., Hsu, C.-Y., A hybrid particle swarm optimization for job shop scheduling problem. *Computers & Industrial Engineering*, *51*(4), 791–808, 2006.

Shah, R., Ward, P. T., Lean manufacturing: Context, practice bundles, and performance. *Journal of Operations Management*, *21*(2), 129–149, 2003.

Srinivasan, K., Kekre, S., Mukhopadhyay, T., Impact of Electronic Data Interchange Technology on JIT Shipments. *Management Science*, *40*(10), 1291–1304, 1994.

Storer, R. H., Wu, S. D., Vaccari, R., New search spaces for sequencing problems with application to job shop scheduling. *Management Science*, *38*(10), 1495–1509, 1992.

Vaessens, R. J. M., Aarts, E. H. L., Lenstra, J. K., Job Shop Scheduling by Local Search. *INFORMS Journal on Computing*, *8*(3), 302–317, 1996.

van Hoorn, J. J., The Current state of bounds on benchmark instances of the job-shop scheduling problem. *Journal of Scheduling*, *21*(1), 127–128, 2018.

Zhang, J., Ding, G., Zou, Y., Qin, S., Fu, J., Review of job shop scheduling research and its new perspectives under Industry 4.0. *Journal of Intelligent Manufacturing*, *30*(4), 1809–1830, 2019.