VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Doan Thanh Son

SOLVING JOB SHOP SCHEDULING PROBLEM – JSSP BY SAT ENCODING

Major: Computer Science

HA NOI - 2024

VIETNAM NATIONAL UNIVERSITY, HANOI
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

Doan Thanh Son

SOLVING JOB SHOP SCHEDULING PROBLEM – JSSP BY SAT ENCODING

Major: Computer Science

Supervisor: PhD. To Van Khanh

HA NOI - 2024

Acknowledgements

I would like to express my gratitude to University of Engineering and Technology for providing me with the opportunity to do this research. I am also thankful to PhD. To Van Khanh for his dedicated support, encouragement, and supervision.

Finally, a deep thank to family, relatives and friends - who are always with me during the most difficult times, always encouraging and encouraging me in life and at work. This accomplishment would not have been possible without them.

In the report, there may be some unintentional errors. I sincerely hope to receive valuable feedback and suggestions from lecturers, colleagues, and supervisors to improve it further. Sincerely thank.

Declaration

I declare that the report has been composed by myselves and that the work has not be submitted for any other degree or professional qualification. I confirm that the work submitted is my own, except where work which has formed part of jointly-authored publications has been included. My contribution to this work have been explicitly indicated below. I confirm that appropriate credit has been given within this report where reference has been made to the work of others.

I certify that, to the best of my knowledge, my report does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my report, published or otherwise, are fully acknowledged in accordance with the standard referencing practices.

I take full responsibility and take all prescribed disciplinary actions for my commitments.

Signature

Doan Thanh Son

Abstract

Abstract: One common scheduling problem is the job shop, in which multiple jobs are processed on several machines. Each job consists of a sequence of tasks, which must be performed in a given order, and each task must be processed on a specific machine. Each machine can only perform one task at a time. This type of scheduling problem has numerous applications in manufacturing, such as in automobile and aircraft production, where different parts need to be processed separately. Finding an optimal schedule helps optimize production time without causing congestion. This research tries to solve open Job-Shop Scheduling Problems (JSSPs) by translating them into Boolean Satisfiability Problem (SAT). The encoding method is based on the one proposed by Crawford and Baker. I also propose a method for estimate the lower bound, upper bound, and a finding the optimal solution alogrithm based on the binary search algorithm. Two problems La03 and Orb07 are solved based on these methods.

Keywords*: scheduling, SAT encoding, job-shop scheduling, makespan*

Table of Contents

Acronyms

CNF	Conjunctive normal form
JSSP	Job-shop scheduling problem
LB	lower bound
SAT	Boolean statisfiability
UB	upper bound

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Motivation

Scheduling involves the intricate task of determining the optimal processing path, timing, assigned machine, and operation for each machining object. The Job Shop Scheduling Problem (JSSP) is a typical scheduling problem, which is one of production scheduling system problems. It has been extensively applied in many different sectors especially big data. Hashem and his team [6] emphasized the significance of scheduling in big data processing and specifically optimized MapReduce job scheduling - a key framework for distributed and scalable data processing. This optimization, based on completion time and cost in cloud service models, addresses the critical task of determining optimal processing paths, timings, assigned machines, and operations for machining objects. In the context of big data processing, scheduling algorithms are pivotal for orchestrating complex workflows, ensuring effective resource utilization, and minimizing completion time. These algorithms facilitate the distribution of computing tasks across various resources, be it clustered nodes or virtual machines in cloud environments. Significantly, they play a vital role in cost reduction within cloud computing, strategically allocating resources based on factors like data transfer costs and computing resource pricing. Addressing scalability concerns, scheduling strategies enable seamless system scaling by efficiently distributing tasks across expanding nodes or virtual machines. In shared cloud environments, scheduling is indispensable for meeting Service Level Agreements (SLAs), ensuring timely completion of critical tasks. Real-time data processing requirements are met through the prioritization of time-sensitive tasks. Scheduling algorithms also consider heterogeneous environments, optimizing task assignments for enhanced

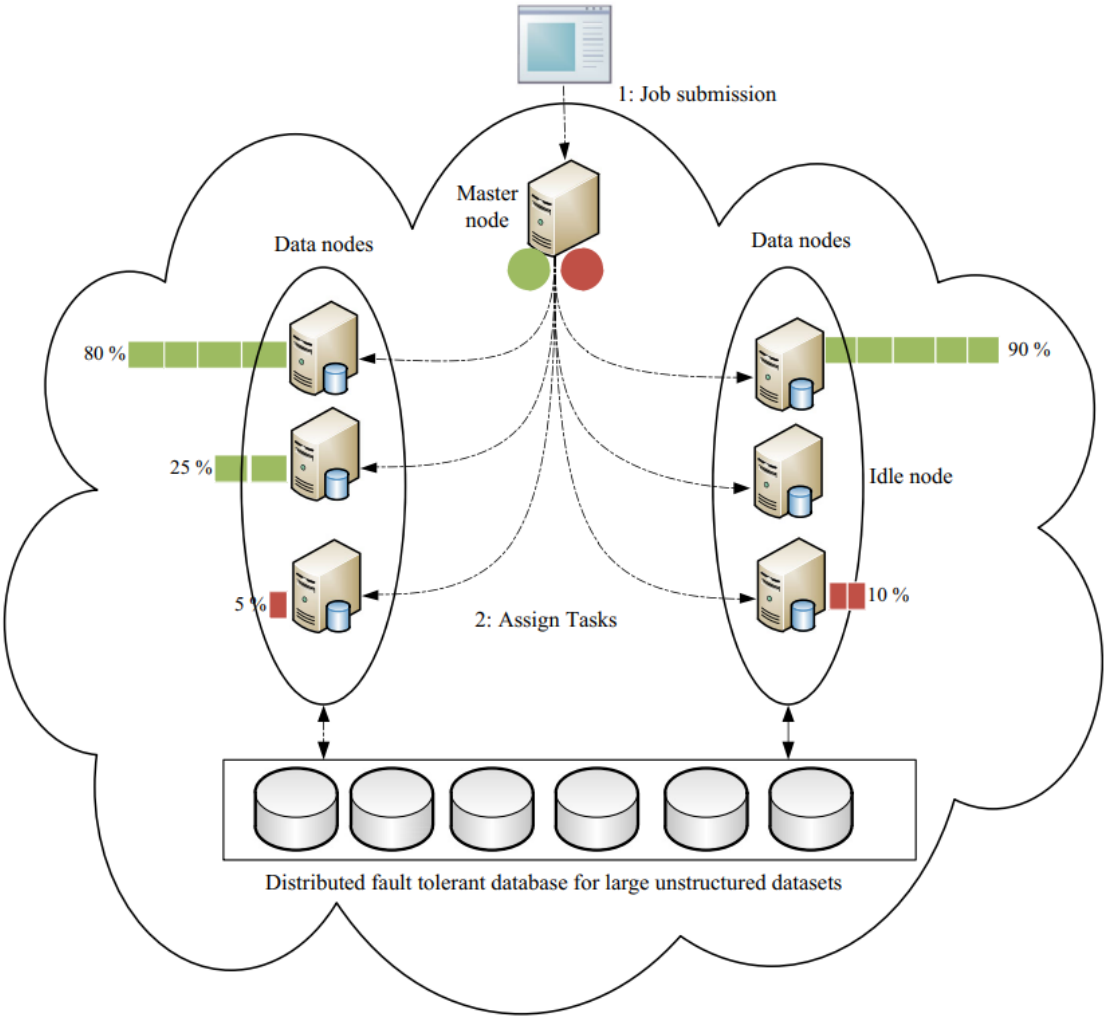

Figure 1.1: Big data scheduling process using the Hadoop MapReduce system

performance in big data clusters and cloud infrastructures. Fault tolerance and reliability are strengthened through intelligent task redistribution in the face of failures or delays. Notably, scheduling algorithms contribute to energy efficiency by consolidating tasks and considering the characteristics of different resources. Lastly, the integration of predictive analytics into advanced scheduling models anticipates resource demands, enabling proactive adjustments to prevent bottlenecks and ensure smooth data processing workflows. In summary, scheduling in big data environments is integral, addressing challenges related to efficiency, cost optimization, scalability, reliability, and the dynamic nature of modern computing infrastructures. The scenario illustrated in Fig 1.1 presents the big data scheduling process using Hadoop MapReduce in a cloud computing environment.

1.2 Related work

The JSSP is an NP-hard problem that has been studied by numerous researchers. NP-hard means in the worst case, all known algorithms for solving such problems require run time exponential in the size of the problem. Various strategies have been proposed to tackle the problem, with research predominantly focusing on two avenues: the approximation method and the exact method.

Scholars have delved into a range of methods, utilizing heuristic and metaheuristic approaches to seek near-optimal solutions. Noteworthy strategies include the application of Genetic Algorithms, Estimation of Distribution Algorithms, Discrete Differential Evolution Algorithms, Linear Programming, Dynamic Programming, and Branch and Bound. These approaches aim to provide near-optimal schedules by efficiently navigating the intricate solution space of the JSSP. The selection of a particular method depends on factors such as problem size, constraints, and computational resources, reflecting ongoing efforts to refine and combine these techniques for improved approximations.

Our focus centers on the exact solution, particularly addressing the Job Shop Scheduling Problem (JSSP) through the application of satisfiability (SAT) algorithms. Crawford and Baker [2] conducted series of experiments applying SAT algorithms to scheduling problems. Koshimura and his team [8] utilized the encoding method introduced by Crawford and Baker to prove optimal upper bounds of ABZ9 (678) and YN1 (884) are indeed optimal. They also improved the upper bound of YN2 nad lower bounds of ABZ8, YN2, YN3 and YN4. Popov [4] extended the application of SAT algorithms to solve task-resource scheduling problems in cloud computing systems, while Cruz-Chávez and Rivera-Lopez [3] applied a local search algorithm to the logical representation of the JSSP. Hamadi and Jabbour [5] proposed a novel clause learning method crucial for modern SAT solvers. Additionally, Micheli and Mishchenko [10] extended the SAT formulation to find a minimum-size network under delay constraints.

1.3 Contributions of our methods

Achieving high-quality solutions for complex problems is often challenging due to either the sheer magnitude of the problem's complexity or suboptimal optimization performance. In this report, considering the prolonged execution time and associated challenges of utilizing the SAT algorithm to address intricate Job Shop Scheduling Problems (JSSP), we endeavor to enhance the SAT encoding method. Our aim is to reduce the

number of clauses and enhance the running efficiency within the solver, thereby striving for a more efficient resolution to complex problems.

1.4 Thesis report structure

In Section 2, we explain the problem and two open JSSPs (Ft20 and ABZ9) which we try to solve in this report. Section 3 delves into the foundational aspects, covering preliminaries and introducing key concepts such as Boolean formular, Boolean satisfiability problem, Conjunctive Normal Form, SAT Solver. The section then displays our optimal encoding technique. Data for experiments and results is considered in Section **??**.

Chapter 2

Problem definition

2.1 Job-shop scheduling problem

The job shop scheduling problem is a common scheduling problem. It involves determining the optimal sequence of processing multiple jobs on various machines to minimize the overall completion time. Each job comprises a series of tasks (operations) that must be executed in a specific order, and each task requires processing on a designated machine. A typical example is the production of a single consumer item, like an automobile, where different components are manufactured using different machines. The goal is to efficiently schedule these tasks on the available machines to achieve the shortest possible schedule duration, adhering to the following constraints:

- No task can commence until the preceding task for the same job is finished.
- Each machine can only handle one task at any given time.
- Once a task has begun, it must continue uninterrupted until completion.

2.1.1 Problem example

Below is a simple example of a JSSP, in which each task is labeled by a pair of numbers (m, t) where m is the number of the machine the task must be processed on and t is the processing time of the task. The numbering of jobs and machines starts at 0.

- $job_0 = [(0, 2), (2, 1), (1, 4)]$
- $job_1 = [(0, 3), (1, 2), (2, 2)]$

- $job_2 = [(1, 4), (2, 3), (0, 5)]$

In this example, there are 3 jobs. job_0 contains 3 tasks. The first, $(0, 2)$, must be processed on machine 0 in 2 units of time. The second, $(2, 1)$, must be processed on machine 2 in 1 unit of time, and so on. Altogether, there are 9 tasks.

2.1.2 A solution for the problem

A solution to the job shop problem is an assignment of a start time for each task, which meets the constraints given above. The diagram from Figure 2.1 shows one possible solution for the problem from Section 2.1.1.

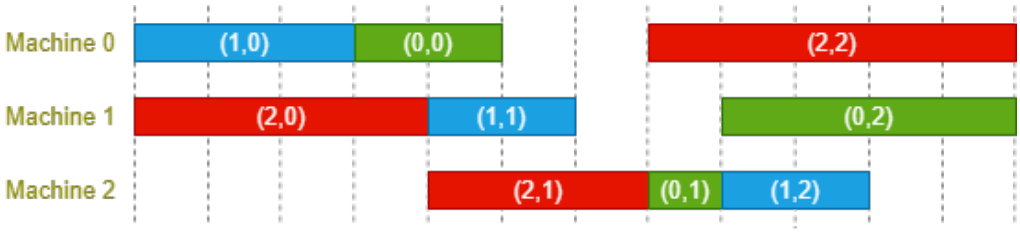

Figure 2.1: The JSSP solution is presented in the form of a diagram.

In the diagram, each task is represented by a rectangle with a length equal to the task's running time and a pair of numbers $(job\_index, task\_index)$. For example, rectangle labeled $(2, 1)$ represents task 1 of job 2 (the numbering of tasks also starts at 0) which have processing time of 3 units. In this solution, the tasks for each job are scheduled at non-overlapping time intervals, in the order given by the problem. Each machine only work on one task at a time and all tasks are not interrupted. The length of this solution is 12, which is the first time when all three jobs are complete. The objective of solving JSSPs is to minimize the length of the solution while still satisfying the given constraints.

2.1.3 Model as disjunctive graph

A JSSP consists of a finite set of m jobs $J = (j_1, j_2, ..., j_m)$, a finite set of n machines $M = (m_1, m_2, ..., m_n)$, each job consists of a chain of operations $O = (o_1, o_2, ..., o_k)$ and each operation has a processing time $\{\tau_1, \tau_2, ..., \tau_k\}$. In addition, all the work of all the operations is defined as the task $T = (t_1, t_2, ..., t_u)$. An instance of the JSSP problem can be represented by means of disjunctive graph $G = (O, A, E)$, the vertices in O represent the operations, the arcs in A represent the given precendence between the operations, the edges in $E = \{(v, w)|v, w \in O, v \neq w, M(v) = M(w)\}$ represent the machine capacity

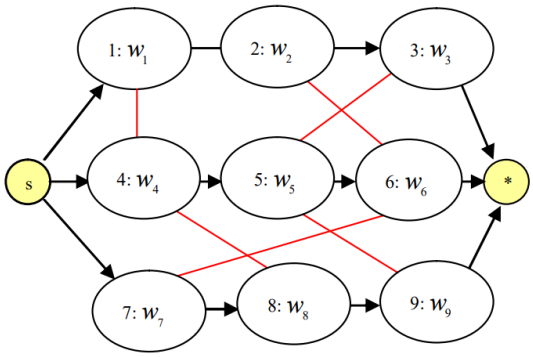

Figure 2.2: Disjunctive graph with edge orientations

constraints, each vertex v has a weight, equal to the processing time $\tau(v)$. The example is show in Fig 2.2. Finding an optimal feasible schedule is equivalent to finding an orientaion E' that minimizes the longest path length in the related graph.

2.2 Input file specifications

Jobshop instances are typically expressed in one of two forms: standard specification or Taillard specification. First is the most widely used specification and second is the specification used by Taillard [12]. In the standard specification, the input data is provided as a text file. On the first line are two numbers, the first is the number of jobs and the second the number of machines. Following there is a line for each job. The order for visiting the machines is presented together with the corresponding processing time. The numbering of the machines starts at 0. Each job has the same number of operations as the number of machines. For example an instance with two jobs on three machines. The first job has three operations where the processing time is 6 on machine 1, 7 on machine 2, 5 on machine 0 and the order that the machines are to be visited by that job is 1, 2, 0. The second job has three operations where the processing time is 4 on machine 0, 3 on machine 2, 9 on machine 1 and the order that the machines are to be visited by that job is 0, 2, 1. The instance would be presented as:

$$
\begin{array}{cccccc}
2 & 3 & & & & \\
1 & 6 & 2 & 7 & 0 & 5 \\
0 & 4 & 2 & 3 & 1 & 9
\end{array}
$$

In the Taillard specification, the input data is also provided as a text file. On the first line are two numbers, the first is the number of jobs and the second the number of machines. Following there are two matrice: the first with a line for each job containing the processor times for each operation, the second with the order for visiting the machines. The

numbering of the machines starts at 1. For example the same instance as above would be presented as:

$$
\begin{array}{ccc}
2 & 3 & \\
6 & 7 & 5 \\
4 & 3 & 9 \\
2 & 3 & 1 \\
1 & 3 & 2
\end{array}
$$

2.3 Output file

2.4 Open job-shop scheduling problems

There is a variety of benchmark problems for job shop: the well-known FT10 (size 10 × 10) and FT20 (20 × 5), and La21, La24, La25 (15 × 10), La27, La29 (20 × 10), La38, La40 (15 × 15), and ABZ7, ABZ8, ABZ9 (20 × 15), a set of 10 problems considered to be hard to solve for classical job shop. The set of Job Shop Scheduling Problems (JSSP) instances mentioned comprises well-established benchmarks commonly used as benchmarks in the academic literature to evaluate the performance of optimization algorithms. The introduction of these instances can be traced back to various sources, and they have been widely adopted in research papers and competitions. Ft10, Ft20 instances are part of the well-known "Fisher and Thompson" (Ft) benchmark set. Ft10 consists of 10 jobs and 10 machines, each job consist of 10 operations. Ft20 consists of 20 jobs and 5 machines, each job consists of 5 operations. Each ABZ*n* instance consists of 20 jobs and 15 machines. Each job of ABZ9 consists of 15 operations. In Section **??** we try to solve these two open JSSPs: Ft20 and ABZ9.

Chapter 3

Materials and Methods

3.1 Preliminaries

A proposition logic formula, also called boolean expression, is build from variables, operators, and parentheses. A Boolean variable x can take one of two possible value 0 ($False$) or 1 ($True$). There are different operators:

- **Negation**: is a unary operator, often denoted by $\neg$. It negates or flips the value of the variables.
- **Conjunction**: denoted by $\wedge$, is a binary operator between two variables that is only $True$ when both variables are $True$.
- **Disjunction**: is a binary operator between two variables that is $True$ when at least one of the variables is $True$, denoted by $\vee$.
- **XOR**: is a binary operator between two variables that is $True$ only when exactly one of the variables is $True$, denoted by $\oplus$.
- **Implication**: The implication operator, written as $\rightarrow$, is a binary operator between two variables. It is $True$, unless the first variable is $True$, while the second variable is $False$.
- **Equivalence**: The equivalence or if and only if operator, denoted by $\leftrightarrow$, is a binary operator between two variables that is $True$, only when both variables have the same value — either both $True$, or both $False$.

Table 3.1: Truth table for negation operator

x	$\neg x$
1	0
0	1

A truth table is a tabular representation of all possible combinations of truth values for a logical expression. It systematically shows the outcomes of a logical operation or a set of propositions under different truth conditions. Each row in the table corresponds to a unique combination of truth values for the variables or propositions involved in the expression. The truth tables for the operators mentioned above are displayed in Table 3.1 and Table 3.2.

Table 3.2: Truth table for binary operators

x	y	$x \wedge y$	$x \vee y$	$x \oplus y$	$x \rightarrow y$	$x \leftrightarrow y$
1	1	1	1	0	1	1
1	0	0	1	1	0	0
0	1	0	1	1	1	0
0	0	0	0	0	1	1

3.2 Boolean satisfiability problem

In context of computer science, the Boolean statisfiability problem (SAT) is the problem of deciding if there exists a truth assignment that satisfies a Boolean formula. Given a boolean formula, SAT asks whether the variables can be replaced by the values $True$ or $False$ in order to make the formula evaluate to $True$. If no such assignment exists - the formula is $False$ for all possible variable assignments, we call it unsatisfiable. Otherwise, the formular is called satisfiable. For example, the formular $a \wedge \neg b$ is satisfiable because we can find the values $a = True$ and $b = False$ which make $a \wedge \neg b = True$. In constrast, $a \wedge \neg a$ is unsatisfiable because it is always $False$ whether $a = True$ or $a = False$.

3.3 Conjunctive normal form

A literal might be interpreted as the negation of a variable (negative literal) or as a variable (positive literal). A disjunction of literals or a single literal is called a clause. If a formula consists of a single clause or a conjunction of clauses, it is in conjunctive normal form (CNF). For example, x_1 is a positive literal, $\neg x_2$ is a negative literal, $x_1 \vee \neg x_2$ is a clause, the formula 3.1 is in CNF.

$$(x_1 \vee \neg x2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge \neg x1 \tag{3.1}$$

The formula 3.1 is satisfiable, by choosing $x1 = False$, $x2 = False$ and $x3$ arbitrarily. Every propositional logic formula can be converted into an equivalent conjunctive normal form using the rules of Boolean algebra; this form, however, may be exponentially longer.

3.4 Problem-Solving Methodology

In this report, the JSSPs are solved using the following method (Figure 3.1):

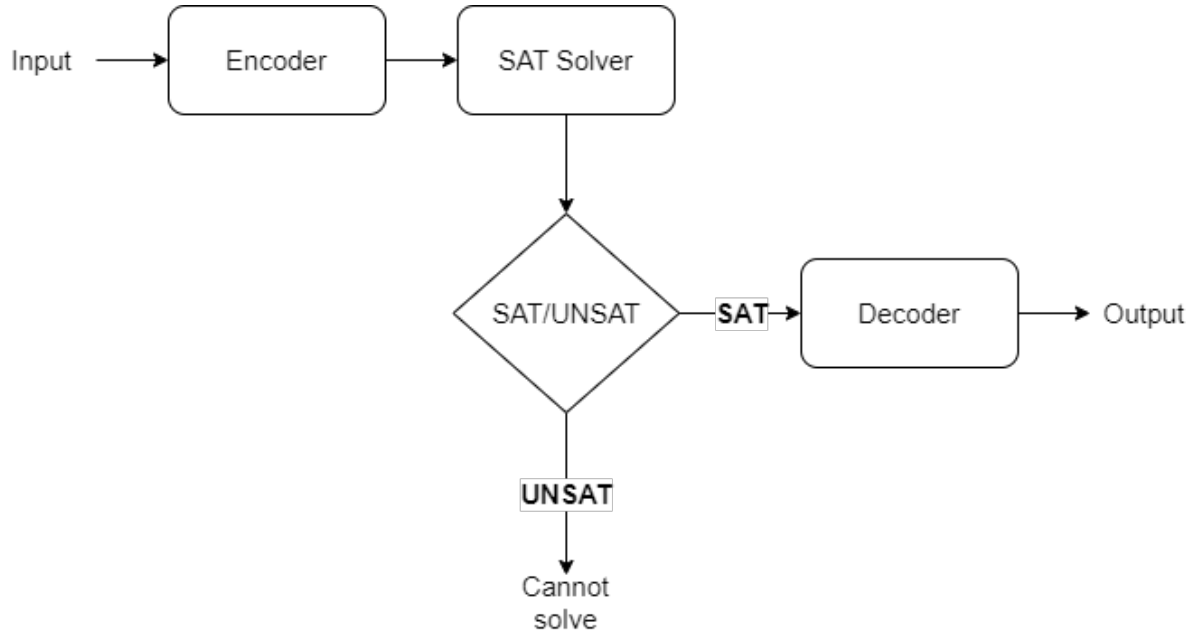

Figure 3.1: SAT execution flow chart

- ***Input***: The input for the problem is represented in the form of standard specification (Section 2.2).
- ***Encoder***: Encoding all the constraints of input to conjunctive normal form (CNF). Encoder will generate a cnf file then pass it to the SAT Solver.

- ***SAT Solver***: From the cnf file, calculate the truth value (True or False) of all the propositional variables. If there does not exist a set of propositional variable values that satisfy the input formula, then return UNSAT; otherwise, return SAT along with those values passed to the Decoder.
- ***Decoder***: Reverse the encoding process and combine it with the truth values to produce the output.
- ***Output***: An assignment of a start time for each task, which meets the problem constraints.

3.5 Makespan, upper bound, lower bound

Under all constraints, the time required to complete all the jobs is called as the makespan L. Our method does not directly calculate the makespan but will instead assign a trial value to the makespan and then try to solve the problem with that assigned makespan.

If the problem can be solved with a certain value of makespan, we call that makespan value satisfiable. If the makespan $L = l$ is satisfiable, all the bigger makespan $(l + 1, l + 2, ...)$ is also satisfiable. This can be proven by assuming that the problem output has been found with a makespan $L = l$. By adjusting the start time of the final task in the output and increasing it by 1, we obtain a new output that satisfies the problem with a makespan $L = l + 1$. Similarly, it can be proven that all makespans greater than l are satisfiable. Therefore, we call this value l the upper bound (UB) for the problem because the optimal result is definitely not larger than l. If any value lower than l is satisfiable, it can be a new UB instead of l.

If the makespan $L = l$ is not satisfiable, all jobs cannot be completed in time l. Therefore, in a shorter period of time, it is even more impossible to complete all jobs. All the lower makespan $(l - 1, l - 2, ...)$ is also unsatisfiable. This value l can be called the problem lower bound (LB) because the optimal result is definitely bigger than l. If we find a value bigger than l which is also unsatisfiable, it can be a new LB instead.

Let S_L be a SAT problem generated by our encoding method under assumption that the makespan equals L. If we find a positive integer k such that S_{k-1} is unsatisfiable and S_k is satisfiable, the we conclude that the minimum makespan is k. Such a k divides SAT problems S_i $(i \geq 1)$ into an unsatisfiable part S_i $(1 \leq i \leq k - 1)$ and a satisfiable part S_i $(i \geq k)$.

For common JSSPs, most UB and LB have been established in various studies.

When we know the UB, LB of the problem, we only need to find the optimal makespan within the range [LB, UB]. In cases of entirely new problems, we can only estimate the value of LB and UB.

3.6 Encoding method

The JSSP is represented in CNF and translated into a SAT problem by the encoding approach proposed by Crawford and Baker [2]. This section introduces the encoding method and provides an illustrative example of encoding for the simple problem introduced in Section 2.1.1. In this section, we call that problem the *sample* problem.

A JSSP consists of a set of n jobs $\{J_0, J_1, ..., J_{n-1}\}$ and a set of m machines $\{M_0, M_1, ..., M_{m-1}\}$. Each job J_l is a sequence of tasks $(O_0^l, O_1^l, ..., O_{q_l-1}^l)$ with q_l is the number of tasks in job J_l. Each task O_i^l is performed on machine $M_{O_i^l}$ for an uninterupted duration p_i^l - processing time. Machine $M_{O_i^l}$ belongs to set of m machines above. A *schedule* or an *output* is is a set of start times for each task O_i^l. The time required to finish all the jobs is called the *makespan* L. As previously introduced, we assign a value to the makespan for calculation.

There are three kinds of propositional variables:

- $pr_{i,j}^{l,k}$ stating that task O_i^l precedes O_j^k.
- $sa_{i,t}^l$ stating that task O_i^l starts at t or after.
- $eb_{i,t}^l$ stating that task O_i^l ends by t or before.

Scheduling constraints are translated by the following rules. Each rule represents a type of clause in our CNF encoding. The key objective here is the identification of the truth values of the propositional variables that satisfy all these rules, which means satisfying the conditions of the problem.

1. O_i^l precedes O_{i+1}^l:

$$pr_{i,i+1}^{l,l} \ (0 \leq l \leq n-1, \ 0 \leq i \leq q_l - 2)$$

This rule corresponds to the constraint that no task can begin until the previous task for the same job is completed. For the sample problem, we have the following clauses:
$pr_{0,1}^{0,0}$

$pr_{1,2}^{0,0}$
$pr_{0,1}^{1,1}$
$pr_{1,2}^{1,1}$
$pr_{0,1}^{2,2}$
$pr_{1,2}^{2,2}$

2. If O_i^l and O_j^k require the same machines ($M_{O_i^l} = M_{O_j^k}$), then O_i^l precedes O_j^k or O_j^k precedes O_i^l:

$$pr_{i,j}^{l,k} \vee pr_{j,i}^{k,l} \ (0 \leq l < k \leq n-1,\ 0 \leq i \leq q_l - 1,\ 0 \leq j \leq q_k - 1)$$

This rule corresponds to the constraint that each machine can only handle one task at any given time. For the sample problem, we have the following clauses:
$pr_{0,0}^{0,1} \vee pr_{0,0}^{1,0}$
$pr_{0,2}^{0,2} \vee pr_{2,0}^{2,0}$
$pr_{0,2}^{1,2} \vee pr_{2,0}^{2,1}$
...
$pr_{1,0}^{1,2} \vee pr_{0,1}^{2,1}$

3. Task O_i^l can only start after all previous task $\{O_0^l, O_1^l, ..., O_{i-1}^l\}$ of the same job J_l end. It requires at least $t = \sum_{u=0}^{i-1} p_u^l$. That is, O_i^l starts at time t or after:

$$sa_{i,t}^l \ (0 \leq l \leq n-1,\ 0 \leq i \leq q_l - 1,\ t = \sum_{u=0}^{i-1} p_u^l)$$

For the sample problem, we have the following clauses:
$sa_{0,0}^0$
$sa_{1,2}^0$
$sa_{2,3}^0$
...
$sa_{2,7}^2$

4. Conversely, to complete all the jobs by L, task O_i^l must end by time $t = L - \sum_{u=i+1}^{q_l-1} p_u^l$. The duration $\sum_{u=i+1}^{q_l-1} p_u^l$ is the minimum time required to complete all later tasks $\{O_{i+1}^l, O_{i+2}^l, ..., O_{q_l-1}^l\}$ of the same job J_l.

$$eb_{i,t}^l \ (0 \leq l \leq n-1,\ 0 \leq i \leq q_l - 1,\ t = L - \sum_{u=i+1}^{q_l-1} p_u^l)$$

For the sample problem, we have the following clauses:
$eb_{0,L-5}^0$

$eb^0_{1,L-4}$
$eb^0_{2,L}$
...
$eb^2_{2,L}$

5. If O^l_i starts at t or after is True, it starts at $t-1$ or after is also True:

$$sa^l_{i,t} \rightarrow sa^l_{i,t-1}\ (0 \leq l \leq n-1,\ 0 \leq i \leq q_l - 1,\ 1 \leq t \leq L)$$

For the sample problem, we have the following clauses:
$sa^0_{0,1} \rightarrow sa^0_{0,0}$
$sa^0_{0,2} \rightarrow sa^0_{0,1}$
$sa^0_{0,3} \rightarrow sa^0_{0,2}$
...
$sa^0_{0,L} \rightarrow sa^0_{0,L-1}$
$sa^0_{1,1} \rightarrow sa^0_{1,0}$
$sa^0_{1,2} \rightarrow sa^0_{1,1}$
...
$sa^2_{2,L} \rightarrow sa^2_{2,L-1}$

6. If O^l_i ends by t or before is True, it ends by $t+1$ or before is also True:

$$eb^l_{i,t} \rightarrow eb^l_{i,t+1}\ (0 \leq l \leq n-1,\ 0 \leq i \leq q_l - 1,\ 0 \leq t \leq L-1)$$

For the sample problem, we have the following clauses:
$eb^0_{0,0} \rightarrow eb^0_{0,1}$
$eb^0_{0,1} \rightarrow eb^0_{0,2}$
...
$eb^0_{0,L-1} \rightarrow eb^0_{0,L}$
$eb^0_{1,0} \rightarrow eb^0_{1,1}$
...
$eb^2_{2,L-1} \rightarrow eb^2_{2,L}$

7. If O^l_i starts at t or after, it must end at $t+p^l_i$ or after. Therefore, it cannot end by $t+p^l_i-1$ or before:

$$sa^l_{i,t} \rightarrow \neg eb^l_{i,t+p^l_i-1}\ (0 \leq l \leq n-1,\ 0 \leq i \leq q_l - 1,\ 0 \leq t \leq L-p^l_i+1)$$

For the sample problem, we have the following clauses:
$sa^0_{0,0} \rightarrow \neg eb^0_{0,1}$

$sa^0_{0,1} \rightarrow \neg eb^0_{0,2}$

...

$sa^2_{2,L-4} \rightarrow \neg eb^2_{2,L}$

8. If O^l_i start at t or after and O^l_i precedes O^k_j, O^k_j cannot start until O^l_i is finished. This means O^k_j can only start at $t + p^l_i$ or after. For each $pr^{l,k}_{i,j}$ from rule 1 and rule 2, we add the clause:

$$sa^l_{i,t} \wedge pr^{l,k}_{i,j} \rightarrow sa^k_{j,t+p^l_i}$$

$$(0 \leq l \leq n-1,\ 0 \leq k \leq n-1,\ 0 \leq i \leq q_l - 1,\ 0 \leq j \leq q_k - 1,\ 0 \leq t \leq L - p^l_i)$$

For the sample problem, we have the following clauses:

$sa^0_{0,0} \wedge pr^{0,0}_{0,1} \rightarrow sa^0_{1,2}$

$sa^0_{0,0} \wedge pr^{0,1}_{0,0} \rightarrow sa^1_{0,2}$

...

$sa^2_{2,L-5} \wedge pr^{2,1}_{2,0} \rightarrow sa^1_{0,L}$

3.7 SAT solver

3.7.1 Definition of SAT solver

A SAT solver is an algorithm for establishing satisfiability. It takes the Boolean logic formula as input and returns SAT if it finds a combination of variables that can satisfy it or UNSAT if it can demonstrate that no such combination exists. In addition, it may sometimes return without an answer if it cannot determine whether the problem is SAT or UNSAT.

3.7.2 MiniSAT solver

This research uses MiniSAT solver developed by Niklas Sörensson and Niklas Een [11]. It is a minimalistic, open-source SAT solver, developed for both researchers and developers; it is released under the "MIT license".

MiniSAT introduces significant advancements in SAT-solving capabilities. The solver incorporates a resolution-based conflict clause minimization technique, rooted in self-subsuming resolution. Experiments demonstrate that this innovation can eliminate over 30% of redundant literals in conflict clauses, leading to reduced memory consumption and the generation of stronger clauses. Noteworthy improvements include an enhanced Variable State Independent Decaying Sum (VSIDS) decision heuristic, where

variable activities decay by 5% after each conflict, outperforming the original VSIDS. The solver also introduces features such as binary clause implementation, aggressive clause deletion, and incremental SAT interface. The utilization of a two-literal watch scheme and conflict analysis further enhances the solver's efficiency. MiniSAT v1.13's commitment to minimizing learned clauses and optimizing decision heuristics positions it as a competitive and efficient SAT solver, particularly demonstrated by its participation in the SAT Competition 2005.

3.7.3 MiniSAT input format

The MiniSAT solver, similar to most SAT solvers, processes input provided in a simplified "DIMACS CNF" format, which is a simple text format and mostly-line-oriented format. Comment lines are identified by the letter "c" at the beginning, and they do not affect the input interpretation. The initial non-comment line serves to initiate the input and includes essential information such as the count of variables and clauses involved in the problem. This specific line is required to follow the structure:

```
p cnf [number of variables] [number of clauses]
```

The next non-comment lines define clauses. Every clause consists of a space-delimited sequence of 1-based variable indices. A positive value denotes a specific variable, while a negative value signifies the negation of a variable. The end of a clause is indicated by a final value of "0". The input file concludes once the last clause has been terminated.

Consider an example of CNF is:

$$(x_1 \vee \neg x_5 \vee x_4) \wedge (\neg x_1 \vee x_5 \vee x_3 \vee x_4) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_2 \vee x_5)$$

The CNF expressions above would be written as MiniSAT input format:

```
c Here is a comment
p cnf 5 4
1 -5 4 0
-1 5 3 4 0
-3 4 0
-2 5 0
```

The "p cnf" line means that this is SAT problem in CNF format with 5 variables and 4 clauses. The first line after it is the first clause, meaning $(x_1 \vee \neg x_5 \vee x_4)$. The number

1 means x_1 and number -5 means $\neg x_5$. The solver's job is to find the set of boolean variable assignments that make all the clauses true.

3.7.4 MiniSAT output format

Upon execution, miniSAT provides several statistics related to its performance on standard error. Depending on whether a solution exists for the given problem, miniSAT will output either "SATISFIABLE" or "UNSATISFIABLE" (without quotation marks) to standard output. If a RESULT-OUTPUT-FILE is specified, miniSAT will write the results to that file. The first line will indicate whether the problem is satisfiable ("SAT") or unsatisfiable ("UNSAT"). In case of a satisfiable problem, the second line will contain an assignment of boolean variables that satisfies the given expression. It's important to note that while there might be multiple solutions, miniSAT is only required to provide one valid assignment.

For the provided example input from Section 3.7.3, the RESULT-OUTPUT-FILE will contain the following output:

```
SAT
1 2 -3 4 5 0
```

This output means that the example problem is satisfiable, with $x_1 = T,\ x_2 = T,\ x_3 = F,\ x_4 = T,\ x_5 = T$ (where T is True and F is False).

3.7.5 Getting more solutions

In case of wanting to obtain an alternative solution, one straightforward approach is to create new input includes a new clause that negates the previous solution. Note that the count of clauses has increased, the count of variables is the same. In the context of our example problem, we could proceed as follows:

```
c Here is a comment
p cnf 5 5
1 -5 4 0
-1 5 3 4 0
-3 4 0
-2 5 0
-1 -2 3 -4 -5 0
```

The last clause of the input above negates the previous solution. After putting this second input, the new output is:

```
SAT
1 -2 -3 4 5 0
```

The new solution $x_1 = T,\ x_2 = F,\ x_3 = F,\ x_4 = T,\ x_5 = T$ also satisfies the problem.

3.7.6 Generate cnf file after encoding

The clauses from Encoding step (Section 3.6) need to be translated to MiniSAT input format before put to the solver. This step can be done by using *bidirectional map* data structure. In computer science, a bidirectional map is an associative data structure in which the $(key,\ value)$ pairs from a one-to-one correspondence. Thus the binary relation is functional in each direction: each $value$ can also be mapped to a unique key. A pair $x,\ y$ thus provides a unique coupling between x and y so that y can be found when x is used as a key and x can be found when y is used as a key. Figure 3.2 describe a bidirectional map with some pairs of $(x,\ y)$.

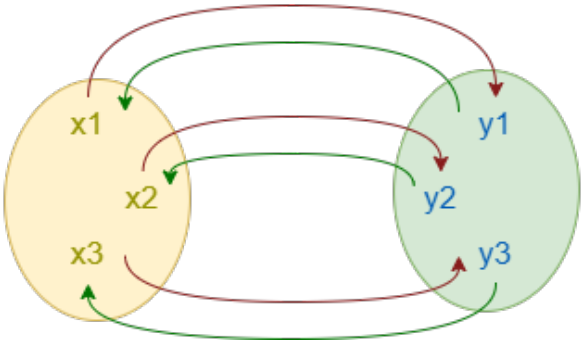

Figure 3.2: Bidirectional map for pairs of $(x,\ y)$

The propositional variables is mapped to continuous positive numbers called *variable index* based on their order of appearance, starting from the number 1. This mapping allows the solver to distinguish between variables. Bidirectional map is used instead of standard map because the decoding step needs to find variables when use *variable index* as a key.

Mapping for the sample problem in case of $L = 12$ is as follows:

$$pr_{0,1}^{0,0} \leftrightarrow 1$$

$$pr_{1,2}^{0,0} \leftrightarrow 2$$

...

$sa^0_{0,1} \leftrightarrow 27$

$sa^0_{0,2} \leftrightarrow 28$

...

$eb^0_{0,0} \leftrightarrow 39$

$eb^0_{0,1} \leftrightarrow 40$

...

$eb^2_{2,11} \leftrightarrow 258$

Each clause is transformed into a line in the solver input file using rules similar to those in Table 3.3. Each line is end with a number 0.

Table 3.3: Example of transformed clause for the sample problem in case of $L = 12$

Clause	**Line**
$pr^{0,0}_{0,1}$	1 0
$pr^{0,1}_{0,0} \vee pr^{1,0}_{0,0}$	7 8 0
$sa^0_{0,1} \rightarrow sa^0_{0,0}$	-27 25 0
$sa^0_{0,0} \rightarrow \neg eb^0_{0,1}$	-25 -40 0
$sa^0_{0,0} \wedge pr^{0,0}_{0,1} \rightarrow sa^0_{1,2}$	-25 -1 51 0

3.8 Decoding method

If the solver step results in satisfiable (SAT), we need a decoding step to convert the solver's output into the problem's output. The problem's output includes the start time for each task. From the Minisat output and bidirectional map, we found the truth assignment for propositional variables. Call this truth assignment M. The start time s^l_i of operation O^l_i is given by extracting the variable $sa^l_{i,t}$ which has the biggest time and is still assigned as true in M . More precisely, s^l_i is given by which satisfies the following expression:

$$(s^l_i \ = \ max\ of\ T^l_i) \wedge (T^l_i \ = \ \{t \mid sa^l_{i,t} \ = \ True\})$$

For the sample problem with $L = 12$, the start after variables of task O^0_0 has the truth assignment as in Table 3.4. So the start time for this task is 3.

Table 3.4: Some truth assignment of the sample problem with $L = 12$

Variable	Truth assignment
$sa^0_{0,0}$	True
$sa^0_{0,1}$	True
$sa^0_{0,2}$	True
$sa^0_{0,3}$	True
$sa^0_{0,4}$	False
$sa^0_{0,5}$	False
...	...
$sa^0_{0,12}$	False

Chapter 4

Experiments and evaluations

4.1 Experimental environment

All experiments were conduct on the personal computer. We use Minisat SAT solver [11] to solve the problems. Details of the hardware configuration of the experimental computer are described in Table 4.1 below:

Table 4.1: Configuration of the experimental environment

Processor	AMD Ryzen 5 5500U with Radeon Graphics 2.10 GHz
RAM	24.0 GB
Operating system	Windows 11 Home Single Language version 23H2
Programming language	Python 3.11.5
SAT Solver	Minisat v1.13

4.2 Method for experiments

4.2.1 Estimate lower bound and upper bound

This section introduces a method for estimating LB and UB values for an entirely new problem that has not had these bounds determined in previous research. The aim is to estimate higher LB values and lower UB values, meaning that the closer the LB and UB are to the optimal result, the better. This approach ultimately helps reduce the number of makespan calculations required.

To find the LB, we need to rely on the encoding rules outlined in Section 3.6. Some

rules may potentially create negative time values if the makespan L is too low, as exemplified by Rule 4 with $t = L - \sum_{u=i+1}^{q_l-1} p_u^l$. The LB we seek is the smallest makespan L that satisfies the condition of ensuring no negative time values for all variables. First, encode the problem with $L = 0$, and find the smallest time value t_{min} among the variables. Value t_{min} will be a negative value. The LB is then derived as the absolute value of that time t.

The desired UB value needs to be a satisfiable makespan value. The simplest approach involves arranging task executions sequentially, similar to the task order in the input: all tasks of job 0 must finish before task 0 of job 1 can start and so on. The task execution order can be represented as follows:

$$O_0^0,\ O_1^0, ...,\ O_{q_0-1}^0,\ O_0^1,\ O_1^1, ..., O_{q_n-1}^{n-1}$$

Although this approach results in a large makespan value, it certainly guarantees the satisfaction of all three JSSP constraints. With this method, at any given point in time, only one task is executed, and only one machine is active. Therefore, it ensures that no conflicts or resource contention arise during the execution of tasks.

4.2.2 Find optimal makespan

After determining LB and UB, the optimal makespan is certainly within the range [LB, UB]. Because optimal makespan divides SAT problems into an unsatisfiable part and a satisfiable part (Section 3.5), we can quickly obtain the result using an algorithm based on binary search. The algorithm is described in detail as Algorithm 1. The introduced algorithm has a complexity of $O(log(n))$ with $n = UB - LB$.

4.3 Experimental results

According to the previously proposed method, we tried to solve two JSSPs: La03 and Orb07. La03 consists of 10 jobs and 5 machines, each job consists of 5 operations and was first introduced by Lawrence in 1984 [9]. Orb07 consists of 10 jobs and 10 machines, each job consists of 10 operations and was first introduced by Applegate and Cook in 1991 [1].

Both problems already have known optimal solutions. La03 was first solved by Applegate and Cook [1] and Orb07 was first solved by Henning [7]. However, to validate the proposed method, this research will solve these problems again from the beginning.

For each problem P, let S_L^P be a SAT instance generated by our encoding method

Algorithm 1 Algorithm to find the optimal makespan

Require: $0 < low < up$ (with low is lower bound, up is upper bound)

Ensure: op is optimal makespan

```
while low ≤ up do
    mid ← (low + up)/2
    if mid is satisfiable then
        if mid − 1 is not satisfiable then
            op ← mid
        else if mid − 1 is satisfiable then
            up ← mid − 1
        end if
    else if mid is not satisfiable then
        if mid + 1 is satisfiable then
            op ← mid + 1
        else if mid + 1 is not satisfiable then
            low ← mid + 1
        end if
    end if
end while
```

under assumption that the makespan equals L. Even though, S^P_{LB} is unsatisfiable, S^P_{UB} is satisfiable as proven in Section 4.2.1, we still solve them to verify the values.

4.3.1 Solving La03 and Orb07

LB and UB was also calculated based on our proposed method. Table 4.2 show the LB and UB results.

Table 4.2: Results of LB and UB

Problem	**LB**	**UB**
La03	268	2383
Orb07	258	2407

Table 4.3 and 4.4 show the experimental results of S^{La03}_L and S^{Orb07}_L, respectively. The values of L are calculated using the algorithm presented in Section 4.2.2. The second and third columns represent the size of the SAT instance. The second column

Table 4.3: Result of S_L^{La03}

L	Number		CPU (secs)	Satisfiable
	Variables	Clauses		
268	27391	146620	25.72	no
531	53690	314940	49.40	no
532	53790	315580	51.33	no
596	60190	356540	57.38	no
597	60290	357180	65.18	yes
662	66790	398780	70.09	yes
663	66890	399420	78.96	yes
795	80090	483900	90.32	yes
796	80190	484540	96.19	yes
1324	132990	822460	165.95	yes
1325	133090	823100	166.95	yes
2383	238890	1500220	404.89	yes

displays the count of propositional variables, while the third column presents the number of clauses involved in the instance.

As N grows, the size of SAT instances exhibits a linear increase. However, the CPU time required for computation escalates exponentially. This exponential reflects NP-hardness of JSSPs.

We have successfully solved La03 and Orb07. The makespan of the optimal solutions for La03 and Orb07 are 597 and 397, respectively. These results are completely consistent with the solutions from other research.

When the calculated values of LB and UB are significantly distant from each other, more computation steps are required. Notably, a very large UB increases the computation time dramatically, significantly affecting the solution's efficiency. To apply this method to other problems with a large number of jobs and machines, we need to optimize UB estimating.

Table 4.4: Result of S_L^{Orb07}

L	Number		CPU (secs)	Satisfiable
	Variables	Clauses		
258	52792	308506	52.38	no
391	79392	480076	77.95	no
392	79592	481366	63.14	no
395	80192	485236	80.34	no
396	80392	486526	74.64	no
397	80592	487816	76.85	yes
398	80792	489106	90.23	yes
399	80992	490396	94.35	yes
406	82392	499426	95.91	yes
407	82592	500716	96.46	yes
423	85792	521356	103.68	yes
424	85992	522646	102.14	yes
457	92592	565216	113.33	yes
458	92792	566506	112.75	yes
524	105992	651646	130.34	yes
525	106192	652936	128.62	yes
793	159792	998656	218.39	yes
794	159992	999946	225.54	yes
1331	267392	1692676	449.77	yes
1332	267592	1693966	466.02	yes
2407	482592	3080716	1098.74	yes

References

[1] D. L. Applegate and W. J. Cook, "A computational study of the job-shop scheduling problem," vol. 3, no. 2, pp. 149–156, May 1991, the JSSP instances used were generated in Bonn in 1986.

[2] J. M. Crawford and A. B. Baker, "Experimental results on the application of satisfiability algorithms to scheduling problems," in *Proceedings of the Twelfth AAAI National Conference on Artificial Intelligence*, ser. AAAI'94. AAAI Press, 1994, p. 1092–1097.

[3] M. A. Cruz-Chávez and R. Rivera-López, "A local search algorithm for a sat representation of scheduling problems," *Lecture Notes in Computer Science*, p. 697–709, 2007.

[4] A. Gorbenko and V. Popov, "Task-resource scheduling problem," *International Journal of Automation and Computing*, vol. 9, no. 4, p. 429–441, 2012.

[5] Y. Hamadi, S. Jabbour, and J. Sais, "Control-based clause sharing in parallel sat solving," *Autonomous Search*, p. 245–267, 2011.

[6] I. A. Hashem, N. B. Anuar, M. Marjani, A. Gani, A. K. Sangaiah, and A. K. Sakariyah, "Multi-objective scheduling of mapreduce jobs in big data processing," *Multimedia Tools and Applications*, vol. 77, no. 8, p. 9979–9994, 2017.

[7] A. Henning, "Praktische Job-Shop Scheduling-Probleme," Ph.D. dissertation, Friedrich-Schiller-Universität Jena, Jena, Germany, Aug. 2002, alternate url: https://nbn-resolving.org/urn:nbn:de:gbv:27-20060809-115700-4. [Online]. Available: http://www.db-thueringen.de/servlets/DocumentServlet?id=873

[8] M. Koshimura, H. Nabeshima, H. Fujita, and R. Hasegawa, "Solving open job-shop scheduling problems by sat encoding," *IEICE Transactions on Information and Systems*, vol. E93.D, no. 8, pp. 2316–2318, 2010.

[9] S. R. Lawrence, "Resource constrained project scheduling: An experimental investigation of heuristic scheduling techniques (supplement)," Ph.D. dissertation, Graduate School of Industrial Administration (GSIA), Carnegie-Mellon University, Pittsburgh, PA, USA, 1984.

[10] M. Soeken, G. De Micheli, and A. Mishchenko, "Busy man's synthesis: Combinational delay optimization with sat," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2017*, 2017, pp. 830–835.

[11] N. Sörensson and N. Een, "Minisat v1.13-a sat solver with conflict-clause minimization," *International Conference on Theory and Applications of Satisfiability Testing*, 01 2005.

[12] E. Taillard, "Benchmarks for basic scheduling problems," *European Journal of Operational Research*, vol. 64, no. 2, p. 278–285, Jan 1993.