MODELING AND SOLVING OPEN SHOP COOPERATIVE TASK SCHEDULING PROBLEMS BASED ON SATISFIABILITY MODULO THEORIES

Yosuke Kakiuchi, Kosuke Kato and Hideki Katagiri

Received January 24, 2017 ; revised March 11, 2017, March 17, 2017

Abstract. In the manufacturing process, it is necessary to schedule the processing order of machines in order to improve productivity. The scheduling can be modeled as an open shop scheduling problem. However, it has a constraint that multiple machines cannot process the same job at the same time, and is not suitable for cooperative works, such as software development or service engineering. In this paper, we propose a novel model of scheduling for cooperative works, based on an open shop scheduling problem. We also propose two formulations for the model to solve the problem with a Satisfiability Modulo Theories (SMT) solver. This paper aims to expand the range of the scheduling problem not only to manufacturing but also cooperative works.

1 Introduction

Scheduling is necessary in many situations in corporate activities, such as production, transportation and personnel allocation in projects. Particularly, in the manufacturing process, production scheduling is carried out to improve productivity. Production scheduling can be formulated into the scheduling of how to assign multiple machines to multiple jobs, and there are several types of problems, according to the assumption of the problem.

The problem assuming the fixed processing order of machines is called a *flow shop scheduling problem*. In this problem, all machines must be assigned in the same order in each job. It is suitable for flow works in which the order of process is not changed. Next, if the processing order is determined for each job, it is called a *job shop scheduling problem*. In a job shop scheduling problem, the processing order is different for each machine, but it is given in advance. Furthermore, the scheduling under the assumption that the order of processing is chosen arbitrarily and can be assigned in any order is called an *open shop scheduling problem*. This problem is a model for a job such that you can complete the process in any order.

Those three scheduling models have a constraint that multiple machines cannot process the same job at the same time. They are suitable for manufacturing, but are not suitable for software development or service engineering, because there is a possibility that one job can be jointly performed by multiple workers at the same time, such as jointly testing a software module or distributively performing a service on different sites.

In this paper, we propose a novel model of scheduling that relaxes the constraint rather than the open shop scheduling problem. In contrast to traditional scheduling models mentioned above, the proposed model allows one worker to be assigned to the same job multiple times, and allows multiple workers to be assigned to same job at the same time. Namely, the proposed model enables workers to do a task cooperatively. From the viewpoint of the flexibility and extensibility, we formulate the proposed model based on SMT (Satisfiability

2010 *Mathematics Subject Classification.* Primary: 90B35, 90C11; Secondary: 90B70, 68M20 .

Key words and phrases. Scheduling, open shop scheduling problem, satisfiability modulo theories, minimum makespan, mixed 0-1 integer programming .

	J_1	J_2	...	J_m
M_1	4	5	...	8
M_2	7	6	...	3
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
M_n	8	4	...	2

Table 1: An example of processing time t_{ij}

Modulo Theories) rather than mathematical programming. We also propose two formulations for the model to solve the problem with a SMT solver. SMT solvers cannot be only used in the field of semiconductor design automation, but also can be used in constraint satisfaction problem. We iteratively apply an SMT solver to obtain an (approximate) optimal solution.

This paper is organized as follows: Section 2 introduces the open shop scheduling problem, and refers to several preceding studies. In section 3, we propose the scheduling model and two formulations of the model. In section 4, we state the solution algorithm and the implementation, and we also show the experimental results. Finally, section 5 concludes this paper.

2 Preliminaries

2.1 Open Shop Scheduling Problem

The open shop scheduling problem is a scheduling to decide the way to assign n machines $\{M_1, M_2, \ldots, M_i, \ldots, M_n\}$ to m jobs $\{J_1, J_2, \ldots, J_j, \ldots, J_m\}$. Given t_{ij} units of processing time by the machine M_i for the job J_j, the open shop problem is a problem of searching a solution optimizing an objective. Table 1 shows an example of t_{ij}.

The open shop problem is presented in the literature [1], and also has been studied as some variants, changing some assumptions. Brucker *et al.* [5] presented the complexity for open shop scheduling with transportation delay. Masuda *et al.* [2] discussed the case where bi-criteria are considered for a two-machine open shop scheduling problem. They also developed a parametric linear programming problem for a two-machine open shop scheduling problem.

The open shop problem can be formulated as integer linear programming. Once it is formulated as an integer programming problem, it can be solved using a powerful solver. However, in the integer linear programming formulation, we cannot deal with propositional logic formulas directly. Hence, it is generally necessary to introduce an auxiliary variable and use indirect expression. In this paper, we adopt Satisfiability Modulo Theories (SMT) formulation. To be more specific, we formulate it as a constraint satisfaction problem based on SMT and obtain an optimal solution by solving it with the SMT solver iteratively. Since most SMT solvers can deal with both propositional logic and linear arithmetic, we shall describe the problem using both of them. In subsequent formulations, it is required that logical expressions become true to satisfy constraints.

One of the formulations for the open shop problem introduces the time s_{ij} and f_{ij} at which machine M_i starts and finishes job J_j respectively. The s_{ij} and f_{ij} are integer variables. If we assume that the objective function is minimizing the maximum of the completion time, the problem formulation is as follows:

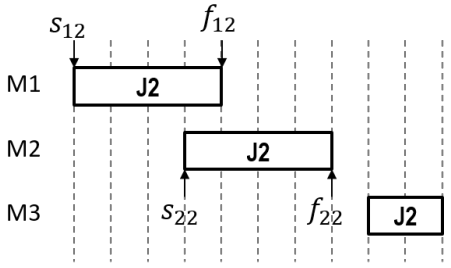

Figure 1: Example of violating expression (4)

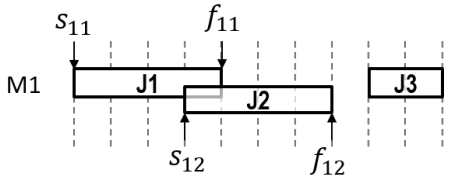

Figure 2: Example of violating expression (5)

$$\text{(1)} \qquad \text{minimize :} \qquad \max\{CT_i;\ 0 \le i \le n\}$$

$$\text{(2)} \qquad \text{subject to :} \qquad s_{ij} \ge 0,\ \forall i\ \forall j$$

$$\text{(3)} \qquad f_{ij} = s_{ij} + t_{ij},\ \forall i\ \forall j$$

$$\text{(4)} \qquad \bigwedge_{\forall i' \ne i} (f_{i'j} \le s_{ij}) \vee (f_{ij} \le s_{i'j}),\ \forall i\ \forall j$$

$$\text{(5)} \qquad \bigwedge_{\forall j' \ne j} (f_{ij'} \le s_{ij}) \vee (f_{ij} \le s_{ij'}),\ \forall i\ \forall j,$$

where CT_i means the completion time of M_i, that is, $CT_i = \max\{f_{ij};\ 1 \le j \le m\}$. $\vee$ is a logical disjunction operator whose result is *true* when at least one of the operands (conditions) is *true*. $\wedge$ (large $\bigwedge$) is a logical conjunction operator whose result is *true* when all of the operands (conditions) are *true*.

Each expression means that:

- Expression (2) guarantees that s_{ij} may not be assigned negative values.
- Expression (3) is a definition of f_{ij}.
- Expression (4) prohibits overlapping of multiple workers at the same time for the same job. Figure 1 shows an example of violating the expression (4) due to overlapping of J_2. We assume that $i' = 1$, $i = 2$ and $j = 2$. Since J_2 on M_1 and J_2 on M_2 are overlapped, the truth value of the condition $f_{12} \le s_{22}$ is $false$, and that of $f_{22} \le s_{12}$ is $false$. Thus, the truth value of expression (4) becomes $false$ in this situation. As in this example, it is $false$ if there is even one overlap in which the same job is assigned to different machines at the same time, otherwise it is *true*.
- Expression (5) prohibits overlapping of multiple jobs at the same time for the same workers. Figure 2 shows an example of violating the expression (5) due to overlapping on M_1. We assume that $i = 1$, $j' = 1$ and $j = 2$. Since J_1 and J_2 on M_1 are overlapped, the truth value of the condition $f_{11} \le s_{12}$ is $false$, and that of $f_{12} \le s_{11}$ is $false$. Thus, the truth value of the expression (5) becomes $false$ in this situation. As in this example, it is $false$ if there is even one overlap in which two or more jobs are assigned to the same machine at the same time, otherwise it is *true*.

Depending on the situation, we can adopt the maximum of the completion time (makespan) [7], the sum of the completion time [4, 6] or the sum of the tardiness [8, 9] as the objective function.

There are some previous studies with respect to the application of SMT solvers into scheduling problems. In one of the previous studies [10], the authors try to model and solve a resource-constrained project scheduling problem with an SMT solver. They demonstrated that an SMT solver is capable of solving a scheduling problem and have good performance. The differences are two points: we model the scheduling problem for co-operative tasks as two different formulations, while they modeled a resource-constrained project scheduling problem. Moreover, we describe the algorithm to optimize the solution for the problem and show the results, while they only outline the optimization.

3 A model of a scheduling problems for co-operative tasks

3.1 Problem

We propose a novel scheduling model for co-operative tasks, based on the open shop scheduling problem. The problem statement is the following.

(Scheduling model for co-operative tasks)
For each worker i and each job j, given efficiency E_{ij} and required work for completion C_j, determine the schedule which satisfies that for each job j, the total amount of work exceeds C_j.

$$\sum_{1 \leq i \leq n} t_{ij} \cdot E_{ij} \geq C_j, \ \forall j,$$

where t_{ij} is the total amount of time the worker i is assigned to the job j. The efficiency E_{ij} is an integer constant, and means how much work can be achieved per unit time. For example, $E_{23} = 6$ means that worker 2 achieves 6 amount of work for job 3 per unit time. If the worker 2 is assigned to job 3 during 4 unit times, she/he achieves $3 \times 4 = 12$ amount of work . The required work for completion C_j is an integer constant, and means the amount of work required for completion of job j.

This model allows one worker to be assigned to the same job multiple times, and allows multiple workers to be assigned to same job at the same time. Only thing that the model prohibit is to assign one worker to two or more jobs. However, the model is too loose and causes frequent job switching. This is inconsistent with actual personnel allocation. We therefore introduce the starting cost B_{ij} which must be spent when the worker is newly assigned to a job. The implementation of B_{ij} is different among formulations.

3.2 Formulation

We formulate the model proposed in the previous section in two ways using 0-1 decision variables and continuous ones. One has no restrictions on job switching, and the other has a restriction. Since we assume the usage of some SMT solver, we formulate them in the form of combining linear arithmetic and propositional logic.

3.2.1 Formulation 1

The first formulation allows unlimited job switching. Let E_{ij} be the efficiency of worker i for job j, C_j be required work for completion for job j, and T be a time frame. We introduce boolean decision variables x_{ij}^t as follows:

$$x_{ij}^t = \begin{cases} 1 & \text{(if worker } i \text{ is assigned to job } j \text{ at time } t\text{)} \\ 0 & \text{(otherwise)} \end{cases}$$

x_{ij}^t determines on/off of assignment at time t. The following constraints are defined from the problem statement:

$$\text{minimize :} \quad \max\{CT_i;\ 0 \le i \le n\} \tag{6}$$

$$\text{subject to :} \quad x_{ij}^t \in \{0,1\},\ \forall i\ \forall j\ \forall t \tag{7}$$

$$\sum_{1\le j\le m} x_{ij}^t \le 1,\ \forall i\ \forall t \tag{8}$$

$$\sum_{1\le i\le n}\sum_{1\le t\le T} f(i,j,t)\ \ge C_j,\ \forall j, \tag{9}$$

where CT_i is the completion time of worker i, and the function $f(i,j,t)$ is as follows:

$$f(i,j,t) = \begin{cases} x_{ij}^t \cdot (E_{ij} - B_{ij}) & (\ (t=0) \vee (x_{ij}^t \ne x_{ij}^{t-1})\ ) \\ x_{ij}^t \cdot E_{ij} & \text{(otherwise)}, \end{cases}$$

that is, when newly assigning to a job, calculate it with a work amount per unit time with penalty (i.e. B_{ij}), and when continuing to assign to a job, calculate it without penalty (i.e. E_{ij}). Short-circuit evaluation is applied to the above OR condition.

3.2.2 Formulation 2

In the second formulation, we limit the number of assignments to each job. It is assumed that a worker can be assigned to each job up to A times. Under this assumption, we introduce integer decision variables s_{ij}^a and f_{ij}^a which is a start time and an end time of the a-th assignment to job j, respectively.

The following constraints are defined from the problem statement:

$$\text{minimize :} \quad \max\{CT_i;\ 0 \le i \le n\} \tag{10}$$

$$\text{subject to :} \quad 0 \le s_{ij}^a \le T,\ \forall i\ \forall j\ \forall a \tag{11}$$

$$0 \le f_{ij}^a \le T,\ \forall i\ \forall j\ \forall a \tag{12}$$

$$s_{ij}^a \le f_{ij}^a,\ \forall i\ \forall j\ \forall a \tag{13}$$

$$\bigwedge_{(j',a')\ne(j,a)} (s_{ij'}^{a'} \le s_{ij}^a) \wedge (f_{ij'}^{a'} \le s_{ij}^a) \vee (f_{ij}^a \le s_{ij'}^{a'}) \wedge (f_{ij}^a \le f_{ij'}^{a'}),\ \forall i\ \forall j\ \forall a \tag{14}$$

$$\sum_{1\le i\le n}\left(\sum_{1\le a\le A} (f_{ij}^a - s_{ij}^a)\cdot E_{ij} - g(i,j,a)\right)\ \ge C_j, \forall j, \tag{15}$$

where CT_i is the completion time of worker i, and the function $g(i,j,a)$ is as follows:

$$g(i,j,a) = \begin{cases} 0 & (s_{ij}^a = f_{ij}^a) \\ B_{ij} & \text{(otherwise)} \end{cases}$$

3.2.3 A solution example and assignments to decision variables

Here, we show examples of formulas for $n = 3$ and $m = 3$. Table 2 shows efficiency E_{ij} and job switching penalty B_{ij}. Table 3 shows the required work for completion C_{ij}.

	J_1	J_2	J_3
M_1	3/2	6/3	7/3
M_2	2/2	2/1	9/3
M_3	4/3	3/2	7/3

Table 2: Example of efficiency E_{ij}/Bij

J_1	J_2	J_3
32	58	64

Table 3: Example of required work for completion C_j

Fig. 3 shows a line chart of a solution example such that all constarints are satisfied. As a result, $T = 10$. We calculate the amount of work for checking whether the constraint is satisfied:

Job 1: $(2 \cdot 2 - 2) + (4 \cdot 8 - 3) + (4 \cdot 1 - 3) = 2 + 29 + 1 = 32$
Job 2: $(7 \cdot 10 - 3) + (2 \cdot 1 - 1) = 67 + 2 = 69$
Job 3: $(9 \cdot 7 - 3) + (7 \cdot 1 - 3) = 60 + 4 = 64$

Formulation 1
In formulation 1, the assignments corresponding to Figure 3 are as follows:

$x^0_{12} = x^1_{12} = \cdots = x^9_{12} = 1,$
$x^0_{23} = x^1_{23} = \cdots = x^6_{23} = 1, x^7_{21} = x^8_{21} = 1, x^9_{22} = 1,$
$x^0_{31} = x^1_{31} = \cdots = x^7_{31} = 1, x^8_{33} = 1, x^9_{31} = 1,$
and all other $x^a_{ij} = 0$.

Formulation 2
In formulation 2, if $A = 3$, the assignments corresponding to Figure 3 are as follows:

$(s^1_{12}, f^1_{12}) = (0, 10)$
$(s^1_{23}, f^1_{23}) = (0, 7), (s^1_{21}, f^1_{21}) = (7, 9), (s^1_{22}, f^1_{22}) = (9, 10)$
$(s^1_{31}, f^1_{31}) = (0, 8), (s^1_{33}, f^1_{33}) = (8, 9), (s^2_{31}, f^2_{31}) = (9, 10)$
and all other $s^a_{ij} = f^a_{ij}$.

4 Experiment

4.1 Implementation

We used SMT (Satisfiability Modulo Theories) solvers to solve scheduling problems. The SMT solvers are usually used in the field of semiconductor design automation, such as logic synthesis, symbolic simulation, formal verification, and so on. They exploit both linear arithmetic and propositional logic, so that they has flexibility to describe various kinds of problems. In this paper, we use Z3 by Microsoft Research [11] as the SMT solver. It is the solver that shows high performance even in some bench-mark tests.

The SMT solver merely returns a feasible solution, that is, a solution which satisfies all of given constraints. Therefore, the time to obtain one of feasible solutions using the SMT solver tends to be less than that to obtain one of optimal solutions using a mathematical

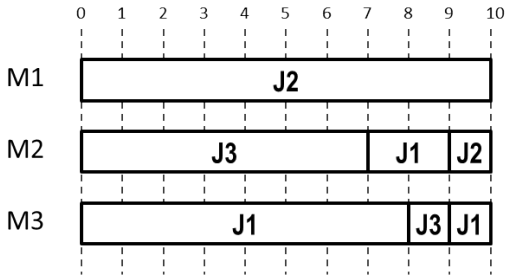

Figure 3: Example of scheduling for a solution

programming solver. On the other hand, the quality of the solution obtained by the SMT solver is not guaranteed in the sense of the objective function value (e.g. makespan). In order to obtain a feasible solution with guaranteed quality, we add a constraint that the objective function value is less than or equal to a given value t to the problem and optimize a solution by solving the problem with the SMT solver iteratively. To be more specific, the SMT solver first solves a constraint satisfaction problem with the additional constraint for a certain t. Depending on the result, after the value of t is increased or decreased, the solver solves it again. This increase (or decrease) value is reduced by half.

Algorithm 1 shows iterative application of the solver we propose.

The procedure optimizes the time frame t so that t becomes the minimum makespan. The function solve() executes the solver and returns the result as a return value. $UNSAT$ means that the solver can never discover a satisfying assignment. SAT means that the solver can discover one satisfying assignment.

The initial time frame T_i is enough small not to satisfy the constraints. Hence, the solver always returns $UNSAT$ on the first call of solve($P(t)$). It causes an increase of time frame t by the half of t_a. As long as the result is $UNSAT$, t will continue to increase, but eventually $P(t)$ will be satisfiable, and solve($P(t)$) will return SAT. If the result is SAT, the solver also returns an assignment which satisfies the constraints. The assignment is one of the available scheduling, but it is not necessarily optimal. We need to find the boundary between SAT and $UNSAT$. In other words, we have to find t where the result changes between SAT and $UNSAT$ when t is changed by 1.

4.2 Experimental Results

The above algorithm was implemented as a Python program. We also used Z3py, the Z3 Python user interface. The computation environment is the following; CPU: Intel Core i5 Processor 2.5 GHz, RAM: 8GB, OS: Windows 7 (64bit).

Table 4 shows experimental results on formulation 1 and 2 ($A = 1, 2$) for each problem. The first column indicates the number of workers and jobs. The second column indicates the minimum of makespan, or T in the formulations, among the solutions such that the constraint is satisfied. The other columns mean elapsed times in each formulation. "Total" is the sum of the time for all iterations. "Round" is the average time of an iteration. "$t.o.$" indicates that the experiment was canceled because the running time exceeded 10,000 seconds.

Overall, formulation 2 in $A = 1$ can obtain a solution in a shorter time than formulation 1. Because of the limitation to A (job switching), formulation 2 can obtain only an approximate solution, but it can obtain all the same results as an exact solution in the experiments. The problem instances in formulation 1 becomes difficult to solve for $m = 5$ or more, because formulation 1 always tries to find and return the optimal solution. Fig-

Algorithm 1: An algorithm of itrative constraint solving

Data:
$P(t)$: problem with time frame t.
T_i: an initial value of the time frame.
T_a: an increments of the time frame.
Result:
t: the minimum makespan.
the last a solution is the best scheduling.

```
t ← T_i
t_a ← T_a
prevResult ← UNSAT
while true do
    result ← solve(P(t))
    if (prevResult, result) == (UNSAT, UNSAT) then
        t ← t + t_a
    else if (prevResult, result) == (UNSAT, SAT) then
        if t_a = 1 then break ;
        else
            t ← t − t_a
    else if (prevResult, result) == (SAT, UNSAT) then
        if t_a = 1 then break ;
        else
            t ← t + t_a
    else if (prevResult, result) == (SAT, SAT) then
        t ← t − t_a
    t_a ← ⌈t_a/2⌉
    prevResult ← result
```

ure 4 shows elapsed time for each makespan in formulation 1. The solver terminated and returned SAT, when the makespan was set to 13 or less. On the other hand, it returned $UNSAT$, when the makespan was set to 14 or more. Moreover, when the makespan was set from 13 to 16, each elapsed time for constraint solving was more than 2,500 seconds. As this figure shows, in formulation 1, problem instances around the optimal makespan are hard to solve, because the constraint is much tighter. It is essential that computational cost of open shop scheduling problem grows exponentially. However, if we set up a time limit, an approximate solution can be obtained sooner. The dotted line in Figure 4 shows an example in the case where the time limit is given as 250 seconds. In this situation, the solution for the makespan of 19 can be obtained as approximate solution.

As a method to find an approximate solution of optimization based on SMT, MAX-SAT [12] is often used. In a MAX-SAT problem, a solver tries to satisfy as many logical clauses as possible. The solution which the solver has found cannot make the constraint true, but most of constraint is satisfied. However, in the scheduling problem in this paper, not the truth value of the constraint but the constant value in the constraint must be optimized. For this reason, it is difficult to apply MAX-SAT straightforwardly.

Rather, in the case of the scheduling problem assuming software development, each clause of constraint to be satisfied, such as makespan, human resource, or tardiness, has priority. If the solver cannot solve the constraint, it tries to solve more relaxed one after

(#Worker, #Job)	Min. make span	Form.1:Time(sec.)		Form.2:Time(sec.)			
				A = 1		A = 2	
		Total	Round	Total	Round	Total	Round
(2,2)	14	21.05	3.12	0.07	0.01	0.78	0.13
(2,3)	17	486.59	81.10	0.15	0.03	758.23	126.37
(2,4)	18	6089.47	1014.91	0.62	0.10	5324.71	887.45
(2,5)	22	*t.o.*	*t.o.*	11.23	1.87	*t.o.*	*t.o.*
(3,2)	9	9.44	1.57	0.09	0.02	63.12	10.52
(3,3)	10	149.92	24.99	0.22	0.04	272.95	45.49
(3,4)	10	505.06	84.18	0.6	0.1	343.70	57.28
(3,5)	14	*t.o.*	*t.o.*	265.99	44.33	*t.o.*	*t.o.*
(4,2)	6	5.05	0.84	0.1	0.02	333.66	55.61
(4,3)	7	43.88	8.78	0.73	0.15	*t.o.*	*t.o.*
(4,4)	8	864.57	172.91	43.04	8.61	*t.o.*	*t.o.*
(4,5)	10	8430.86	1405.14	5196.04	866.01	*t.o.*	*t.o.*
(5,2)	5	1.78	0.30	0.14	0.02	3010.57	501.76
(5,3)	6	76.17	12.70	16.48	2.75	*t.o.*	*t.o.*
(5,4)	6	442.59	73.77	26.37	4.40	*t.o.*	*t.o.*
(5,5)	8	4447.72	889.54	5974.65	1194.93	*t.o.*	*t.o.*

Table 4: Experimental Results

ignoring clauses of low priority. We guess that more practical scheduling results will be obtained by doing so.

5 Conclusion

In this paper, we presented a novel model for co-operative tasks based on open shop scheduling problem. We also proposed two types of formulations for the model: one allows unlimited job switching and another limits job switching. The model is suitable for scheduling in software development or service engineering, and contributes to automation of the scheduling in the fields. The experimental results show that it is costly to obtain exact solutions, but it requires less cost to find an approximate solution.

For further study, it is required to develop more efficient formulation and algorithms. Solvers are powerful, but in some cases it is better to use a customized algorithm. It is also important to find not optimal but good solution, because the solution is enough useful if it is used as a reference.

References

[1] Teofilo Gonzalez and Sartaj Sahni: "Open Shop Scheduling to Minimize Finish Time", Journal of the ACM Vol. 23 Issue 4, pp. 665-679, 1976.

[2] Teruo Masuda and Hiroaki Ishii: "Two machine open shop scheduling problem with bicriteria", Discrete Applied Mathematics Vol.52 Issue 3, pp. 253-259, 1994.

[3] Ellur Anand and Ramasamy Panneerselvam, "Literature Review of Open Shop Scheduling Problems", Intelligent Information Management vol.7, pp. 33-52, 2015.

[4] Peter Brucker, Johann Hurink, Bernd Jurisch, Birgit Wstmann: "A branch & bound algorithm for the open-shop problem", Discrete Applied Mathematics, Vol. 76, pp. 43-59, 1997.

[5] Peter Brucker, Sigrid Knust, T.C. Edwin Cheng and Natalia V. Shakhlevich: "Complexity Results for Flow-Shop and Open-Shop Scheduling Problems with Transportation Delays", Annals of Operations Research vol.129, pp.81-106, 2004.

[6] Eric Taillard: "Benchmarks for basic scheduling problems", European Journal of Operational Research, Vol.64, pp. 278-285, 1993.

[7] Moshe Dror: "Openshop Scheduling with Machine Dependent Processing Times", Discrete Applied Mathematics, Vol. 39, pp. 197-205, 1992.

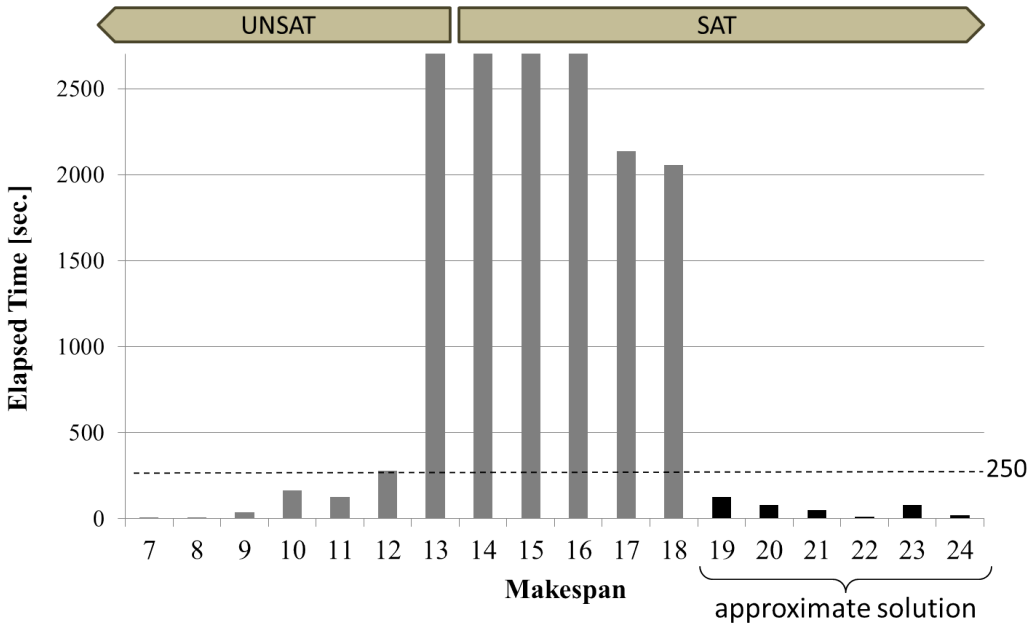

Figure 4: Formulation 1: result of for 3 workers and 5 jobs and approximate solutions

[8] Jorge M.S. Valentea, Rui A.F.S. Alvesb: "Heuristics for the single machine scheduling problem with quadratic earliness and tardiness penalties", Computers & Operations Research, Vol. 35, pp.36963713, 2008.

[9] Seyed Hossein Hashemi Doulabi: "A Mixed Integer Linear Formulation for the Open Shop Earliness-Tardiness Scheduling Problem", Applied Mathematical Sciences, Vol. 4, pp. 1703-1710, 2010.

[10] Carlos Anstegui, Miquel Bofill, Miquel Palah, Josep Suy and Mateu Villaret: "Satisfiability Modulo Theories: An Efficient Approach for the Resource-Constrained Project Scheduling Problem", in proceedings of 9th Symposium of Abstraction, Reformulation, and Approximation, 2011.

[11] Leonardo de Moura and Nikolaj Bjrner: "Z3: An efficient SMT solver", in prodeedings of 14th International Conference of Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008), pages 337-340, 2008.

[12] Pierre Hansen, Brigitte Jaumard: "Algorithms for the maximum satisfiability problem", Computing, vol. 44, no. 4, p.279-303, 1990.

Communicated by *Hiroaki Ishii*

Hideki Katagiri

Kanagawa University

3-27-1 Rokkakubashi, Yokohama, 221-8686 Japan

E-mail: katagiri@kanagawa-u.ac.jp

Yosuke Kakiuchi
Hiroshima Institute of Technology
2-1-1 Miyake, Saeki-ku, Hiroshima 731-5193 Japan
E-mail: y.kakiuchi.du@it-hiroshima.ac.jp

Kosuke Kato
Hiroshima Institute of Technology
2-1-1 Miyake, Saeki-ku, Hiroshima 731-5193 Japan
E-mail: k.katoh.me@it-hiroshima.ac.jp