*Advances in Engineering Research, volume 164*
8th International Conference on Manufacturing Science and Engineering (ICMSE 2018)

# An efficient SAT algorithm for complex job-shop scheduling

Hong Huang[1, a] and Shaohua Zhou[1,b]

[1]59 Qinglong Road, Southwest University of Science and Technology, Mianyang, Sichuan, P.R.China 621010

[a]huanghong@swust.edu.cn, [b]23082945@qq.com

**Keywords:** job-shop scheduling; Boolean Satisfiability; coding optimization; exact solution

**Abstract.** The job-shop scheduling problem (JSSP) is a typical scheduling problem, which belongs to NP-hard problem. This paper tries to solve the complex JSSP by translating it into Boolean Satisfiability Problem. Even though the representation of JSSP in SAT is not a new issue, the solution to the complex JSSP is still a difficult problem because of processing time too long. In this paper, we optimized the SAT encoding method, thus reducing the number of clauses and their processing efficiency in the solver. We used the Ft20 and ABZ8 problem to experiment, the results show that the optimized coding method can greatly improve the processing efficiency.

## Introduction

Production scheduling system is a research hotspot in manufacturing system and one of the most difficult problems in theoretical research. The task of scheduling is to determine the processing path, time, machine and operation for each machining object according to the production objectives and constraints. Excellent scheduling strategy for improving the optimization of production systems, improving economic efficiency, has a great role [1].

The JSSP is a typical scheduling problem, which is one of production scheduling system problems. It has been widely used in many fields [2-4]. The JSSP belong to NP-hard problem, and many researcher have studying it. Various approaches have been proposed for solving the problem. The related work mainly takes two methods to study: approximation method and exact method.

In the case of approximate solutions, many scholars have proposed many methods in recent years. Genetic Algorithm [5, 6], estimation of distribution algorithm, discrete differential evolution algorithm, linear programming, dynamic programming, branch definition and other algorithm [7-11], have been widely used in the JSSP.

The exact solution is mainly relevant to our works. At present, the researchers focus on solving the JSSP problem with SAT Crawford and Baker [12] gave a series of experiments applying satisfiability algorithms to solve scheduling problems. Koshimura and his team [13] used the encoding method proposed by Crawford and Baker, proved that the best known upper bounds 678 of ABZ9 and 884 of YN1 are indeed optimal. In addition, they improved the upper bound of YN2 and lower bounds of ABZ8, YN2, YN3 and YN4. Popov [14] proposed using satisfiability algorithms to solve the task-resource scheduling problems in cloud computing system. Cruz-Chávez and Rivera-Lopez [15] gave the application of a local search algorithm for a logical representation of the JSSP problem. Hamadi and Youssef [15] proposed a new clause learning method, which is the important for modern SAT solvers. Micheli and Mishchenko, A extended the SAT formulation to find a minimum-size network under delay constraints [17]. Other scholars have done a lot of work in the field [18].

To the exact solution, or the complexity is large, or the optimization performance is poor, it is difficult to high-quality solution complex problems [13]. In this paper, taking into account the SAT algorithm to solve complex JSSP problem takes a long time and other issues, we tries to optimize the SAT encoding method, reducing the number of clauses and their running efficiency in the solver, so as to achieve a more efficient goal of solving complex problems. In section 2, we consider the problem of finding an optimal feasible schedule. In particular, we show the relationship between the graph and JSSP. In section 3, we represent the optimized SAT coding method. Data for experiments is considered in section 4. In section 5, we propose a data automatic analysis method, which can generate the Gantt chart to represent the solution.

## Problem definition

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 processing time $\{t_1, t_2, ...t_k\}$. In addition, all the work of all the operations is defined as the task $T=(t_1, t_2, ..., t_u)$ in the paper.

An instance of the JSSP problem can be represented by means of a disjunctive graph $G=(O, A, E)$, the vertices in $O$ represent the operations, the arcs in A represent the given precedence between the operations, the edge in $E=\{(v, w) \mid v, w \in O, v \neq w, M(v)=M(w)\}$ represent the machine capacity constraints, each vertex v has a weight, equal to the processing time $t(v)$. The example is shown in Fig. 1.

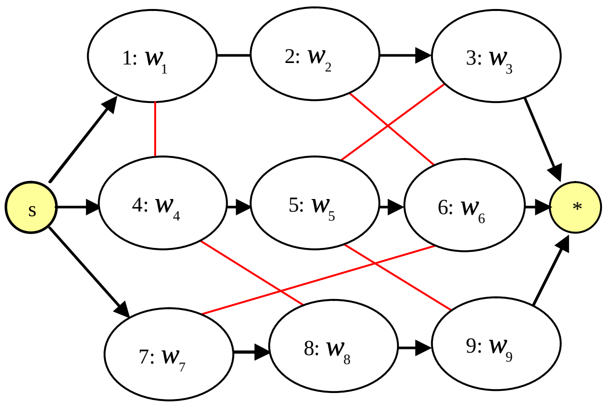

Fig.1 Disjunctive graph with edge orientations

Finding an optimal feasible schedule is equivalent to finding an orientation *E'* that minimizes the longest path length in the related digraph.

## SAT encoding

Each operation $o$ consists of four basic elements: (1) a starting time $s$, (2) a processing time $p$, (3) a deadline time $d$, (4) a ready time $r$. At the same time, we need to define several constraints: (1) If operation $o_i$ precedes $o_j$, We can express it as $pr_{o_i,o_j}(1 \leq o_i \leq n3, 1 \leq o_j \leq n3)$, (3)Resource capacity constraints, written $c_{o_i,o_j}$, state operation $o_i$ and $o_j$ conflict, because operation $o_i$ and $o_j$ require the same machine. Under all constraints, the time required to complete all the jobs is called as the makespan $L$. The objective of the JSSP is to determine the schedule which minimizes $L$.

We introduce two kinds of propositional variables:

- $sa_{i,t}$, it states that operation $i$ starts at *t* or after.
- $eb_{i,t}$, it states that operation $i$ ends at *t* or before.

We translate the JSSP into a SAT problem by the Crawford encoding [4]. It includes constraints and coherence conditions coding. In the translation, we do a lot of optimization to improve the efficiency of coding operation. Scheduling constraints are then translated by:

1. $pr_{o_i,o_j}(1 \leq o_i \leq n3, 1 \leq o_j \leq n3)$

It states that $\sigma_i$ precedes $\sigma_j$.

2. $pr_{o_i,o_j} \vee pr_{o_j,o_i}(1 \leq o_i \leq n3, 1 \leq o_j \leq n3)$

If $o_i$ and $o_j$ require the same machines, then $o_i$ precedes $o_j$ or $o_j$ precedes $o_i$.

3. $pr_{o_i,o_j} \in U \wedge U - pr_{o_i,o_j} = \emptyset$

We define the set of $pr_{o_i,o_j}$ is U.

4.$sa_{o_i,t}(1 \leq o_i \leq n3, t = \sum_{u=1}^{k-1} p_u)$

Operation $o_i$ is not able to start before all previous operations $\{o_1, o_2, \cdots, o_{i-1}\}$ end. It required at least $t = \sum_{u=1}^{j-1} p_u$. That is, $o_i$ starts at time $t$ or later.

5.$eb_{o_i,t}$ $(1 \leq o_i \leq n3, t = L - \sum_{u=k+1}^{n3} p_u)$

In order to complete all the jobs by $L$, it is necessary for Operation $o_i$ to end by time $t$ or before, $t = \sum_{u=x+1}^{n3} p_u$.

In addition, it is also necessary to add a collection of "coherence conditions" on the introduced variables. In all conditions below, $o_i$ and $o_j$ are quantified over all relevant operations and $t$ is quantified over all relevant times.

3. $sa_{o_i,t} \rightarrow sa_{o_i,t-1}(1 \leq o_i \leq n3, \sum_{u=1}^{k-1} p_u \leq t \leq L)$

It states that operation $o_i$ starts at or after ***t***, it starts at or after ***t*** -1. In particular, the starting time of ***t*** is not 1, but $\sum_{u=1}^{k-1} p_u$. That is because the ready times can be determined,$sa_{o_i,t}(t < \sum_{u=1}^{k-1} p_u)$ will be always true, so they can be eliminated. This will reduce the number of clauses, thereby increasing the efficiency of the solver.

4.$eb_{o_i,t} \rightarrow eb_{o_i,t+1}(1 \leq o_i \leq n3, 1 \leq t \leq L - \sum_{u=i+1}^{n3} p_u)$

This ensures that if $o_i$ ends by ***t***, then it ends by ***t*** +1. In particular, the ending time of ***t*** is not *L*, but $L - \sum_{u=t+1}^{n3} p_u$. That is because the deadlines of each operation can be determined, If these times are known, the clauses in the codification that contain $eb_{o_i,t}$ will be always true, so they can be eliminated. This will reduce the number of clauses, thereby increasing the efficiency of the solver.

5. $sa_{o_i,t} \rightarrow eb_{o_i,t+p_i-1}(1 \leq o_i \leq n3, 1 \leq t \leq L - p_i + 1)$

If $o_i$ starts at or after time ***t***, then it cannot end before time $t + p_i - 1$. In particular, the ending time of ***t*** is not *L*, but $L - p_i + 1$.

6. $sa_{o_i,t} \wedge pr_{o_i,o_j} \rightarrow sa_{o_j,t+p_i}(1 \leq t \leq L - p_i + 1, pr_{o_i,o_j} \subseteq U)$

If $o_i$ starts at or after ***t*** and $o_j$ follows $o_i$, then $o_j$ can't start until $o_i$ is finished. In particular, $pr_{o_i,o_j} \subseteq U$ is key. It is because it is focus of the different operating to establish the relationship, which is a strong connection. It produces clauses that accounts for the majority of all the clauses. After a lot of experiments we found that the $pr_{o_i,o_j}$ is only related to those $pr_{o_i,o_j}$ which appear in the constraint conditions.

## Data for experiments

We need to convert the SAT code to the CNF formula and represent them in numbers. It is important that variables need to be represented by continuous numbers. Otherwise, the number of variables will increase when solver deal with them, thus affecting the efficiency of problem solving

According to the previously proposed method, we tried to solve two open JSSPs: Ft20 and ABZ9. Ft10 consists of 10 jobs and 10 machines. Each job consists of 10 operations, and Ft20 consists of 20 jobs and 5 machines, each job consists of 5 operations. ABZ9 consists of 20 jobs and 15 machines, and each job consists of 15 operations.

All experiments were conducted on the personal computer (CPU: Intel 2.60GHz, Memory: 4.00GB, OS: Ubuntu 14.04.5). We use the SAT solver MiniSAT2.2.0 to solve these problems.

In order to quickly get the critical value, we take the following method.

1. We estimate or use other fast algorithms for a smaller value *N1*and a larger one *N2*, and calculate it with Minisat.
2. If the result of *N1* is no, the result of *N2* is yes, then (*N1+N2)/2* is assigned to N3, and calculate it with Minisat.
3. If the result of *N3* is yes, repeat step 2.
4. If the result of *N3* is no and the result of *N3*+1 is no also, then *N2*-1 is assigned to N3, and calculate it with Minisat.
5. If the result of *N3* is no and the result of *N3*+1 is yes, then *N3*+1 is the optimized result.

## Solving Ft20

TABLE I. RESULTS OF FT20

| N | Number | | CPU (secs) | Satisfiable |
|---|---|---|---|---|
| | *Variables* | *Clauses* | | |
| 1095 | 220980 | 1922795 | 97826.800 | no |
| 1096 | 221180 | 1925055 | 18.520 | yes |
| 1097 | 221380 | 1927315 | 19.404 | yes |
| 1099 | 221780 | 1931835 | 17.840 | yes |
| 1100 | 221980 | 1934095 | 15.176 | yes |
| 1115 | 224980 | 1967995 | 12.712 | yes |
| 1125 | 226980 | 1990595 | 15.472 | yes |
| 1150 | 231980 | 2047095 | 14.000 | yes |
| 1160 | 233980 | 2069695 | 13.824 | yes |
| 1165 | 234980 | 2080995 | 13.796 | yes |
| 1200 | 241980 | 2160095 | 21.568 | yes |

From Table I, we can see: the optimized of the Ft20 problem is 1096. But its *L* is 1096 plus $p_i$ (the last task of the processing time), isn't 1096.

## Solving ABZ9

In addition, we succeeded to solve the problem of ABZ9. Table II shows the experimental results. The first columns show the size of the SAT instance. The second shows the number of propositional variables and the third shows the number of clauses in the corresponding SAT instance. As *N* increases, the size of SAT instances increases linearly, while the CPU time increases exponentially. This reflects NP-hardness of JSSP.

The only exception is the result of *N*=678, which is much easier than *N*=677. The main reason is that *N*=678 is satisfiable while *N*=677 is unsatisfiable. We finish solving *N*=678 when a truth assignment satisfying *N*=678 is found. Therefore, we should examine the whole search.

TABLE II. RESULTS OF ABZ9

| N | Number | | CPU (secs) | Satisfiable |
|---|---|---|---|---|
| | Variables | Clauses | | |
| 661 | 402580 | 2026319 | 7463.37 | no |
| 665 | 404980 | 2056717 | 39210.50 | no |
| 666 | 405580 | 2064404 | 50485.40 | no |
| 669 | 407380 | 2086838 | 183690.10 | no |
| 670 | 407980 | 2094796 | 27216.70 | yes |
| 678 | 412780 | 2158273 | 293.05 | yes |

ABZ9 problem solving is difficult, which was deducted on the cluster machine usually [3]. This paper greatly improves the efficiency of the operation by optimizing the coding method and research method. In table III, the comparison of the number of clauses produced is shown.

TABLE III. THE COMPARISON OF NUMBER OF CLAUSES PRODUCED

| **N** | **Number Clauses** | | **Code reduction** | **reduction** |
|---|---|---|---|---|
| | ***After optimization*** | ***before optimization*** | | |
| 661 | 2026319 | 4402236 | 2375917 | 53.97% |
| 665 | 2056717 | 4429756 | 2373039 | 53.57% |
| 666 | 2064404 | 4436636 | 2372232 | 53.47% |
| 669 | 2086838 | 4457276 | 2370438 | 53.18% |
| 670 | 2094796 | 4464156 | 2369360 | 53.08% |
| 678 | 2158273 | 4519196 | 2360923 | 52.24% |

Use this encoding method, we get the optimal solution with just 308359.122s CPU time(*7463.37 + 39210.5 + 50485.4 + 183690.1 + 27216.7 + 293.052 = 308359.122*), while the result of the previous study is 1187257s CPU time [3] (*6595 + 8946 + 10363 + 12673 + 14279 + 16616 + 21905 + 26847 + 39692 + 42939 + 64439 + 104448 + 119798 + 132638 + 240493 + 268354 + 45702 + 10530 = 1187257*). At the same time, we deduct the code processing environment requirements from the cluster machine into the personal computer.

## Conclusions

Compared with the traditional inexact solution algorithm, the SAT algorithm can find the optimal solution, which will play an extremely important role in improving the production efficiency. It can not only be used as a solution to the static scheduling problems, but also as the solution of the non-optimal solution performance comparison benchmark.

Although the SAT algorithm has been used to solve the Scheduling problem before, it takes a long time to get the solution because of the complexity of the problem. In this paper, based on the analysis of these solutions, we optimized the SAT coding. Experiments show that it is effective to optimize the coding. It not only reduces the time to get the solution, but also reduces the requirements for the computer.

Next, we will continue to optimize the coding and do some other jobs, trying to solve more complex problems, such as ABZ8, YN2, YN3, YN4, etc.

## Acknowledgements

This work was financially supported by the industry and information integration of the Sichuan economic and Information Committee([2016]118).

## References

[1] Lopes R V, Menascé D. A taxonomy of job scheduling on distributed computing systems. IEEE Transactions on Parallel and Distributed Systems, 27(12): 3412-3428, 2016.

[2] Hashem I A T, Anuar N B, Marjani M, et al. Multi-objective scheduling of MapReduce jobs in big data processing. Multimedia Tools & Applications, 2017(1):1-16.

[3] Liu L, Zhang M, Lin Y, et al. A survey on workflow management and scheduling in cloud computing//Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on. IEEE, 2014: 837-846.

[4] Rasooli A, Down D G. COSHH: A classification and optimization based scheduler for heterogeneous Hadoop systems. Future Generation Computer Systems, 36(3):1-15, 2014.

[5] Zhang Z, Chen Y, Tan Y, et al. Non-Crossover and Multi-Mutation Based Genetic Algorithm for Flexible Job-Shop Scheduling Problem. IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences, 99(10): 1856-1862, 2016

[6] Zhou S, Li X, Chen H, et al. Minimizing makespan in a no-wait flowshop with two batch processing machines using estimation of distribution algorithm. International Journal of Production Research, 54(16): 4919-4937, 2016.

[7] Tsai J T, Fang J C, Chou J H. Optimized task scheduling and resource allocation on cloud computing environment using improved differential evolution algorithm. Computers & Operations Research, 40(12): 3045-3055, 2013.

[8] Say B, Cire A A, Beck J C. Mathematical programming models for optimizing partial-order plan flexibility//ECAI. 2016: 1044-1052.

[9] Bahena B M, Cruz-Chávez M A, Díaz-Parra O, et al. Neighborhood Hybrid Structure for Minimum Spanning Tree Problem[C]//Electronics, Robotics and Automotive Mechanics Conference (CERMA), 2012 IEEE Ninth. IEEE, 2012: 191-196.

[10] Cruz-Chavez M A, Martinez-Oropeza A, Lopez R R. Relaxation of job shop scheduling problem using a bipartite graph//Electronics, Robotics and Automotive Mechanics Conference (CERMA), 2010. IEEE, 2010: 132-136.

[11] Frausto-Solis J, Martinez-Rios F. Golden annealing method for job shop scheduling problem//Proceedings of the 10th WSEAS international conference on Mathematical and computational methods in science and engineering. World Scientific and Engineering Academy and Society (WSEAS), 2008: 374-379.

[12] Crawford J M, Baker A B. Experimental results on the application of satisfiability algorithms to scheduling problems//AAAI. 1994, 2: 1092-1097.

[13] Koshimura M, Nabeshima H, Fujita H, et al. Solving open job-shop scheduling problems by SAT encoding. IEICE TRANSACTIONS on Information and Systems, 93(8): 2316-2318, 2010.

[14] Gorbenko A, Popov V. Task-resource scheduling problem. International Journal of Automation and Computing, 9(4): 429-441, 2012.

[15] Cruz-Chávez M A, Rivera-López R. A local search algorithm for a SAT representation of scheduling problems//International Conference on Computational Science and Its Applications. Springer, Berlin, Heidelberg, 2007: 697-709.

[16] Cruz-Chávez M A, Rivera-López R. A local search algorithm for a SAT representation of scheduling problems//International Conference on Computational Science and Its Applications. Springer, Berlin, Heidelberg, 2007: 697-709.

[17] Soeken M, De Micheli G, Mishchenko A. Busy man's synthesis: Combinational delay optimization with SAT//2017 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2017: 830-835.

[18] Hamadi Y, Jabbour S, Sais J. Control-based clause sharing in parallel SAT solving//Autonomous Search. Springer Berlin Heidelberg, 2011: 245-267.