Compact Representation of Time-Index Job Shop Problems Using a Bit-Vector Formulation

Compact Representation of Time-Index Job Shop Problems Using a Bit-Vector Formulation

Sabino Francesco Roselli[1] and Kristofer Bengtsson[1] and Knut Åkesson[1]

***Abstract*— The Job Shop Scheduling Problem (JSP) is a combinatorial optimization problem where jobs visit single-capacity machines while minimizing a cost function, typically the makespan. The problem can be extended to fit typical industrial scenarios such as flexible assembly shop floors or for coordinating fleets of automated vehicles. General purpose optimizers can handle extended versions of the problem that typically arise in industrial problems. Mixed Integer Linear Programming (MILP) solvers and recently optimizing Satisfiability Modulo Theory (SMT) solvers can be used as general solvers for JSP problems. There exist different formulations of JSP problems, among them the time-index (TI) model. The TI offers the advantage of providing strong lower bounds, though its drawback is the model size.**

In this paper we present a new formulation of the TI model suitable for optimizing SMT-solvers that support bit-vector theories. The new formulation is significantly more compact than the standard TI formulation and is thus reducing one of the major issues with the TI model.

We benchmark two different optimizing SMT solvers supporting bit-vector theories, comparing the standard formulation of the TI to the new formulation on a set of benchmark instances. The computational analysis shows that the new formulation outperforms the standard one, being up to twice faster and regardless of the solver employed; moreover the model generated with the new formulation is considerably smaller than with the standard formulation.

I. INTRODUCTION

The Job Shop Problem (JSP) is the assignment of jobs to resources, where each job is a sequence of operations, such that the makespan is minimized. Resources have single capacity, thus two operations cannot use the same resource at the same time. The problem is NP-hard and has been studied since the 60s, yet it is often emerging when developing automation systems and because of its complexity, it cannot be solved to optimality for large problem instances.

A thorough study on the subject is presented in [1], where the authors point out that, many different techniques have been applied to solve the problem: both approximation and exact methods. Whether the target is the true optimal or an approximate solution, it is important to develop general techniques that would be able to handle variations of the problem, since real-world scenarios typically involve additional requirements and as soon as a new constraint is added, tailor-made algorithms might no longer be feasible.

Thus general purpose algorithms like Mixed Integer Linear Program (MILP) solvers are often used in industrial applications since they can handle additional constraints without changing the main optimization algorithm. Note that new constraints could have a significant impact on the performance, but they will not disrupt the solution method.

MILP solvers can be used to provide approximate solutions by terminating the optimization procedure when the best incumbent (current best solution) is within a predefined gap from a lower bound to the problem, or after a time limit.

Another general purpose approach that can offer similar flexibility is optimizing Satisfiability Modulo Theory (SMT) solvers [2], [3]. SMT solvers support different theories, from reals, linear arithmetics, arrays to bit-vectors; these theories allow for a flexible modelling environment to easily instantiate quite complicated constraints.

In our previous works, [4] and [5], we showed that SMT solvers are a viable option to deal with the JSP both in its standard and flexible variants, since they could handle medium-large size problems (instances counting up to 15 concurrent jobs and 10 machines) in a relatively short time (time limit set to 1200 seconds). Our analysis showed that the SMT solver Z3 [6] outperformed a state-of-the-art MILP solver in terms of running time, by more than one order of magnitude for large instances.

We also implemented three model formulations of the JSP for SMT and benchmarked them over a set of instances generated according to predefined rules. One of the models we compared is known as the time-index model (TI) and was first presented by [7]; it is based on boolean (or binary) variables to represent the possibility for each job on each machine (operation) to start at a given time-step. Time is discretized and for each operation, one variable will evaluate to True to indicate at which step such operation is starting.

Though our comparison proved the *disjunctive* model [8] to have the best performance, the TI is still widely employed to tackle the problem, in academia, [9] and [10] as well as in industrial applications [11]. This is due to the strong lower bounds that LP relaxations of the model provide [12], since they can be used to implement branch and bound or list-scheduling algorithms. The downside of the TI is its size: the number of constraints required to build the model becomes prohibitive even for relatively small instances, making the model-generation a bottleneck.

In [13], different optimization problems are tackled using both boolean (BL) and bit-vector (BV) models and, while in general a bit-vector formulation results in a much more compact representation, for some problems, it also leads to increase in performance by more than one order of magnitude.

Given these premises, it is reasonable to assume that a bit-vector based formulation could provide a more compact model and outperform the standard TI model based on boolean (or binary) variables.

The contributions of this paper are: (i) comparison of

We gratefully acknowledge financial support from Chalmers AI Research Centre (CHAIR) and AB Volvo (Project ViMCoR).
[1]Department Electrical Engineering, Chalmers University of Technology, Göteborg, Sweden {rsabino, kristofer.bengtsson, knut}@chalmers.se

a new formulation of the TI model, based on bit-vectors, with the standard formulation over a set of benchmark and generated instances; the comparison is carried out both on the model generation phase (in terms of model size) and the running time to solve such models. (ii) Benchmark of two different state-of-the-art SMT solvers, namely Z3 and Opti-MathSAT [14], chosen because they have shown consistently good performance in the latest SMT competitions and they come with a built-in optimizing tool.

In the next section, a formal mathematical description of the problem is given; in sections III and IV the two model formulations are presented; in section V a complexity analysis over the two models is given; in section VI, the experimental methods are described and in the following section, results are discussed. Finally, conclusions are drawn in section VII.

II. Problem Formulation

The JSP problem consists of a set of n jobs $J = \{j_i\}_{i=1}^{n}$, where each job has its own processing order through a set of k machines, $M = \{m_i\}_{i=1}^{k}$. Also, let d_{mj} model the duration of the execution of the same operation. Operations are defined as the execution of a job on a certain machine and, as each job has to visit each machine, the total number of operations in the problem is nk. Each job will go through all machines sequentially. Let o_i^j model the index of the machine to be used for job j executing operation i in sequence. The index of the machines for each step in the job sequence is thus given by $(o_1^j, \ldots, o_i^j, \ldots, o_k^j)$.

Finding a solution to the job-shop scheduling problem means to assign operations to machines so that all jobs are completed. The constraints in this kind of problem are two:

- as there exist a sequence of operations for each job, operations belonging to the same job must be executed in the right order;
- operations requiring the same machine and belonging to different jobs cannot overlap in time.

Given these two constraints, the target is to find a feasible schedule such that the overall makespan is minimized. Other variants of the problem involve different cost functions such as tardiness or lateness, but in order to compare our results to the optimal values available in the literature we followed the standard of the makespan.

We are now to present the TI model formulation, first in its standard form and then implemented using BV. An important feature of TI models is that they require an upper bound of the makespan. Since time is discretized, the upper bound is necessary to define a sufficient number of variables to describe the behaviour of the system at each time-step. If the upper bound is not large enough, the model will yield an infeasible result. On the other hand, the larger the upper bound, the more variables and constraints are needed. A trivial upper bound is the sum of all operation durations but tighter bounds can be calculated quickly using heuristic algorithms. However, computing tight upperbounds is beyond the scope of this paper. In the following we will assume that H is a given upper bound for a problem instance.

III. Standard Time-Index Model

In this model the execution time is split into steps, whose length is the minimum time-step. For instance, if the duration of an operation is n time-steps, n steps will be taken since it starts and until it is completed. The time-steps will be $T = \{0, \ldots, H\}$. Let d_{mj} be a natural number that models the number of steps it takes for machine m to execute job j. The decision variables are $T_{\max}$ and s_{mjt}, where $T_{\max}$ is an integer variable and s_{mjt} are boolean variables that evaluate to true if job j starts on machine m at time t. The model formulation for minimizing the makespan is given by:

$$\text{minimize } T_{\max} \text{ subject to}$$

$$\bigvee_{t=0}^{H} s_{mjt} \qquad \forall m \in M, j \in J \quad (1)$$

$$s_{mjt} \rightarrow \bigwedge_{t' \in T \setminus \{t\},} \neg s_{mjt'} \qquad \forall m \in M, j \in J, t \in T \quad (2)$$

$$s_{mjt} \rightarrow \bigwedge_{t'=t}^{t+d_{mj}} \neg s_{mj't'} \qquad \forall j, j' \in J, j \leq j', m \in M, t \in T \setminus \{0\} \quad (3)$$

$$s_{mjt} \rightarrow T_{\max} \geq t + d_{mj} \qquad \forall m \in M, j \in J, t \in T \setminus \{0\} \quad (4)$$

$$x_{o_{i-1}^j jt} \rightarrow \bigwedge_{t'=0}^{t+d_{o_{i-1}^j j}} \neg x_{o_i^j jt'} \qquad \forall i = 2, \ldots, k, j \in J, t \in T \setminus \{0\} \quad (5)$$

Equations (1) and (2) ensure that start time for each operation occurs only once; equation (3) prevents other operations to start on a machine while it is already executing one; equation (4) defines the variable used in the objective function; equation (5) models precedence among the operations of a job: if the $(i-1)^{th}$ operation of job j starts at time-step t, the i^{th} operation of the same job cannot start before time-step $t + d_{o_{i-1}^j j}$.

IV. Time-Index Model with Bit-Vectors

This model formulation is based on the time-index model presented in the previous subsection; instead of having boolean variables for each time-step and operation, there is a fixed sized bit-vector for each operation, whose size is given by H.

A bit-vector of size *n* is an element $\vec{b} = (b_{n-1}, \ldots, b_0) \in \mathbb{B}^n$. The index *i* maps the $(i)^{th}$ component of the vector, i.e. $b[i] = b_i$. Conversion from (to) a integer number is defined by $int : \mathbb{B}^n \rightarrow Z$ $(bv : Z \rightarrow \mathbb{B}^n)$ with $Z = [-2^{n-1}, 2^{n-1}) \subset \mathbb{Z}$ and $int(\vec{b}) := -2^{n-1} + \sum_{i=0}^{n-2} b_i \cdot 2^i$ $(bv := int^{-1})$.

Constraints can be defined by using bit-vector operations as well as arithmetic and logic operations. Let $\vec{a}, \vec{b} \in \mathbb{B}^n$ be two-bit vectors. Then, the bit-vector operation $\circ \in \{\wedge, \vee, \cdots\}$ is defined by $\vec{a} \circ \vec{b} := (\vec{a}[0] \circ \vec{b}[0], \cdots, \vec{a}[n-1] \circ \vec{b}[n-1])$.

In the following model, some of the constraints are defined based on bit-vector manipulation formulas presented in [15].

The decision variables are defined as follows:

- $\vec{s}_{mj}$ is a bit-vector variable of size H that has exactly one bit set. The position of such bit defines the step at which job j starts on machine m;
- $\vec{w}_{mj}$ is a bit-vector variable of size H that has as many bits set as time-steps the job j takes to be executed on machine m. The rightmost bit in the trail corresponds to the bit set on the variable $\vec{s}_{mj}$;
- $\vec{e}_{mj}$ is a bit-vector variable of size H that has all bit sets from the time-step the operation is completed until the last position on the vector.
- $T_{\max}$ is a bit-vector variable of size H.

Also, $\vec{\underline{d}}_{mj}$ is a bit-vector constant of size H whose leftmost bits are set based on the duration of job j on machine m.

$$\text{maximize } T_{\max} \text{ subject to}$$

$$\vec{s}_{mj} \neq 0 \qquad \forall j \in J, m \in M \quad (6)$$

$$\vec{s}_{mj} \ \wedge \ (\vec{s}_{mj} - 1) = 0 \qquad \forall j \in J, m \in M \quad (7)$$

$$\vec{w}_{mj} = \bigvee_{i=0}^{d_{mj}} (\vec{s}_{mj} \gg i) \qquad \forall j \in J, m \in M \quad (8)$$

$$\vec{e}_{mj} = \neg\vec{w}_{mj} \ \wedge \ (\vec{w}_{mj} - 1) \qquad \forall j \in J, m \in M \quad (9)$$

$$\vec{s}_{jm} \ \wedge \ \vec{\underline{d}}_{mj} = 0 \qquad \forall j \in J, m \in M \quad (10)$$

$$\neg\vec{e}_{mo^j_{i-1}} \wedge \vec{s}_{mo^j_i} = 0 \qquad \forall j \in J, m \in M, i = 2 \dots k \quad (11)$$

$$\bigwedge_{\substack{j,j' \in J \\ j \leq j'}} (\vec{w}_{mj} \ \wedge \ \vec{w}_{mj'}) = 0 \qquad \forall m \in M \quad (12)$$

$$T_{\max} = \bigvee_{\substack{j \in J \\ m \in M}} \vec{e}_{mo^j_k} \quad (13)$$

Equation (6) makes sure that at least one bit is set and (7) makes sure that at most one bit is set for the bit-vector modeling the starting time for job j on machine m. Thus together they guarantee that each job will start exactly once on each machine. This is achieved by forcing the bit-vector to be a power of *two*; equations (8) and (9) are used to define the variables $\vec{w}_{mj}$ and $\vec{e}_{mj}$; equation (10) sets the latest start time for each operation: since a time horizon is given, operation cannot start too late, otherwise they will not be completed within the given number of steps; equation (11) sets the precedence constraint among operations belonging to the same job; equation (12) sets the objective function as the bit-wise operation $\vee$ among the $\vec{e}_{mj}$ operations occupying the last position in the sequence for each job: the larger the value of such vectors, the sooner the operation is completed, therefore the problem is a maximization one; finally equation (13) is taking care of non-overlapping constraint by setting the conjunction of the $\vec{w}_{mj}$ variables representing operations using the same machine equal to zero.

A. Example on the use of bit-vectors

In order to make the concept about fixed sized bit-vector variables and constants clearer, we provide a short example on how the values are assigned: Let the time horizon be set to 10 time-steps. This implies that the size of the vectors will be of 10 bits. Let's assume that for the operation of job *j* visiting machine *m* the starting time is set (by the solver) on the 5-th time-step and that such operation has a duration equal to three time-steps. Therefore, the variables for such operation will be set as follows:

$\vec{s}_{mj}$	0	0	0	0	1	0	0	0	0	0
$\vec{w}_{mj}$	0	0	0	0	1	1	1	0	0	0
$\vec{e}_{mj}$	0	0	0	0	0	0	0	1	1	1
$\vec{\underline{d}}_{mj}$	0	0	0	0	0	0	0	0	1	1

The variable $\vec{s}_{mj}$ has the 5th left most bit set, the variable $\vec{w}_{mj}$ has as many bits set as time-steps in the duration of the operation, starting from the 5th left most bit, the variable $\vec{e}_{mj}$ has all bits set form the time-step the operation is completed. Finally, the constant $\vec{\underline{d}}_{mj}$ has as many bits set as time-steps in the duration of the operation minus one, starting from the right most bit. This constant is required to set the constraint about the latest start of an operation. In the model, constraints (6) to (9) are needed to define the variables and constraint.

Here are given some examples of how constraints are enforced: In the following, the black *x* represent the possible assignments for the variables, while the red ones point out the forbidden assignments.

$\vec{s}_{mj}$	x	x	x	x	x	x	x	x	x	x
$\vec{\underline{d}}_{mj}$	0	0	0	0	0	0	0	0	1	1

Constraint (10) prevents an operation from starting too late by imposing the bit-wise operation $\wedge$ equal to zero between the variable $\vec{s}_{mj}$ and its duration constant $\vec{\underline{d}}_{mj}$.

$\vec{e}_{mo^j_{i-1}}$	0	0	0	0	0	0	0	1	1	1
$\neg\vec{e}_{mo^j_{i-1}}$	1	1	1	1	1	1	1	0	0	0
$\vec{s}_{mo^j_i}$	x	x	x	x	x	x	x	x	x	x

Constraint (11) sets the precedence constraints by allowing the following operation to start only after the previous one is completed. Assuming operation o^j_{i-1} is completed at time-step 7, operation o^j_i cannot start until that time-step.

$\vec{e}_{mo^j_k}$	0	0	0	0	0	0	0	1	1	1
$\vec{e}_{mo^{j'}_k}$	0	0	0	0	0	0	1	1	1	1
$\vec{e}_{mo^{j''}_k}$	0	0	0	0	1	1	1	1	1	1
$T_{\max}$	0	0	0	0	1	1	1	1	1	1

The objective function is set as the bit-disjunction of the variable $\vec{e}_{mj}$ for the last operation in each job. Maximizing $T_{\max}$ means having the latest operation completed as early as possible.

$\vec{w}_{mj}$	0	0	0	0	1	1	1	0	0	0
$\vec{w}_{mj'}$	x	x	x	x	x	x	x	x	x	x

Constraint (13) prevents operations employing the same machine to overlap, as shown in the example, where the red crosses define the forbidden execution times for the operation belonging to job *j'*

Finally, the result is presented as a bit-vector whose set bits represent the steps left before the time horizon is reached, since the last operation was completed. The integer value that represents the makespan to the JSP is given by the bit-vector $T_{\max}$ as

$$H - \text{number of bits set in } T_{\max}$$

Since the duration of an operation is positive we can conclude that not all operations could be finished at time 0, meaning that the left-most bit in T_{max} has to be 0. From how T_{max} is defined we know that it will have a sequence of zeros followed by a sequence of ones. Thus, with an increasing number of ones the signed/unsigned interpretation of T_{max} will result in a larger value. To minimize the makespan we will thus maximize T_{max}.

B. Bit-vector manipulation

In general, one must be careful when performing operations on bit-vectors since some of them will produce a different result, depending on whether the bit-vector is signed or not. The *right-shift* operation, for instance, comes in two different versions; other operators are unaffected by the interpretation: the bit-wise operators.

In this work, the actual value of the number represented by a bit-vector is not of interest, since we are only interested in the bit-patterns (the sequence of zeros and ones in a bit-vector) to represent time-steps: if the left-most bit of a bit-vector of size *four* is set, we are not interested in its value in decimals (it would be 8 for a signed and -8 for unsigned), it tells us that something is happening at time-step 0. Only in constraint (13) we are interested in the actual value of T_{max}, since this is the value that is maximized. For T_{max} it would make a difference to have signed or unsigned bit-vectors only if we were to set its leftmost bit. This, in turn, could only happen if it was possible to complete all jobs at time-step zero, which is by definition impossible, since operations have a duration larger than 0. Hence, the constraint is valid.

Also, the difference between signed and unsigned bit-vectors lies in the interpretation of bit-patterns and the tricks used to manipulate the bit-vectors are designed to set and unset bits regardless of the interpretation. In fact, they involve bit-wise operators such as $\wedge$, $\vee$ and $\neg$ with the exception of constraints (7) and (9): in both cases a subtraction is performed. However, subtracting a bit-vector means to add its negation plus *one*; since addition and negation both work independently of the interpretation, there is no risk of producing invalid results, as long as the operands involved in the subtraction are positive. This is always the case, since one of them is the value *one* and the other, w_{mj}, is inferred from s_{mj}, which is always positive because of constraints (6) and (7).

V. Models Size

The formulations presented in the previous sections, though similar in many aspects, lead to a significant difference in the model size. The reason for such difference lies in the way the time horizon, H, is handled by the two models: in the BV model the time horizon is used only to define the size of the bit-vector variables; therefore, equations (6)-(10) in the BV model only generates nk constraints (n jobs, k machines), while equation (11) generates $nk(k-1)$ constraints, equation (12) generates k constraints and (13) generates one. Also, the length of the constraints, in terms of number of clauses for each constraint, is short: only one clause for constraints (6), (7), (9), (10) and (11). For the constraints expressed by equation (8), the number of clauses depends of the duration of job j visiting machine m, for equation (12), the length is $n^2/2$, and for equation (13), the length is nk.

On the other hand, in the BL model, there is one variable for each job, machine and time-step. Therefore constraints are dependent on the time horizon as well, i.e. equation (1) generates nk constraints, each of length H. Equation (2) generates nkH constraints each of length H, since for each machine *m*, job *j* and time-step *t*, it is necessary to iterate the $\wedge$ connective over all time-steps but *t*. Equation (3) generates the largest amount of constraints, since it iterates over any two jobs *j* and *j'*, for each machine *m* and time-step *t*. Also, for each value of these indexes, the $\wedge$ connector has to be iterated for as many times as the duration of operation *j* on machine *m*, leading to $n^2/2 \cdot kH$ constraints each of length d_{mj}. Equation (4) generates nkH constraints of length one and, finally, equation (5) generates one constraint for each machine, job and time-step, and within each constraint an additional iteration of the $\wedge$ connective for as many times as the duration of operation *j* on machine *m* is required, leading to nkH constraints, each of length d_{mj}.

Given these premises, we can infer that:

BL model size: Given a JSP, with n jobs and k machines with time horizon H, the number of variables for the BL model is nkH, the number of constraints is proportional to $O(n^2kH)$ and their length in number of clauses is $O(H)$.

BV model size: Given a JSP with n jobs and k machines, the number of decision variables for the BV model is $3nk$, and the size of each variable is H bits, thus the total size is proportional to nkH. The number of constraints is $O(nk^2)$ and their length in number of clauses is $O(n^2)$.

Note that H is typically much larger than k and n. Therefore the number of constraints, as well as their length is expected to be significantly larger for the BL model. The favorable size of BV models compared to BL models is confirmed empirically in the following section.

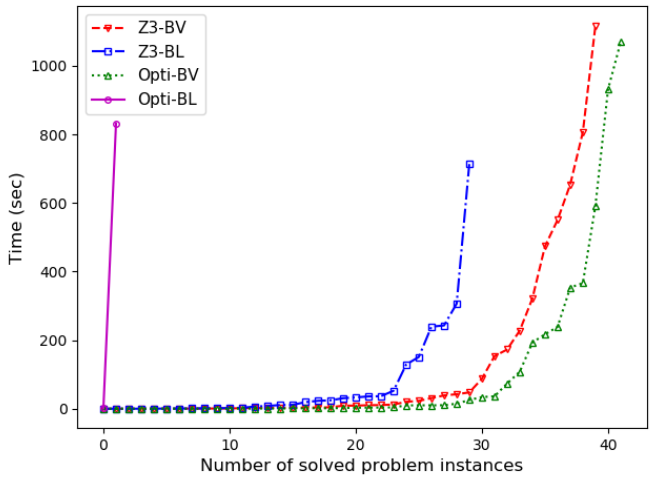

Fig. 1: Performance comparison between bit-vector (BV) and boolean (BL) time-index models over the generated instances using Z3 and OptiMathSAT. The maximum time allowed for each instance is 1200 seconds.

TABLE I: Comparison of models implemented using Z3 and OptiMathSAT. The time showed in the table is the geometric mean calculated over all the instances belonging to the category they refer to. For each class the number of solved instances (out of the total number of instances belonging to such class) is given. The symbol '-' means that no instance has been solved. The model size is also reported in Megabytes.

Problems	Model Size		Z3-BV		Z3-BL		Opti-BV		Opti-BL	
	BV	BL	Time	Opt	Time	Opt	Time	Opt	Time	Opt
3x3	0.009	0.243	0.13	5/5	0.13	5/5	0.25	5/5	32.3076	2/5
4x4	0.016	0.900	0.49	5/5	0.94	5/5	0.25	5/5	-	-
5x5	0.027	2.050	1.83	5/5	5.05	5/5	0.64	5/5	-	-
6x6	0.040	3.837	4.06	5/5	20.88	5/5	1.47	5/5	-	-
7x7	0.058	6.739	13.05	5/5	68.51	5/5	4.22	5/5	-	-
8x8	0.079	11.526	32.50	5/5	203.31	5/5	13.12	5/5	-	-
9x9	0.104	18.696	176.15	5/5	-	-	71.69	5/5	-	-
10x10	0.132	24.818	609.07	4/5	-	-	410.21	5/5	-	-
11x11	0.168	37.307	1117.60	1/5	-	-	740.85	3/5	-	-
12x12	0.205	48.927	-	-	-	-	-	-	-	-
13x13	0.249	69.687	-	-	-	-	-	-	-	-

VI. Computational Analysis

We evaluate the properties of the two models by generating problem instances using the Taillard instance generator specification [16]. In total 55 problems are generated from size 3x3 to 13x13 with 5 problems of each size.

A. Experimental setup

The solvers whose performance were compared are Z3-4.8.7 and OptiMathSAT-1.5.1. The time limit is 1200 seconds. Solvers are run in their default setting. All the experiments were performed on an *Intel Core i7 6700K, 4.0 GHZ, 32GB RAM* running *Ubuntu-18.04 LTS*.

Since finding a good upper bound for the model is beyond the scope of this paper, and the optimum is known for all instances evaluated, we used as a value for H the optimum increased by 10%.

Since both Z3 and OptiMathSAT can read input in the SMT standard language [17], it has been possible to translate the models into such language and then run the solvers directly from the terminal, to avoid delays due to the API's use. Also, this allows to run the solvers on exactly the same models, and to keep track of the models size for comparison. The implementation of the scheduler is available at [18].

B. Experimental Results

Table I summarizes the results of the computational analysis: instances are sorted by size (5 instances for each class) and for each different combination of model and solver, the number of solved instances is reported as well as the average time to find the optimum of the solved ones. We decided to employ the geometric mean to reduce the effect of outliers. The average model size for both the BL and BV model is also shown for each class.

The evaluation of the BV model implemented with Z3 showed that all instances could be solved within the time-limit up to size 9x9, while only 4 of size 10x10 and 1 of size 11x11 could be solved to optimality; no larger instance is solved within 1200 seconds. The performance of BV implemented with OptiMathSAT is significantly better, being OptiMathSAT roughly twice as fast as Z3 and able to solve all instances of size 10x10 and 3 of size 11x11. When it comes to the BL model, Z3 outperforms OptiMathSAT by more than one order of magnitude, being able to solve instances up to size 8x8 in a relatively short time: less than a second for size 3x3 and 4x4, 5 seconds for size 5x5 and respectively 20, 70 and 200 for the remaining sizes. On the other hand, the BL model implemented with OptiMathSAT is only able to solve 2 instances out of 55 (of size 3x3) and it still takes 30 seconds to do so.

When it comes to the model size, it turns out (as expected) that the BL model quickly scales up, going from 0.2 Megabytes for instances of size 3x3 to almost 70 MB for the larger ones. On the other hand, the BV model size is barely affected by the instance size, being still largely under 1MB for the larger instances. The time required to generated the model may be strictly dependent on the implementation, but it is still related to the model size, so the larger the model, the slower the generation time. With our implementation, the time to generate the BV model for the instances of size 13x13 was still below three seconds, while for the BL model it was around 600 seconds.

C. Results Discussion

The Computational Analysis proves the BV model to be faster than the BL model regardless of the solver it is implemented with, as shown in Figure 1; the best combination of solver-model is OptiMathSAT running the BV model, while the solver that showed the best performance when running the BL model was Z3. The reason for this behaviour lies not only in the efficiency of the underlying SAT engine within the SMT solver, but also in the way the particular theory the model belongs to [19]: some solvers simply *bit-blast* the model, meaning that they generate boolean variables to be able to handle it with Propositional Logic (eager approach), while others combine the SAT solver with a *Theorem Prover* to use specific Procedures to check feasibility (lazy approach). The latter method can, in some cases, save a lot of computational effort, increasing the efficiency of the overall approach. So, depending on the strategy employed by each solver (eager or lazy) and the efficiency of the procedure for the specific theory, one solver can be very good at solving one model, while being rather slow for another.

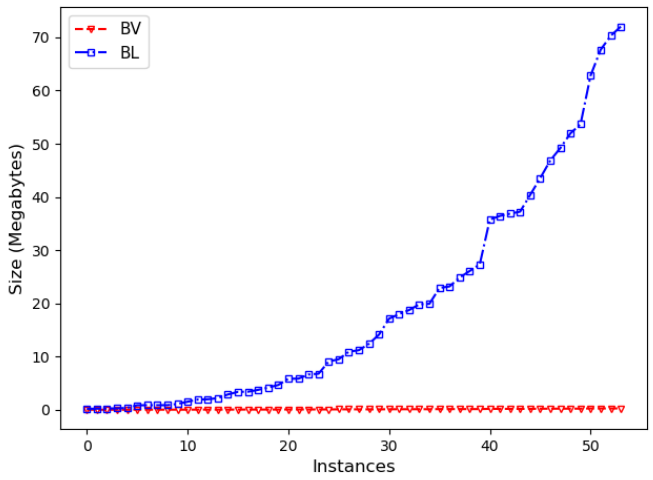

Fig. 2: Comparison between model sizes of the bit-vector and The boolean implementations of the time-index model, over the generated instances.

Another interesting result is the model size: the BV model proved to be extremely compact, increasing only linearly with the instance size (and with a very low coefficient), while the BL model's increase is roughly quadratic (see Figure 2). For other problem formulations, the model generation can usually be neglected, since it requires a very short time, compared to the solving time itself. But with time discretization, the number of variables is much higher, and the number of constraints generated out of them is even higher.

VII. Conclusion

In this paper we have presented a new approach to implement the time-index model for the Job Shop Problem using bit-vectors. We benchmarked two state-of-the-art SMT solvers over a set of instances generated according to a standard method and the resulting model turned out to be more efficient than the existing one based on boolean variables, regardless of the solver employed. We also showed that the more compact formulation leads to a drastic decrease in the model size, which in turn affects the model generation time (a bottleneck for the time-index model).

Since the time-index model is widely employed both in industry and academia to deal with the JSP, this contribution can improve the performance in many applications.

References

[1] I. A. Chaudhry and A. A. Khan, "A research survey: Review of flexible job shop scheduling techniques," *International Transactions in Operational Research*, vol. 23, no. 3, pp. 551–591, 2016.

[2] R. Sebastiani and P. Trentin, "Pushing the envelope of optimization modulo theories with linear-arithmetic cost functions," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2015, pp. 335–349.

[3] N. Bjørner, A.-D. Phan, and L. Fleckenstein, "νZ-an optimizing SMT solver," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2015, pp. 194–199.

[4] S. F. Roselli, K. Bengtsson, and K. Åkesson, "SMT solvers for job-shop scheduling problems: Models comparison and performance evaluation," in *2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2018, pp. 547–552.

[5] S. F. Roselli, K. Bengtsson, and K. Åkesson, "SMT solvers for flexible job-shop scheduling problems: A computational analysis," in *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2019.

[6] L. De Moura and N. Bjørner, "Z3: An efficient SMT solver," in *International conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2008, pp. 337–340.

[7] E. H. Bowman, "The schedule-sequencing problem," *Operations Research*, vol. 7, no. 5, pp. 621–624, 1959.

[8] A. S. Manne, "On the job-shop scheduling problem," *Operations Research*, vol. 8, no. 2, pp. 219–223, 1960.

[9] E. G. Birgin, P. Feofiloff, C. G. Fernandes, E. L. De Melo, M. T. Oshiro, and D. P. Ronconi, "A MILP model for an extended version of the flexible job shop problem," *Optimization Letters*, vol. 8, no. 4, pp. 1417–1431, 2014.

[10] L. Jin, Q. Tang, C. Zhang, X. Shao, and G. Tian, "More MILP models for integrated process planning and scheduling," *International Journal of Production Research*, vol. 54, no. 14, pp. 4387–4402, 2016.

[11] K. Thörnblad, A.-B. Strömberg, M. Patriksson, and T. Almgren, "Scheduling optimisation of a real flexible job shop including fixture availability and preventive maintenance," *European Journal of Industrial Engineering*, vol. 9, no. 1, pp. 126–145, 2015.

[12] J. Van den Akker, C. A. Hurkens, and M. W. Savelsbergh, "Time-indexed formulations for machine scheduling problems: Column generation," *INFORMS Journal on Computing*, vol. 12, no. 2, pp. 111–124, 2000.

[13] R. Wille, D. Große, M. Soeken, and R. Drechsler, "Using higher levels of abstraction for solving optimization problems by boolean satisfiability," in *2008 IEEE Computer Society Annual Symposium on VLSI*, IEEE, 2008, pp. 411–416.

[14] R. Sebastiani and P. Trentin, "OptiMathSAT: A tool for optimization modulo theories," *Journal of Automated Reasoning*, pp. 1–38, 2018.

[15] H. S. Warren, *Hacker's delight*. Pearson Education, 2013.

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

[17] C. Barrett, A. Stump, C. Tinelli, *et al.*, "The SMT-LIB standard: Version 2.0," in *Proceedings of the 8th international workshop on satisfiability modulo theories (Edinburgh, England)*, vol. 13, 2010, p. 14.

[18] *Benchmark code*, `https://github.com/sabinoroselli/Job_Shop.git`, Accessed: 2020-07-06.

[19] D. Kroening and O. Strichman, *Decision procedures - An Algorithmic Point of View*. Springer, 2016.