Journal of Turkish

Operations Management

A Monte Carlo simulation approach to the gap-time relationship in solving the Job Shop Scheduling Problem

Oguz Toragay[1*], Shaheen Pouya[2*]

[1]Mechanical, Robotics, and Industrial Engineering, Lawrence Technological University, Detroit, MI, USA
otoragay@ltu.edu, ORCID No: https://orcid.org/0000-0003-0690-2198
[2]Industrial and Systems Engineering, Auburn University, Auburn, AL, USA
szp0155@auburn.edu, ORCID No: https://orcid.org/0000-0003-2436-8849
*Corresponding Author

Article Info

Article History:

Received:	20.04.2023
Revised:	23.05.2023
Accepted:	26.05.2023

Keywords:
Integer programming,
Scheduling problem,
Random generation,
Solving time approximation,
Combinatorial optimization

Abstract

This article presents a study on the job shop problem, a combinatorial optimization problem that models scheduling and resource allocation in industrial settings. The article aims to investigate the relationship between optimality gap and required computational resources, considering various optimality gap levels that are applicable in real-life situations. The study uses a Monte Carlo simulation to analyze the behavior of solvers in solving different sizes of random-generated scheduling problems. The findings of the study offer insights into the worthiness of reaching an optimal solution versus implementing a near-optimal solution and starting the work. The codes used in the study are accessible on the author's GitHub account.

1. Introduction

The scheduling problem or job shop problem is considered one of the most important and well-studied combinatorial optimization problems that can be modeled as an integer programming (IP) problem. Its applications cover many industrial cases such as manufacturing and resource allocation (Kis, 2002). In this work, different instances of the scheduling problem are being solved to study the correlation between optimality gap or the results' quality on one hand and the required solving time or computational resources on the other hand. We considered various optimality gaps during the solving process of each problem and analyzed the required time to reach the considered optimality gap (solution quality) to meaningfully describe the worthiness of reaching optimal solution. The gaps are chosen based on their applicability in real-life situations. Most of the time, from the industry point of view, reaching the 5%, 10%, or even 20% of optimality gap can save huge number of resources for the company. Under those conditions, from the managerial point of view, it will not be worthy for the company to spend huge amount of computational power and time, to reduce the optimality gap while by implementing the near-optimal solution the tasks can be scheduled and the saved computational time then would be used for more profitable activities. In plain language, we want to answer this critical question: “Is it worth it to reach the optimal schedule while we can implement partial schedule, start doing the jobs, and making money?” To this end, understanding the behavior of the solvers in solving different sizes of scheduling problems is important. We designed a Monte Carlo simulation to achieve a better statistical understanding and trustworthy rates of time in solving different instances of the problem.

The remainder of this work is organized as follows. After a brief review of the related works in the literature in Section 2, we first describe the problem by providing the considered mathematical model for the problem in hand. In addition, we also explain the process of randomly generating instances of selected problem size in Section 3. We then explain the solving approach and the obtained results while a comprehensive analysis of the results of various instances of the problems with various sizes, as well as statistical approaches to interpret the results toward estimating the effect of gap will be explained in Section 4. We then conclude this work by providing next potential steps in this research and recommend future works in this area.

2. Literature Review

The scheduling problem, (a.k.a. job shop problem, permutation flow shop problem) involve determining the optimal order and timing of processing a set of jobs on a set of machines, considering various constraints such as machine availability, job precedence, and processing times (Garey, 1976; Chankong and Haimes, 1983). Integer Programming (IP) has been widely used for modeling of scheduling problems where these problems are well-studied in operations research literature. The objective would be the minimization of some forms of cost, such as completion time, tardiness, makespan, or it can be the maximization of some form of benefit, such as profit, earliness, or number of jobs done by due date or even target-defense and chemical ionic properties of a mixture (Belotti, 2013; Taha, 2013; Sattarkhan, 2023; Pourghorban et al, 2022; Mokhtari et. Al, 2022).

Using IP allows for a comprehensive and more accurate modeling of the problem, as integer variables are often required to model constraints such as machine availability and job precedence (Linderoth and Wright, 2005). IP models have proven to be a useful tool for solving scheduling problems in various industries, including manufacturing, project management, and logistics (Pferschy and Stanek, 2017). Computation time is a critical aspect of job shop problems modeled as IPs, which can vary significantly based on several factors, including the size of the problem, the complexity of the constraints, the presence of integer variables, and the implemented solution method.

In the theory of computing and operations research community, the concept of NP-hardness is used to explain the correlation between the size of problem in one hand and the required solution time of the other hand. An NP-hard problem is one for which no known algorithm exists that can solve the problem in polynomial time, meaning that the amount of time required to solve the problem increases exponentially as the size of the problem increases (Garey and Johnson, 1979). Scheduling problems (there exist various versions of scheduling that consider different specification of the system) are NP-hard. For instance, NP-hardness of identical parallel machines scheduling problem with sequence-dependent setup times is studied where the objective is to minimize the maximum lateness of jobs has been proven (Mohabbati-Kalejahi and Yoon, 2015). The complexity of the constraints can also impact the computation time, as constraints that are more complex or involve more variables will (most of the time) require more computational power to be solved (Engin and Isler, 2022). The presence of integer variables in the model increases the computation time, as the feasible region of the IPs generally are not convex (Fischetti and Monaci, 2020). Branch-and-bound based solvers then must consider both the continuous and integer components of the problem, which makes the problem more difficult to solve (Zhou et. Al., 2022). Despite these challenges, researchers and practitioners continue to develop new methods and algorithms to reduce the required computation time and improve the efficiency of methods in solving the scheduling IP problems. Some of the most used solution methods include branch-and-bound algorithms, linear programming techniques, and heuristic methods (Belotti, 2013; Gharbi et al., 2019; Liu, 2023).

Required computational time, then, is a crucial factor in determining the efficiency of various solution processes for scheduling IP problems. With the advancement of computer technology, the computational time for solving large-scale scheduling problems has been significantly reduced (Vaisi et. Al., 2023). However, the question of whether these measurements of computational time are trustworthy and consistent, remains open. In recent years, research has been conducted to compare the computational time for job shop problems solved by different algorithms and on different computational setups. These studies aim to determine the relative performance of algorithms and the impact of computer hardware as well as the solution approach on the difficulty of finding the solution combined with determination of the computational times (Fowler and Monch, 2022; Yavary et al., 2018; Winston, 2022). Some researchers have also focused on the impact of the size and the complexity of the problems, known as "curse of dimensionality", suggesting new heuristic methods for optimization problems which otherwise need significantly long computational time (Toragay and Silva, 2021; Shirneshan et. Al., 2022; Soleimani et al., 2023). For instance, (Qaio et al., 2018) studied the computational time required by various algorithms for solving scheduling problems with time windows and found that there is a direct relation between required computational time and the size of the problem in hand. The study of Baker and Keller (2010) concluded that the choice of algorithm and availability of the computational resources are key factors in determining the computation time for scheduling problems. The trend in this type of recent studies is towards developing more efficient algorithms to reduce the computational time. However, although there have been many significant improvements toward reducing the required computational time, accurately estimating these times is still a challenging research topic, especially with the advancement in technology and the increasing computational power (Jen-Shiang, 2006; Grigoriev, 2007).

On the other hand, in recent years, notable improvements have been observed in the development of modern heuristic approaches towards solving these problems almost completely (zero percent optimality gap) as well as new approximation methods that tend to reduce the consumption power (Gharibi, 2021; Ahmadinejad et. Al., 2020). Although recently, solving different types of IP as well as any other optimization approaches combined with simulations such as Monte-Carlo simulation have found many new applications in diverse research areas such as construction smart buildings (Fatehi, 2023), social networks (Shahparvari, 2022) or even prompt usages such as COVID-19 pandemic (Kazemzadeh, 2023); still a significant amount of effort are directed towards solving the classic glitches. Even in some cases a problem as big as 400 jobs and 20 machines has been successfully solved using heuristic approaches (Alba et al, 2022).

The research area that has not received attention in this field is the measurement of relative computational time in solving IPs based on different optimality gaps. Aiming to highlight the mentioned gap in the research in this field, this work involves comparing the computational time required to solve a job shop problem with a 0% gap (to optimality) with the time required to solve the same problem with a given gap, for instance 10%.

3. Methodology

3.1. Monte Carlo simulation

A Monte Carlo simulation is a computational technique that uses random sampling and probability distributions to model and analyze complex systems or processes. In a Monte Carlo simulation, a problem or scenario is simulated multiple times using random input variables. Each simulation, also known as a trial or iteration, involves sampling values from probability distributions associated with the variables in the model. By running a large number of trials, the simulation can generate a range of possible outcomes and their associated probabilities (Mooney, 1997). In this study as a Monte Carlo Simulation a number of 1200 of instances of the problem have been created and solved to have enough significance for the results. Although this number could have been higher, for the sake of time limitation, the recorded times in 1200 runs is considered as the main source of data analysis. Of course, for future studies, more data points can be generated by using the created code (which is publicly available) to randomly generate and solve the problem instances.

3.2. Mathematical model

While there are many different versions of the scheduling problem and particularly the job shop scheduling problem, one of the main models that have been worked on in the scheduling literature, is the type of job shop problem that is addressed in this study with certain number of machines, jobs, and a schedule for each job that contains various working hours on different machines (Waller 1973). The mathematical model of this problem and our approach in solving it are illustrated below:

Assume that we have *m* machines to work on *n* different jobs. For each job the machine order is fixed. Without loose of generality, we assumed that for each job we have *m* tasks to be done on different machines. For the mathematical model of this system, the goal is to determine the work schedule for the machines, such that the sum of completion times is minimized. The objective function is defined as follow:

$$\min \sum_{j=1}^{n} t_{a_{m,j}j} \tag{1}$$

Where $a_{m,j}$ is the m^{th} machine that will work on job *j* (Integer Parameter). t_{ij} is defined as the times when machine *i* finishes working on job *j* (Integer Variable). $t_{a_{m,j}j}$ is the time when the m^{th} machine is finished working on job *j* (Integer Variable). Also, we define variables s_{ij} and p_{ij} as the start time and processing time of machine *i* working on job *j (both Integer Parameters)*. And finally:

$$y_{ijk} = \begin{cases} 1 & \text{if job } j \text{ is done before job } k \\ 0 & \text{if job } j \text{ is done after job } k \end{cases}$$

Other assumptions of the system are that a machine can only process one job at a time, each job must be processed to completion, i.e., once started on a machine it cannot be interrupted, and each job can be processed in one machine at a time. The constrains of this problem are listed below:

$$s_{a_k j j} \geq t_{a_{k-1,j} j} \qquad \forall j \in \{1,2,\dots,n\} \;\; and \;\; k \in \{2,3,\dots,m\} \tag{2}$$

$$t_{ij} = s_{ij} + p_{ij} \qquad \forall\, i,j \in \{1,2,\dots,n\} \tag{3}$$

$$s_{ik} \geq t_{ij} - y_{ijk} * M \qquad \forall\, i,j,k \in \{1,2,\dots,n\} \tag{4}$$

$$s_{ij} \geq t_{ik} - \left(1 - y_{ijk}\right) * M \qquad \forall\, i,j,k \in \{1,2,\dots,n\} \tag{5}$$

$$s_{ij} \geq 0, t_{ij} \geq 0 \qquad \forall\, i,j \in \{1,2,\dots,n\} \tag{6}$$

$$s_{ij}, t_{ij}, p_{ij} \in \mathbb{Z}^n \qquad y_{ijk} \in \{0,1\} \tag{7}$$

This big M or disjunctive approach has been developed and reviewed several times in the literature and still is considered as one of the main solutions to job shop problems. (Garey et al., 1976; Taillard, 1993; Kis, 2002).

3.3. Coding in python

All the codes that are explained here are publicly available at https://github.com/oguztoragay/SchedulingGAP.git address. The coding and implementation of all the problems have been done in Python. We specifically explain the packages that we used for the coding stage, in particular the main Algebraic Modeling Language (AML) that we applied. One of the most important aspects that needs to be considered even before starting to code the mathematical models, is to decide about the algebraic modeling language to be used in the coding step of optimization problem. In this regard we first code the model in Pyomo (AML package for python) (Bynum et al., 2021) but early attempts to solve the models revealed that using Pyomo, it is not straight forward to collect necessary data about the times that the solvers reach the predefined optimality gap during the solving process. Gurobi solver (Gurobi Optimization, 2023) is one of the commercial solvers which has its own library to be used in Python (this library is called gurobipy and can be installed via python package managers).

Using the callback functions defined in gurobipy library, we could extract the required solving times to be later used in the statistical analysis. Moreover, some complementary packages such as NumPy, pandas, and strings were also used to randomly generate the problem instances, preprocessing of the gathered data from the solutions, and store the results. An "instance generator" function is defined to get as input three numbers, i.e., number of machines, number of jobs, and number of instances to be generated. On each iteration this function generates a random instance from the given size, solves it, and stores the results. In this function jobs are identified as a range of integers (job 1, job 2, etc.), and the machines are identified as a range of letters (machine A, Machine B, etc.). After creating a shuffled list of machines, an identical list of orders of machines is created and with defining the needed sets the random orders for each job could be simulated. The instance generation is summarized in the following pseudocode.

Pseudocode for Instance generator

```
Get the number of machines as $m$ and number of jobs as $n$:
For all jobs:
    For tasks in job n:
        Shuffle list of machines,
        Generate machine – task combinations,
        Assign random processing time $t$ to the combination,
        Save all in lists
    Return the lists
Generate an optimization problem instance using the lists
Solve the instance
Save solutions and collected data
```

Note that, the processing time, t, that is randomly assigned to the machine-task combinations is an integer number in [1, 10]. As an example of the pseudocode output, if we feed the function with m = 3 and n = 4, we will have:

Work order: {(1,0):E, (2,0):B, (3,0):L, (1,1):L, (2,1):E, (3,1):B, (1,2):E, (2,2):B, (3,2):L, (1,3):L, (2,3):B, (3,3):E}

Process time: {(B,0):9, (E,0):3, (L,0):7, (B,1):1, (E,1):3, (L,1):5, (B,2):3, (E,2):3, (L,2):8, (B,3):9, (E,3):5, (L,3):6}

In this example, the machines are named: E, B, and L while the jobs are 0, 1, 2, 3. For instance, for job 1, the order of tasks are using machines: L, E, B respectively and the process (working) time for each of them are 5, 3, 1 respectively. After the data is generated (creation of one random instance of the scheduling problem), the data is reshaped from dictionary and formed as an IP to be solved by Gurobi solver.

3.4. Solving

As stated before, the Gurobi solver is used in this work. To keep track of the times needed for different gaps, the callback functions are used which allows the solver to continue the solving process without interruption while collecting the required data. Accordingly, the setting of the solver would be to solve the problem to 20% gap and report the consumed time for that solution, continue solving the problem for 10% gap and report the consumed time for solution and do the same process for 5% and 0% (optimum solution) again. For each instance of the problem, four different times are reported that represent the needed time to solve that instance of the problem for 20%, 10%, 5% and 0% gaps.

The main reason for choosing these four gaps is to be as close as possible to the real-life applications of solving this type of problem. It is assumed that, when a similar problem needs to be solved in a real organization, the gaps of more than 20% are not viable to be implemented due to being far from optimal solution. In addition to the 20% gap, 10% and 5% were chosen due to their simplicity and the sole assumption that in industry, there is negligible difference between 80% and 85% accuracy while 90% generates more accurate feeling in the user.

It is considered that if a user tries to run the scheduling problem in real life and allocate jobs to machines, a better knowledge about the anticipated times to reach these "easy to understand" gaps could create a huge benefit. Data storage is the last step in the code that records the four solution times related to reaching 20%, 10%, 5%, and 0% gap into a list. This list of various instances then stored in a file for later analysis.

The computer setup that we run the experiments has the following configuration: a desktop computer working on a Microsoft Windows operating system with Core i9 processor and 64 GB of ram on which Gurobi 10.0.0 has been installed. This system was not used for any other computational application while working on the designed experiments for this work.

4. Results and Analysis

In this section we first provide the results for all the instances that we solved for different combinations of machine and job numbers. Then we analysis different aspects of the solutions and provide the observations that we found interesting.

4.1. Design of experiment

To determine the problem sizes, we considered two aspects, on one hand, if the problem is small-scale (being solved in less than a fraction of a second) then the main consumed time would have been related to the computer's internal calculations and problem definition rather than the consumed time for the branch and bound tree generation and in this case the recorded times could have had high bias. On the other hand, if the problem is large-scale, the running time can exceed weeks of calculation time.

Accordingly, the selection of the number of machines and jobs are for specific solution times that are not very small (less than a second) and not very large (more than 5 hours) which leaded to number of machines between 5 to 8 and number of jobs between 8 to 10). The largest problem in this study was with 8 Machines and 10 Jobs which is not considered a large problem at the moment with new methods and solutions and computational powers (Alba et al, 2022). Moreover, there are other numbers that could generate calculation times in the desired range which also could be addressed in the future studies. Table 1 shows the summary of all experiments for all the selected problem sizes.

Table 1. Summary of all instances in the designed experiment

Experiment	*Number of Machines*	*Number of Jobs*	*Number of Instances*	*Average calculation time (sec)*	*Total calculation time (min)*
1	5	8	100	2.53	4
2	5	9	100	49.34	82
3	5	10	100	5368.16	8947
4	6	8	100	2.31	4
5	6	9	100	43.90	73
6	6	10	100	5585.48	9309
7	7	8	100	2.61	4
8	7	9	100	73.12	122
9	7	10	100	3454.67	5758
10	8	8	100	5.54	9
11	8	9	100	73.70	123
12	8	10	100	2657.05	4428
Total	-	-	1200	17318.4	28864

We solved in total 1200 instances from 12 different problem sizes, which took around 500 hours of calculation time. The main reason for limiting the number of instances to 100 in each problem size is basically the limitations in the total available calculation time. One immediate interesting observation is that, although we expected the calculation time for the problems with 8 machines to be significantly higher than those with 5 machines, the calculation time is decreased when we increase the number of machines. While explaining the reason behind this observation is beyond the scope of this study, the advancements in applying sophisticated heuristic approaches that the commercial solvers equipped with to solve this type of combinatorial problems, is one of the effective reasons behind this observation.

4.2. Distribution of the solving times

One of the biggest targets of this study is to find a trend in the calculation time while solving the IP problems specifically scheduling IP problems. Since the value of solving time in this study cannot be understood solely, in most of the graphs a relative time is used. We define the following two concepts to illustrate the relative time.

Relative time for a gap: For each instance, defined as the time to reach specific gap divided by the total calculation time (time to reach 0% gap) to solve that instance. For example, if the time to reach 10% gap (i.e., solving the problem with 90% accuracy) in a problem instance is 60 seconds and the total solution time (i.e., finding the optimal solution) is 120 seconds, then for that instance, the relative time for 10% gap is 50%.

Considering the number of instances in the experiment and the variety of solution times (from seconds to hours), it cannot be easy to distinguish the meaning of time as a scalar value, without comparing the solution times to each other, i.e., there will not be meaningful comparison of the long or short solution times. Accordingly, for this study, a relative measure of time is considered for a better understanding of the results.

Relative percentage of total solution time: For each instance this value calculated as the total solution time (reaching optimality gap of 0%) for that instance divided to the largest solution time in all 100 instances from the same problem size. For example, if the total solution time for an instance is 250 seconds and the longest recorded solution time for all instances in that problem size is 1000 seconds, then the relative percentage of total solution time for that instance is 25%. The distributions of the relative time for 20%, 10%, 5% and 0% gaps are given in Figure 1. This distribution is the frequency histogram for each gap for all the 1200 instances. The distribution of solution times in 20% gaps is flatter than the one for 0% gap where it becomes more logarithmic. In the 20% gap, less than 50% of the problems were solved in 0 to 10% of their relative time (to the biggest 20% gap solution time for their similar instance of the problem) while this percent for 0% gaps goes up to more than 80%. Considering the differences on relative times for each gap, as the computer tries to solve the problem, the beginning of solutions seems to have a more even distribution. This issue is clearer in the below graph and combines all four data together.

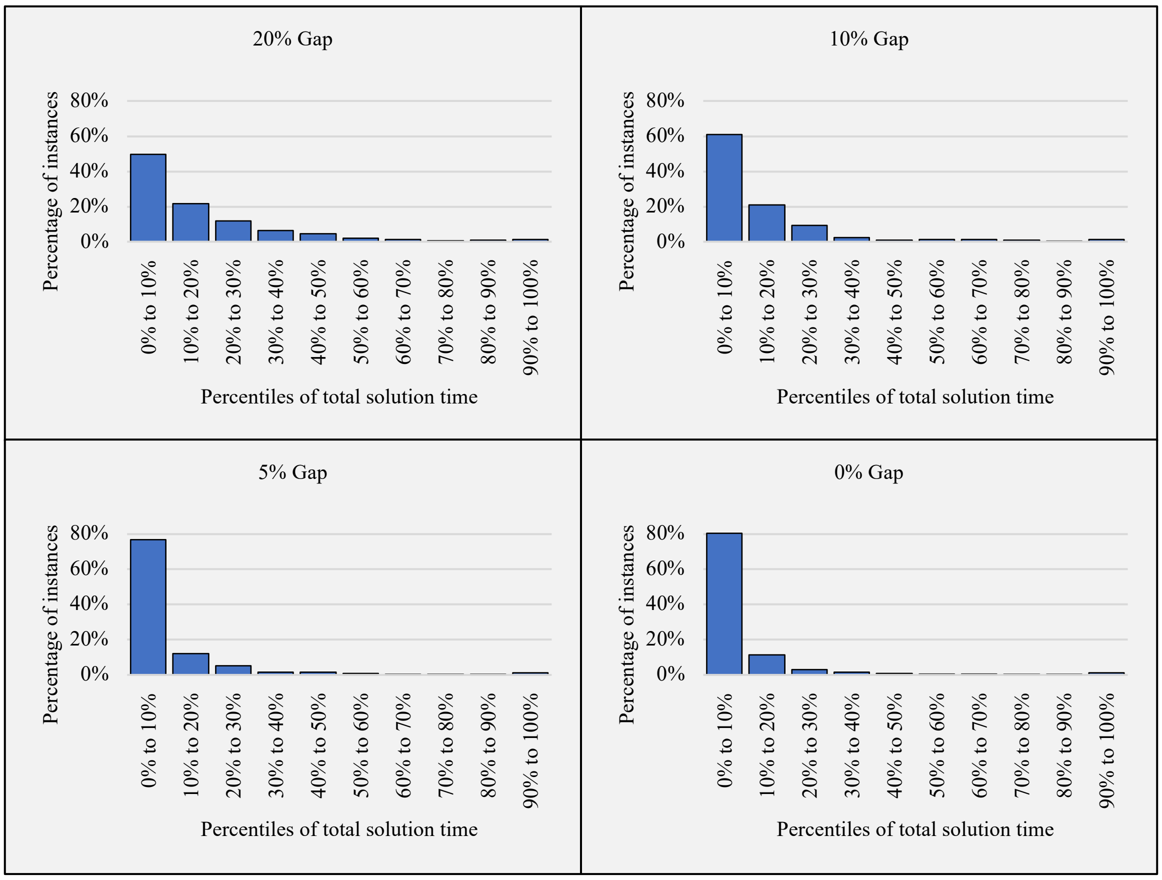

Figure 1. Solution time distribution for various optimality gaps

Moreover, Figure 2 shows that in all four different gaps, most of the time, the solution time is a fraction of time the solver needs to spend on solving the problem. This issue will be addressed in more detail in the following sections.

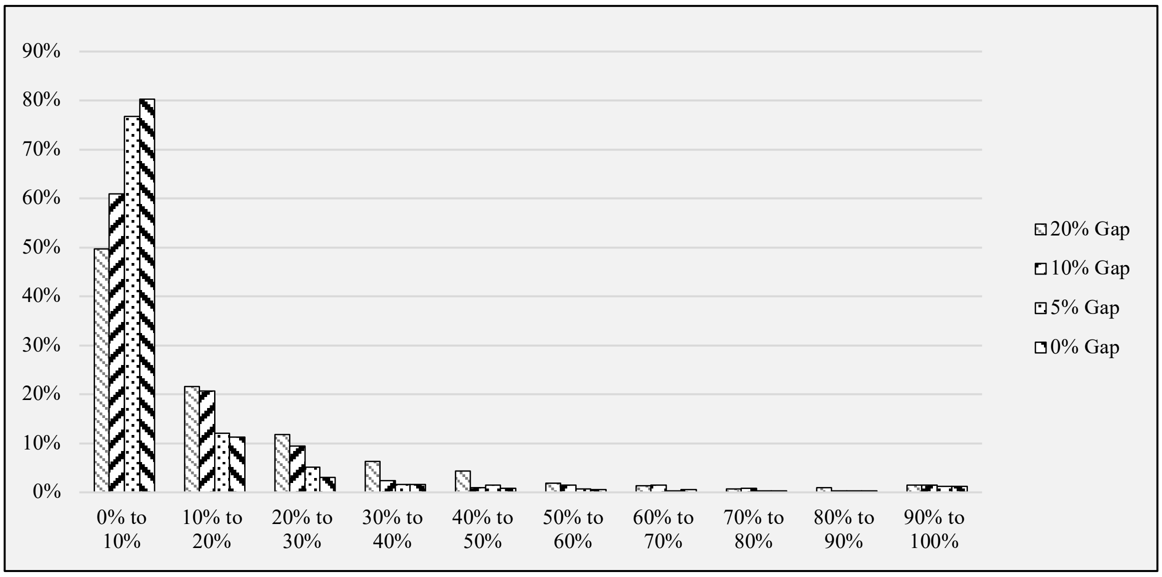

Figure 2. The distribution of the relative times.

4.3. The relation between size of the problem and different gaps

As reported in Table 1, the average solution times for different instances of the scheduling problem varies between 2.31 to 5585.5 seconds based on the number of machines and jobs. This simply illustrates that the solver requires more solving time to solve a larger problem instance in terms of parameters (number of machines and number of jobs), which is an expected observation.

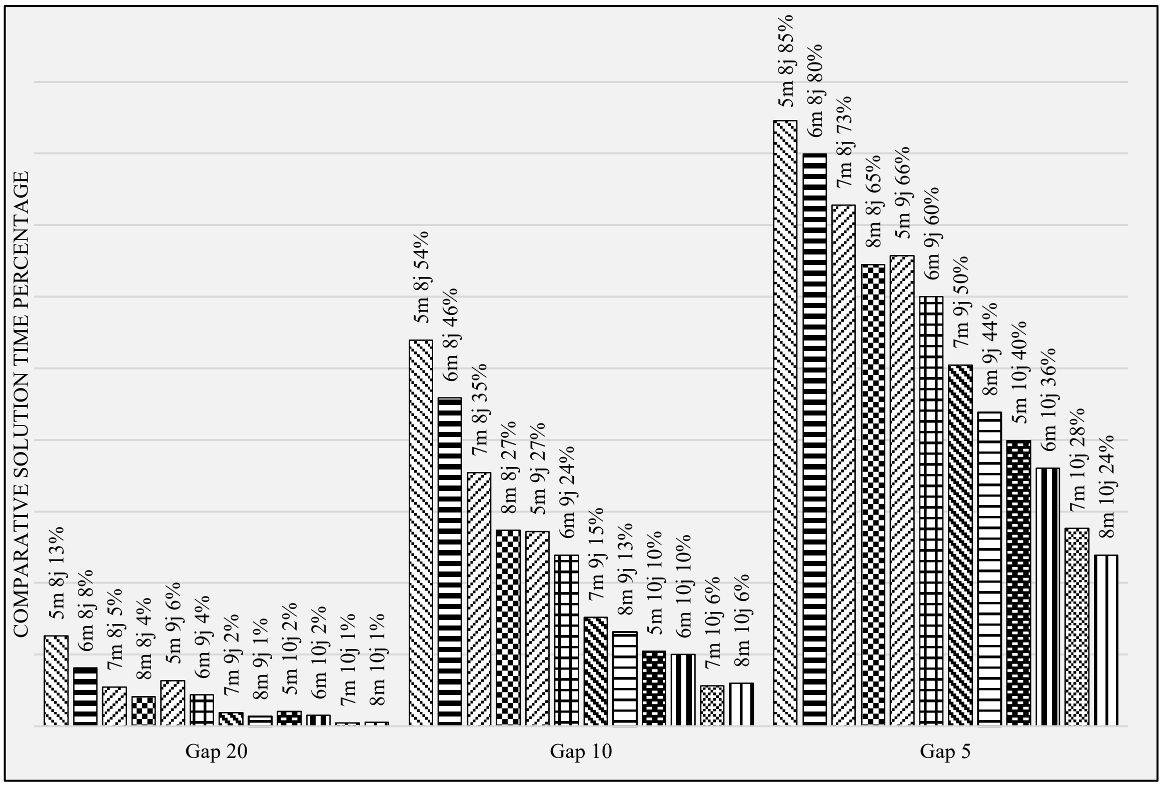

Figure 3. Percentage of Total solution time based on each instance of problem.

On the other hand, as shown in Figure 3, the size of the problem also affects the ratio of time to reach different gaps. For example, it takes on average 85% of total solution time to reach the 5% gap when solving an instance with 5 machines and 8 jobs while the same rate is 24% when solving an instance with 8 machines and 10 jobs. This trend can be observed for every recorded gap in this study while the criteria to order the instances in this graph is the complication of the problem and not the average calculation time. Table 2 shows the details for all the problem sizes in the designed experiment. It can be inferred that the number of jobs has a greater impact than the number of machines on the solving time in this type of scheduling problem.

Table 2. Summary of relative times for gaps in all problem sizes (ascending order based on calculation time)

	Machines	*Jobs*	*Total calculation time (min)*	*Gap 20*	*Gap 10*	*Gap 5*
▤	6	8	4	8%	46%	80%
▨	7	8	4	5%	35%	73%
▧	5	8	4	13%	54%	85%
▩	8	8	9	4%	27%	65%
▦	6	9	73	4%	24%	60%
▨	5	9	82	6%	27%	66%
▤	8	9	123	1%	13%	44%
▧	7	9	122	2%	15%	50%
▥	8	10	4428	1%	6%	24%
▩	7	10	5758	1%	6%	28%
▩	5	10	8947	2%	10%	40%
▥	6	10	9309	2%	10%	36%

An interesting trend can be observed when comparing Table 2 data to Figure 3, which suggests the time-gap rates also follow the complication of the problem rather than just the solution time. It is worth mentioning that this order assumes that a more complicated problem takes more solving time. The general expectation is that, as the number of machines increases, the problem gets more complicated; and consequently, more calculation time should be expected. Whereas based on Table 2 this correlation does not exist.

The odd behavior behind the lack of correlation between the number of machines and average solution time could be interpreted because of the limited number of instances for each size of the main problem (100 instances). Although this could be the only logical explanation behind this behavior, the number of instances were high enough to see other trends. Considering the central limit theorem, and the total number of experiments, these complications should be addressed in future studies.

4.4. Probabilistic view on the solution times

Table 3 lists the distribution of Relative percentage of total solution time for all the runs combined. Considering the total number of 1200 instances of the scheduling problem, approximately 95% of the total instances have been solved in a time less than 30 percent of the longest solving time among all the instances in that size category. Additionally, more than 80% of the cases have Relative percentage of total solution time less than 10%. These rates are illustrated in Figure 4.

Table 3. Distribution of Relative solution times

Relative percentage of total solution time	*0% to 10%*	*10% to 20%*	*20% to 30%*	*30% to 40%*	*40% to 50%*	*50% to 60%*	*60% to 70%*	*70% to 80%*	*80% to 90%*	*90% to 100*
Number of cases in that segment	964	135	36	19	10	7	6	4	4	15
Percentage of total runs	80	11	3	2	1	1	1	0	0	1
Cumulative percentage	80.3	91.6	94.6	96.2	97.0	97.6	98.1	98.4	98.8	100

Based on the above observation, there is a slight chance (5%) for an instance to be solved in a significantly long time while more than 80% of the instances need a fraction of that long solving times (less than 10%) to be solved.

5. Conclusion and Future Work

In this work, we designed an experiment to study the correlation between problem size on one hand and the required solving time on the other hand, for the specific scheduling problems that is mathematically modeled as IP problems using Big M method. Solving 1200 randomly generated various instances of a problem in around 500 hours, we conclude that most of the optimization problems could have an acceptable part of the result in a small fraction of the total solution time. Also, this fraction gets smaller when the problem gets larger. Accordingly, organizations and industries could benefit from this phenomenon to save an enormous amount of time and resources just with allocating jobs with a small proximity until the results of the optimization is finalized. If a company uses a time schedule which is in 10% gap to the final schedule (and since we know the allocation between 10% gap and 0% gap could be identical in Integer Programming) as a primary solution and gives the tasks based on that; the company would save days of waiting annually.

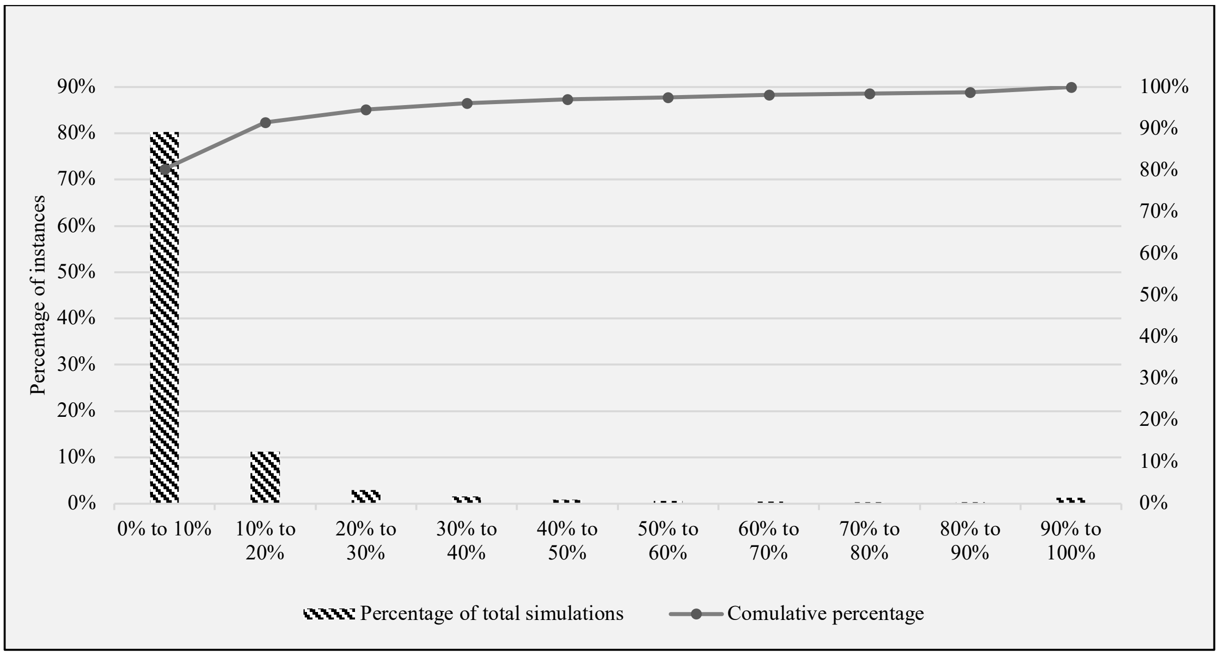

Figure 4. Distribution and Cumulative frequency of Relative solution times

Correspondingly, we concluded that the number of jobs in the problem is a more effective factor than the number of machines when we study the length of the required time to solve the problem. In addition, we did not observe the expected correlation between the required solving time on the one hand and the problem's complicatedness on the other hand. We noted that this odd behavior can be related to the complex process that commercial solvers have behind the scenes. Looking at the effect of the solvers' heuristic steps in solving the problem is an area that needs further study.

The effect of applying heuristics, generated various cuts, branch-and-bound node selection criteria and many other parameters of Gurobi on the required solving time for scheduling problems are planned to be studied in future works of the authors. We aim to analyze the required solution time for this type of scheduling problem in order to help schedulers make better business decisions. Moreover, this study acknowledges that its findings and conclusions are not universally generalizable to all types of job shop scheduling problems, highlighting the imperative need for future academic research to address the unexplored dimensions and complexities specific to those particular problem variants.

Conflict of interest

The authors declare that they have no conflict of interest.

Contribution of authors

The authors have equal contribution in all the writing, modeling, and experimental design.

References:

Ahmadinejad, M., Taheri, N., & Moaiyeri, M. H. (2020). Energy-efficient magnetic approximate full adder with spin-Hall assistance for signal processing applications. Analog Integrated Circuits and Signal Processing, 102, 645-657. http://doi.org/10.1007/s10470-020-01630-z

Baker, K. R., & Keller, B. (2010). Solving the single-machine sequencing problem using integer programming. Computers & Industrial Engineering, 59(4), 730-735. http://doi.org/10.1016/j.cie.2010.07.028

Belotti, P., Kirches, C., Leyffer, S., Linderoth, J., Luedtke, J., & Mahajan, A. (2013). Mixed-integer nonlinear optimization. Acta Numerica, 22, 1-131. http://doi.org/10.1017/S0962492913000032

Bynum, M. L., Hackebeil, G. A., Hart, W. E., Laird, C. D., Nicholson, B. L., Siirola, J. D., ... & Woodruff, D. L. (2021). Pyomo-optimization modeling in python (Vol. 67). Berlin/Heidelberg, Germany: Springer. http://doi.org/10.1007/978-3-030-68928-5

Chankong, V., & Haimes, Y. Y. (2008). Multiobjective decision making: theory and methodology. Courier Dover Publications. ISBN 9780486462899

Chen, J. S. (2006). Using integer programming to solve the machine scheduling problem with a flexible maintenance activity. Journal of Statistics and Management Systems, 9(1), 87-104. http://doi.org/10.1080/09720510.2006.10701195

Engin, O., & İşler, M. (2022). An efficient parallel greedy algorithm for fuzzy hybrid flow shop scheduling with setup time and lot size: a case study in apparel process. Journal of fuzzy extension and applications, 3(3), 249-262. https://doi.org/10.22105/jfea.2021.314312.1169

Fatehi, N., Politis, A., Lin, L., Stobby, M., & Nazari, M. H. (2023, January). Machine Learning based Occupant Behavior Prediction in Smart Building to Improve Energy Efficiency. In 2023 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT) (pp. 1-5). IEEE. http://doi.org/10.1109/ISGT51731.2023.10066411

Fischetti, M., & Monaci, M. (2020). A branch-and-cut algorithm for mixed-integer bilinear programming. European Journal of Operational Research, 282(2), 506-514. http://doi.org/10.1016/j.ejor.2019.09.043

Fowler, J. W., & Mönch, L. (2022). A survey of scheduling with parallel batch (p-batch) processing. European journal of operational research, 298(1), 1-24. http://doi.org/10.1016/j.ejor.2021.06.012

Garey, M. R., Johnson, D. S., & Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. Mathematics of operations research, 1(2), 117-129. http://doi.org/10.1287/moor.1.2.117

Garey, M. R., & Johnson, D. S. (1979). Computers and intractability (Vol. 174). San Francisco: freeman. ISBN: 9780716710448

Gharbi, A., Mrad, M., Chalghoumi, S., & Ladhari, T., (2019). Enhanced lower bounds and exact procedures for total completion time minimization in a two-machine permutation flowshop with release dates. International Transactions in Operational Research, 26(6), 2432-2449. https://doi.org/10.1111/itor.12421

Gharibi, K., & Abdollahzadeh, S. (2021). A mixed-integer linear programming approach for circular economy-led closed-loop supply chains in green reverse logistics network design under uncertainty. Journal of Enterprise Information Management, (ahead-of-print). http://doi.org/10.1108/JEIM-11-2020-0472

Grigoriev, A., Sviridenko, M., & Uetz, M. (2007). Machine scheduling with resource dependent processing times. Mathematical programming, 110, 209-228. http://doi.org/10.1007/s10107-006-0059-3

Gurobi Optimization, LLC. (2023). Gurobi Optimizer Reference Manual. Retrieved from https://www.gurobi.com/wp-content/plugins/hd_documentations/documentation/10.0/refman.pdf

Héctor, G., Nucamendi-Guillén, S., & Avalos-Rosales, O. (2022). A mixed integer formulation and an efficient metaheuristic for the unrelated parallel machine scheduling problem: total tardiness minimization. EURO Journal on Computational Optimization, 10, 100034. https://doi.org/10.1016/j.ejco.2022.100034

Kazemzadeh, F., Safaei, A. A., Mirzarezaee, M., Afsharian, S., & Kosarirad, H. (2023). Determination of influential nodes based on the communities' structure to maximize influence in social networks. Neurocomputing, 534, 18-28. http://doi.org/10.1016/j.neucom.2023.02.059

Kis, T. (2002). On the complexity of non-preemptive shop scheduling with two jobs. Computing, 69, 37-49. http://doi.org/10.1007/s00607-002-1455-z

Liu, Q., Kosarirad, H., Meisami, S., Alnowibet, K. A., & Hoshyar, A. N. (2023). An Optimal Scheduling Method in IoT-Fog-Cloud Network Using Combination of Aquila Optimizer and African Vultures Optimization. Processes, 11(4), 1162. http://doi.org/10.3390/pr11041162

Linderoth, J., & Wright, S. J. (2005). Computational grids for stochastic programming. In Applications of stochastic programming (pp. 61-77). Society for Industrial and Applied Mathematics. http://doi.org/10.1137/1.9780898718799.ch5

Mohabbati-Kalejahi, N., & Yoon, S. W. (2015). Parallel machines scheduling problem for minimization of maximum lateness with sequence-dependent setup times. In IIE Annual Conference. Proceedings (p. 837). Institute of Industrial and Systems Engineers (IISE). Retrieved from https://www.proquest.com/scholarly-journals/parallel-machines-scheduling-problem-minimization/docview/1791989138/se-2

Mokhtari, A., Bagheri, H., Ghazvini, M., & Ghader, S. (2022). New mathematical modeling of temperature-based properties of ionic liquids mixture: Comparison between semi-empirical equation and equation of state. Chemical Engineering Research and Design, 177, 331-353. http://doi.org/10.1016/j.cherd.2021.10.039

Mooney, C. Z. (1997). Monte carlo simulation (No. 116). Sage. http://doi.org/10.4135/9781412985116

Pferschy, U., & Staněk, R. (2017). Generating subtour elimination constraints for the TSP from pure integer solutions. Central European journal of operations research, 25, 231-260. http://doi.org/10.1007/s10100-016-0437-8

Pourghorban, A., Dorothy, M., Shishika, D., Von Moll, A., & Maity, D. (2022, December). Target Defense against Sequentially Arriving Intruders. In 2022 IEEE 61st Conference on Decision and Control (CDC) (pp. 6594-6601). http://doi.org/10.1109/CDC51059.2022.9992425

Qiao, L., Zhang, Z., & Huang, Z. (2021). A scheduling algorithm for multi-workshop production based on BOM and process route. Applied Sciences, 11(11), 5078. http://doi.org/10.3390/app11115078

Sattarkhan, M. H. (2023). Production Scheduling of Parallel Identical Lines in a Multi-Product Manufacturing System with Genetic Algorithm. International Journal of Research in Industrial Engineering, 12(1), 7-20.

Shahparvari, S., Hassanizadeh, B., Mohammadi, A., Kiani, B., Lau, K. H., Chhetri, P., & Abbasi, B. (2022). A decision support system for prioritised COVID-19 two-dosage vaccination allocation and distribution. Transportation Research Part E: Logistics and Transportation Review, 159, 102598. http://doi.org/10.1016/j.tre.2021.102598

Shirneshan, H., Sadegheih, A., Hosseini-Nasab, H., & Lotfi, M. M. (2022). A two-stage stochastic programming approach for care providers shift scheduling problems. Journal of Applied Research on Industrial Engineering. https://doi.org/10.22105/jarie.2022.349970.1488

Soleimani, M., Mahmudi, F., & Naderi, M. H. (2023). Some results on the maximal graph of commutative rings. Advanced Studies: Euro-Tbilisi Mathematical Journal, 16(supp1), 21-26. http://doi.org/10.32513/asetmj/1932200823104

Taha, H. A. (2013). Operations Research: An Introduction. Pearson Education. ISBN 9789332518223

Taillard, E. (1993). Benchmarks for basic scheduling problems. European journal of operational research, 64(2), 278-285. http://doi.org/10.1016/0377-2217(93)90182-M

Toragay, O., & Silva, D. F. (2021). Fast heuristic approach for control of complex authentication systems. Applied Stochastic Models in Business and Industry, 37(4), 744-766. http://doi.org/10.1002/asmb.2619

Vaisi, B., Farughi, H., Raissi, S., & Sadeghi, H. (2023). A bi-objective optimal task scheduling model for two-machine robotic-cell subject to probable machine failures. Journal of applied research on industrial engineering, 10(1), 141-154. https://doi.org/10.22105/jarie.2022.336768.1465

Waller, L. (1973). A Branch-bound-and-imply Algorithm for an Improved Attack Upon the Job-shop Scheduling Problem. Computing Laboratory Technical Report Series. Retrieved from https://eprints.ncl.ac.uk/file_store/production/160014/DC7B64C6-C876-4D3D-8DAC-270375AEFADE.pdf

Winston, W. L. (2022). Operations research: applications and algorithms. Cengage Learning. ISBN: 9780357907818

Yavary, A., & Sajedi, H. (2018, June). Solving dynamic vehicle routing problem with pickup and delivery by CLARITY method. In 2018 IEEE 22nd International Conference on Intelligent Engineering Systems (INES) (pp. 000207-000212). IEEE. http://doi.org/10.1109/INES.2018.8523908

Zhou, T., El-Wahed Khalifa, H. A., Najafi, S. E., & Edalatpanah, S. A. (2022). Minimizing the Machine Processing Time in a Flow Shop Scheduling Problem under Piecewise Quadratic Fuzzy Numbers. Discrete Dynamics in Nature and Society. http://doi.org/10.1155/2022/3990534