2018 IEEE 14th International Conference on Automation Science and Engineering (CASE)
Munich, Germany, August 20-24, 2018

SMT Solvers for Job-Shop Scheduling Problems: Models Comparison and Performance Evaluation

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

Abstract— **The optimal assignment of jobs to machines is a common problem when implementing automated production systems. A specific variant of this category is the job-shop scheduling problem (JSP) that is known to belong to the class of NP-hard problems. JSPs are typically either formulated as Mixed Integer Linear Programming (MILP) problems and solved by general-purpose-MILP solvers or approached using heuristic algorithms specifically designed for the purpose.**

During the last decade a new approach, satisfiability (SAT), led to develop solvers with incredible abilities in finding feasible solutions for hard combinatorial problems on Boolean variables. Moreover, an extension of SAT, Satisfability Modulo Theory (SMT), allows to formulate constraints involving linear operations over integers and reals and some SMT-solvers have been also extended with an optimizing tool.

Since the JSP is a well-known hard combinatorial problem, it is interesting to evaluate how SMT-solvers perform in solving it and how they compare to traditional MILP-solvers. We therefore evaluate state-of-the-art MILP and SMT solvers on benchmark JSP instances and find that general-purpose open-source SMT-solvers are competitive against commercial MILP-solvers.

I. INTRODUCTION

The Job-shop scheduling problem (JSP) is a well known problem within the Operation Scheduling Community, where the target is to allocate resources to operations while minimizing some cost function. In [1] a thorough study on the subject is presented, providing information about the evolution of techniques and algorithms to deal with the JSP whether the target was the true optimal or an approximation of it.

Industrial scheduling problems are typically extensions of the pure job-shop problem described above. Additional extensions are the possibility to have alternative resources that can process an operation, constraints on setup-time for resources, constraints on the idle-time between operations, etc. In many applications it might be desired to not minimize the make-span but instead make scheduling decisions based on given deadlines for operations and then minimize the tardiness or maximum lateness.

A recent study by [2] shows promising results in solving JSP problems by employing Constraint Programming (CP) to implement Local-Search (LS) algorithms. A similar solution has also been implemented by Beck et Al. [3], leading again to good results. Apparently the combination of CP and LS is a powerful instrument to takle the JSP, although also other techniques are used in practice. In fact, according to Beck himself, as shown in [4], both in industry and academia, Mixed Integer Linear Programming (MILP) is largely employed for the JSP. Based on this the authors did a benchmark on three popular MILP solvers, IBM ILOG CPLEX [5], Gurobi [6], and SCIP [7]. CPLEX and Gurobi are considered to be state-of-the-art commercial solvers [8], while SCIP is a fast non-commercial solver. In [4] the authors also compared alternative MILP problem formulations that have been proposed in the literature for the classical job-shop problem. The tools were evaluated on a large set of benchmark job-shop problems. The result of the benchmark is that for the tools evaluated, the performance of CPLEX and Gurobi were comparable and significantly better than SCIP and that the traditional disjunctive formulation of the job-shop problem was superior to both Time-Index and Rank-Based formulations.

While many industrial problems can be solved by general purpose MILP-solvers, the combinatorial complexity of the JSP implies that for sufficiently large problems it will not be possible to find an optimal solution within reasonable time. For such problems, specific-purpose methods have been developed: they often make use of heuristics as well as hybrid techniques including local search and genetic algorithms, e.g. [9] and [10]), that lead to *good enough* solutions in a reasonable amount of time [11].

An alternative technology to solve combinatorial problems have emerged within the community of formal verification of hardware and software: satisfability search (SAT) [12]. This approach consists of expressing the problem through Boolean variables in Conjunctive Normal Form (CNF) and determine whether there exists an assignment to these variables such that the formula evaluates to true. Today SAT-solvers can deal with large combinatorial problems by efficiently exploiting their structure and generate a valid model in a reasonable time, even when the problem happens to be NP-complete. Of course this does not mean that there are not NP-complete problems that will take exponential time also for SAT-solvers, but that many man-made models are no longer intractable.

On the other hand, Boolean logic is in many cases not expressive enough for representing many real-world problems where first-order logic is used together with integers and reals. To handle this type of models and at the same time take advantage of the very performant SAT-solvers, a new approach, called Satisfability Modulo Theory (SMT), [13], [14] has been developed. SMT-solvers implements special

We gratefully acknowledge financial support from the EUREKA ITEA3-projektet ENTOC (Label nr. 15015, Engineering Tool Chain for Efficient and Iterative Development of Smart Factories) and Vinnova.
[1]Department Electrical Engineering, Chalmers University of Technology, Göteborg, Sweden {rsabino, kristofer.bengtsson, knut}@chalmers.se

decision procedures, so called theories, to extend to the original Boolean satisifiability problem.

Both SAT and SMT are employed to prove satisfability of formulas; it is often the case that not only one but several satisfable solutions exist for a given formula and each of them has a cost referring to some parameter related to the problem itself. It is therefore possible to turn the satisfability problem into an optimization one by setting an objective function where one not only wants to find a satisfable solution, but the optimal one in respect of the cost we assign to the variables.

Ever since SAT and SMT solvers started spreading and being employed in real-world applications, users have built their own custom loops to achieve optimality, often based on heuristics tuned for their specific problems. Lately, some of these solvers have been extended by including optimizing tools that make use of state-of-the-art algorithms to deal with a list of different problems on SMT formulas with linear objective functions on Boolean, rational and integer domain (or a combination of them).

The performance of SMT-solvers are evaluated annually in a benchmark competition. The latest competition was the The 12th International Satisfiability Modulo Theories Competition (SMT-COMP 2017). The SMT-solvers compete in different categories in which Z3 [15], MathSAT5 [16], and CVC4 [17] have shown consistently good performance. Among these solvers Z3 and MathSat5 have versions that support optimization as well. The optimizing version of Z3 is described in [18] and is included in the latest version of Z3. MathSat5 optimizing tool is a special version of it named OptiMathSAT [19]. CVC4 does not have support for optimization and is thus excluded from this benchmark.

In order to have a valid comparison with the current technology, we also tested the models on a MILP solver. We chose Gurobi, since it is considered to be among the state-of-the-art solvers in such field.

The contributions in this paper are. (i) Adapting three existing formulations of the classical job-shop problem to make them suitable for SMT-solvers. (ii) Benchmarking two optimizing SMT-solvers on a set of benchmark job-shop problems.

In the next section the problem is described in detail, providing all necessary information to understand the models. In section three, we describe Z3 implementation for such models, we compare Z3 performance with Gurobi and OptiMathSAT using the Disjunctive model and we discuss the results. Finally we draw conclusions in section four.

II. Problem Description

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, $M = \{m_i\}_{i=1}^{k}$. 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 $n \cdot k$. 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 $(o_1^j, \ldots, o_i^j, \ldots, o_k^j)$. Also, let d_i^j model the duration of the execution of the same operation.

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 make-span is minimized. Finding an optimal schedule is an NP-complete problem, [20].

We now present the three model formulations, (i) the Disjunctive model, (ii) the Time-Index model, and (iii) the Rank-Based model for the classical JSP, based on [4]. They are adapted to fit the SMT language.

A. Time-Index Model

In this model the execution time is split into steps, whose length is the minimum time unit. For instance, if the duration of an operation is n-seconds (or minutes), n steps will be taken since it starts and until it is completed. In order to create a model with such feature, we need to have a guess of the overall execution time, in other words, an upper bound H. A trivial upper bound is the sum of all operations duration as if they were executed in a sequence (the worst case scenario). The greater the upper bound, the longer it takes to create the model and therefore the slower the overall execution. Nevertheless, as finding good upper bounds for such model is beyond the scope of this paper, the trivial one is used. In this model the decision variable is:

- s_{mjt} is a Boolean variable that is true if job j starts on machine m at time t.

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

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

$$s_{mjt} \rightarrow \bigwedge_{t'=0}^{t-1} \neg s_{mjt'} \wedge \bigwedge_{t'=t+1}^{H} \neg s_{mjt'} \qquad \forall t = 1, \ldots, H, m \in M, j \in J \quad (2)$$

$$s_{mjt} \rightarrow \bigwedge_{t'=t}^{t+p_{mj}} \neg s_{mj't'} \qquad \forall j, j' \in J, j \leq j', t = 1, \ldots, H, m \in M \quad (3)$$

$$s_{mjt} \rightarrow T_{max} \geq t + d_{mj} \qquad \forall m \in M, j \in J, t = 1, \ldots, H \quad (4)$$

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

Equation (1) and (2) are needed to make sure that one and only one operation is executed on a machine per each time step. Equation (3) allows each machine to execute only one operation at a time. Equation (4) sets the objective function as larger than operations completion times. Equation (5) takes care of the precedence constraint among operations of the same job.

B. Rank-Based Model

In this model the focus is on the machine side: each machine has as many positions as operations in a job or, in other words, a position is the cardinal step in the execution sequence of all jobs. Finding a schedule for this model means to find in which position a job is executed on a certain machine. The decision variables are defined as follows:

- x_{mjt} is a Boolean variable indicating whether job j is executed on machine m at the t-th position
- s_{mt} is an Integer variable representing the starting time of position t of machine m

minimize T_{max} subject to

$$s_{mt} \geq 0 \qquad \forall m \in M, t = 1 \dots k \quad (6)$$

$$x_{o_k^j jt} \rightarrow T_{max} \geq s_{o_k^j t} + d_{o_k^j j} \qquad \forall j \in J, t = 1 \dots k \quad (7)$$

$$x_{mjt} \rightarrow s_{mt} + d_{mj} \leq s_{m,t+1} \qquad \forall t = 1 \dots k, m \in M, j \in J \quad (8)$$

$$(x_{o_{i-1}^j jt_1} \wedge x_{o_i^j jt_2}) \rightarrow s_{o_{i-1}^j t_1} + d_{o_{i-1}^j j} \leq s_{o_i^j t_2} \qquad \forall t_1, t_2 = 1 \dots k, i = 2 \dots k, j \in J \quad (9)$$

$$\left(x_{mjt} \rightarrow \bigwedge_{t'=0}^{t-1} \neg x_{mjt'} \wedge \bigwedge_{t'=t+1}^{k} \neg x_{mjt'}\right) \wedge \bigvee_{k=0}^{k} x_{mjt} \qquad \forall t = 1 \dots k, m \in M, j \in J \quad (10)$$

$$\left(x_{mjt} \rightarrow \bigwedge_{j'=0}^{j-1} \neg x_{mj't} \wedge \bigwedge_{j'=j+1}^{n} \neg x_{mj't}\right) \wedge \bigvee_{t=0}^{k} x_{mj't} \qquad \forall t = 1 \dots k, m \in M, j \in J \quad (11)$$

Equation (6) restricts the start variable domain to the natural numbers. Equation (7) sets the objective function. Equation (8) ensures the precedence constraint among the operations executed on a machine, while equation (9) takes care of the operations precedence within the same job. Equation (10) ensures that each job can be assigned only once to a certain machine, while equation (11) states that a position can be assigned only to one job.

C. Disjunctive Model

The decision variables in this model are as follows:

- s_{mj} is an integer variable and models the start time of job j on machine m.

minimize T_{max} subject to

$$s_{mj} \geq 0 \qquad \forall j \in J, m \in M \quad (12)$$

$$T_{max} \geq s_{o_k^j, j} + d_{o_k^j, j} \qquad \forall j \in J \quad (13)$$

$$s_{o_i^j, j} \geq s_{o_{i-1}^j, j} + d_{o_{i-1}^j, j} \qquad \forall j \in J, i = 2, \dots, k \quad (14)$$

$$s_{mj} \geq s_{mj'} + d_{mj'} \vee s_{mj'} \geq s_{mj} + d_{mj} \qquad j, j' \in J, j \leq j', m \in M \quad (15)$$

In this model equation (12) restricts variables domain to be larger than or equal to zero, as they represent the start time of operations and thus they can not be negative. Equation (13) impose the objective function to be lager than or equal to the start time of the last operation of each job plus its duration. Equation (14) is about precedence constraints among the operations belonging to the same job: one can not start until the previous one is over. Equation (15) takes care of resource sharing by stating that two operations sharing the same resource cannot take place at the same time: either one starts once the other is over or the other way around.

III. Experiments

The solvers whose performance were compared are Z3-4.6.0, Gurobi-7.5.2 and OptiMathSAT-1.5.0. The time limit is 600 seconds. Solvers are run in their default setting. The instances used for the comparison are either generated through an instance generator or taken from benchmark sets. All the experiments were performed on an *Intel Core i7 6700K, 4.0 GHZ, 32GB RAM* running *Ubuntu-16.04*.

An instance is a matrix of integers where each row represents a job; for each row the odd elements represent the machine needed to execute the operation whose duration is pointed out by the next even element. The execution order is given by the position within the row. The instances used in the experimental phase are:

- Generated instances: instances generated according to Taillard instance generator[21] of small-medium size (from 3x3 to 9x9);

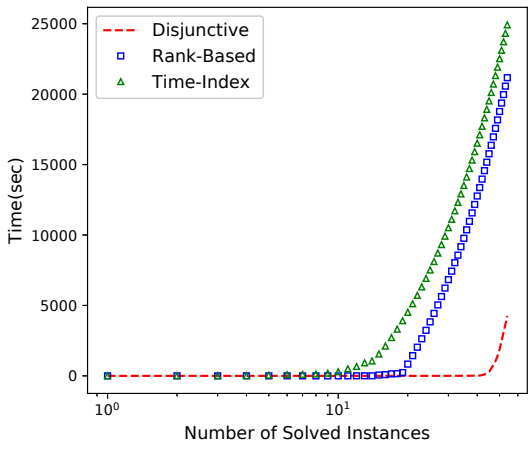

Fig. 1: Performance comparison between Disjunctive, Time-Index and Rank-Based models over the benchmark instances. The maximum time allowed for each instance is 600 seconds.

TABLE I: Comparison of models implemented using Z3. 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 symbol '*' means that the time limit for the model translation into SMT-lib2 has been exceeded.

Problems	Disjunctive		Rank-Based		Time-Index	
	Time	Opt	Time	Opt	Time	Opt
Generated Instances						
3x3	0.01	5/5	0.18	5/5	3.66	5/5
4x4	0.01	5/5	0.110	5/5	32.76	5/5
5x5	0.02	5/5	1.196	5/5	164.67	5/5
6x6	0.04	5/5	38.904	5/5	528.31	2/5
7x7	0.12	5/5	535.25	1/5	-	0/5
8x8	0.27	5/5	-	0/5	*	*
9x9	0.18	5/5	-	0/5	*	*
Applegate Instances						
10x10	7.28	10/10	-	0/5	*	*
Taillard Instances						
15x15	240.84	7/10	-	0/5	*	*

- Lawrence [22]: forty instances of increasing size from 10x5 to 30x10;
- Applegate and Cook [23]: ten instances of size 10x10;
- Storer [24]: five instances of size 20x10;
- Taillard [21]: ten instances of size 15x15 and ten of size 20x15.
- Fisher&Thompson [25]: one instance of size 20x5.

A. Models Comparison

In this phase the three models presented in the previous section are compared Z3 (Figure 1). We decided to employ the geometric mean to reduce the effect of outliers. The instance generation has been carried out by randomly assigning values belonging to the interval $[1, 20]$ to the operations. Five instances have been generated per each class going from the size 3x3 to 9x9.

The models have been created through the Python API for Z3 and then translated into the SMT-lib2 format [26] to be run directly and avoid possible delays due to the conversion while measuring the execution time. Since the Time-Index model scales up very bad in size an additional timeout has been set for the model translation into SMT format and the * symbol in the table denotes that such time limit has been exceeded. The results show that the Time-Index model can easily solve very small-size instances but it scales bad and provides no solution for problems larger than 6x6. The Rank-Based model is more efficient in terms of time but it can solve only some instances more than Time-Index.

The Disjunctive model, on the other hand, takes only few milliseconds to solve the smaller instances and it stays far below one second while dealing with instances up to 9x9 size. The average time increases by over forty times for solving Applegate instances due to some outliers that take over one minute but the result is still remarkable if compared to the other two models, even when it comes to Taillard Instances, although only seven out ten instances can be solved within ten minutes.

An additional test has been run on some hard-to-solve instances, to check how the best solution increases over time, as shown in Figure 2. Usually a solution close to the optimal is found quickly but then it is hard to improve it. This might be due to the solver getting stuck in some local optimum.

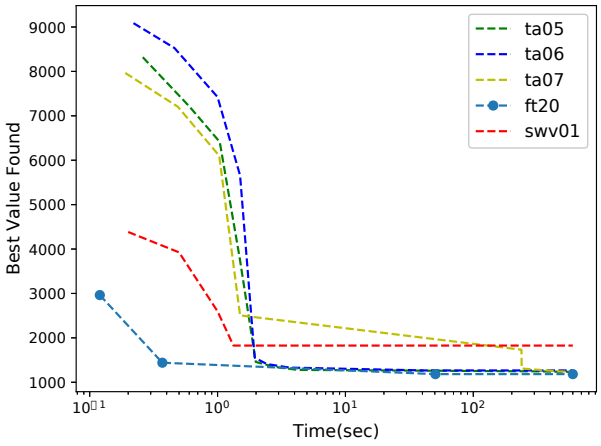

Fig. 2: Relation between time and best value found for benchmark instances 'ta05, 'ta06', 'ta07', 'ft20', 'swv01' running the Disjunctive model in Z3 during 3600 seconds.

B. Solvers Comparison

The second phase is about comparing different solvers using the Disjunctive model. When running the Disjunctive model on Applegate instances, OptiMathSAT could solve six out of ten of them and it was on average four times slower than Gurobi, which could solve seven. Z3 could solve all of them in less than eight seconds each. All the solvers could easily deal with the 10x5 instances from Lawrence and while Z3 solution was almost instantaneous Gurobi and OptiMathSAT had similar performance taking around twenty seconds to produce a solution. The only other class of instances Gurobi and OptiMathSAT were able to find a solution for is Lawrence 10x10 and also in this case Gurobi was quite faster than OptiMathSAT (around ten times) and both much slower than Z3, which was also able to solve some of Lawrence instances 15x10 in less than five minutes, 15x15 in less than three, and seven of the Taillard instances 15x15, as shown also in the previous section. No other solution was found within the given time.

TABLE II: Comparison of SMT and MILP solvers running the Disjunctive Model over the benchmark instances. 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.

	Z3		OptiMathSAT		Gurobi	
Problems	**Time**	**Opt**	**Time**	**Opt**	**Time**	**Opt**
Applegate Instances						
10*10	7.28	10/10	276.14	6/10	60.13	7/10
Lawrence Instances						
10x5	0.43	5/5	18.41	5/5	16.21	5/5
15x5	-	0/5	-	0/5	-	0/5
20x5	-	0/5	-	0/5	-	0/5
10x10	0.56	5/5	215.85	5/5	18.20	5/5
15x10	263.61	3/5	-	0/5	-	0/5
20x10	-	0/5	-	0/5	-	0/5
30x10	-	0/5	-	0/5	-	0/5
15x15	156.37	2/5	-	0/5	-	0/5
Storer Instances						
20x10	-	0/5	-	0/5	-	0/5
Taillard Instances						
15x15	240.84	7/10	-	0/10	-	0/10
20x15	-	0/10	-	0/10	-	0/10

C. Results Discussion

The results presented in the previous section show a big difference in performance between the two SMT solvers. When compared to Gurobi, Z3 is typically faster and can provide an optimal solution to larger instances within the given time limit. OptiMathSAT, although employing the same technology, is considerably slower than Z3 and, in most cases, also than Gurobi. Since Z3 and OptimathSAT have shown comparable performance in previous works [19], it might be the case that Z3 heuristic and linear optimization algorithms suits better the JSP problem than OptiMathSAT's. By having a look under the hood of νZ we found out it employs a portfolio of different approaches to solve optimization problems [18]. Among them are some very efficient algorithms to deal with linear arithmetic using Simplex over non-standard numbers to find unbounded objectives, as explained by Z3 developers in [27]. This method allows to find the solution in one call without need for iterating over potentially many of them.

Unlike νZ, that uses Z3 has a black box and it is built on top of it, OptiMathSAT has an inline architecture that calls the SMT solver only once and within it the SAT solver is then modified to handle the optimization. An insight about the optimizing algorithms running within the solver is given in Sebstiani's work [28]. Improved versions of the Branch&Bound algorithm are developed to exploit the features of MathSAT5 when dealing with linear algebra over Reals (LRA), integers (LIA) or a combination of both ($LRIA$).

IV. Conclusions

In this paper we have compared three models for JSP suitable for optimizing SMT-solvers. We also compared the disjunctive model formulation in Z3 and OptiMathSAT with Gurobi, a state-of-the-art commercial MILP solver. On the benchmark examples Z3 outperforms Gurobi, and Gurobi outperforms OptiMathSAT. The results are very interesting because SMT-solvers are general purpose solvers that can easily include additional constraints that are relevant in industrial applications making them an attractive choice for real applications. There exist options such as dedicated algorithms based on CP and local search that provide better performance; nevertheless the results shown in this paper classify SMT-solvers, and Z3 in particular as a good alternative, especially since it is available under an open-source license (MIT License) and hence it is possible to use it for commercial purpose. Today, the licensing costs for commercial MILP-solvers can be substantial, something that will restrict the numbers of applications where scheduling can be motivated. Since optimization was added very recently to SMT-solvers and the research on SMT-solvers is a very active field with rapid progress it is reasonable to expect that the performance of optimizing SMT-solvers will continue to improve.

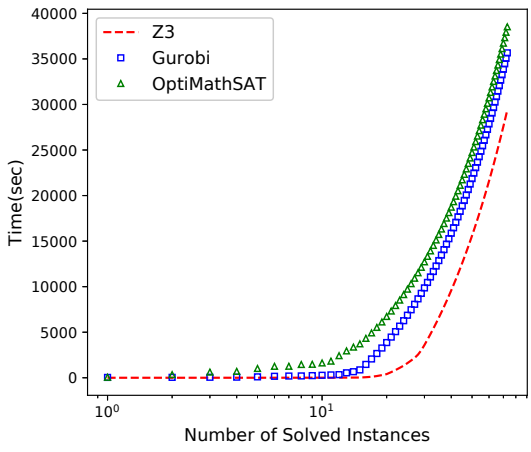

Fig. 3: Performance comparison between SMT solvers Z3 and OptiMathSAT and MILP solver Gurobi over the benchmarck instances. The maximum time allowed for each is 600 seconds.

REFERENCES

[1] A. S. Jain and S. Meeran, "Deterministic job-shop scheduling: Past, present and future," *European journal of operational research*, vol. 113, no. 2, pp. 390–434, 1999.

[2] Y. Nagata and I. Ono, "A guided local search with iterative ejections of bottleneck operations for the job shop scheduling problem," *Computers & Operations Research*, vol. 90, pp. 60–71, 2018.

[3] J. C. Beck, T. Feng, and J.-P. Watson, "Combining constraint programming and local search for job-shop scheduling," *INFORMS Journal on Computing*, vol. 23, no. 1, pp. 1–14, 2011.

[4] W.-Y. Ku and J. C. Beck, "Mixed integer programming models for job shop scheduling: A computational analysis," *Computers & Operations Research*, vol. 73, pp. 165–173, 2016.

[5] IBM. (2015). 12.6 users manual, [Online]. Available: `https://www.ibm.com/support/knowledgecenter/SSSA5P_12.6.2/ilog.odms.studio.help/pdf/usrcplex.pdf`.

[6] Gurobi. (2015). Gurobi optimizer reference manual, [Online]. Available: `http://www.gurobi.com`.

[7] A. Gleixner, L. Eifler, T. Gally, G. Gamrath, P. Gemander, R. L. Gottwald, G. Hendel, C. Hojny, T. Koch, M. Miltenberger, B. Müller, M. E. Pfetsch, C. Puchert, D. Rehfeldt, F. Schlösser, F. Serrano, Y. Shinano, J. M. Viernickel, S. Vigerske, D. Weninger, J. T. Witt, and J. Witzig, "The SCIP optimization suite 5.0," eng, ZIB, Takustr.7, 14195 Berlin, Tech. Rep. 17-61, 2017.

[8] "Miplib 2010: Mixed integer programming library version 5," English (US), *Mathematical Programming Computation*, vol. 3, no. 2, pp. 103–163, 2011.

[9] J. F. Gonçalves, J. J. de Magalhães Mendes, and M. G. Resende, "A hybrid genetic algorithm for the job shop scheduling problem," *European journal of operational research*, vol. 167, no. 1, pp. 77–95, 2005.

[10] E. Balas and A. Vazacopoulos, "Guided local search with shifting bottleneck for job shop scheduling," *Management science*, vol. 44, no. 2, pp. 262–275, 1998.

[11] A. S. Jain and S. Meeran, "A state-of-the-art review of job-shop scheduling techniques," Technical report, Department of Applied Physics, Electronic and Mechanical Engineering, University of Dundee, Dundee, Scotland, Tech. Rep., 1998.

[12] J. Franco and J. Martin, "A history of satisfiability.," *Handbook of satisfiability*, vol. 185, pp. 3–74, 2009.

[13] C. W. Barrett, R. Sebastiani, S. A. Seshia, C. Tinelli, *et al.*, "Satisfiability modulo theories," *Handbook of satisfiability*, vol. 185, pp. 825–885, 2009.

[14] L. De Moura and N. Bjørner, "Satisfiability modulo theories: Introduction and applications," *Commun. ACM*, vol. 54, no. 9, pp. 69–77, Sep. 2011.

[15] 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.

[16] A. Cimatti, A. Griggio, B. J. Schaafsma, and R. Sebastiani, "The MathSAT5 SMT solver.," in *TACAS*, N. Piterman and S. A. Smolka, Eds., ser. Lecture Notes in Computer Science, vol. 7795, Springer, 2013, pp. 93–107.

[17] C. Barrett, C. L. Conway, M. Deters, L. Hadarean, D. Jovanović, T. King, A. Reynolds, and C. Tinelli, "CVC4," in *Proceedings of the 23rd International Conference on Computer Aided Verification*, ser. CAV'11, Snowbird, UT: Springer-Verlag, 2011, pp. 171–177.

[18] 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.

[19] R. Sebastiani and P. Trentin, "OptiMathSAT: A tool for optimization modulo theories," in *International Conference on Computer Aided Verification*, Springer, 2015, pp. 447–454.

[20] M. R. Garey, D. S. Johnson, and R. Sethi, "The complexity of flowshop and jobshop scheduling," *Math. Oper. Res.*, vol. 1, no. 2, pp. 117–129, May 1976.

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

[22] S. Lawrence, "Resouce constrained project scheduling: An experimental investigation of heuristic scheduling techniques (supplement)," *Graduate School of Industrial Administration, Carnegie-Mellon University*, 1984.

[23] D. Applegate and W. Cook, "A computational study of the job-shop scheduling problem," *ORSA Journal on computing*, vol. 3, no. 2, pp. 149–156, 1991.

[24] R. H. Storer, S. D. Wu, and R. Vaccari, "New search spaces for sequencing problems with application to job shop scheduling," *Management science*, vol. 38, no. 10, pp. 1495–1509, 1992.

[25] H. Fisher, "Probabilistic learning combinations of local job-shop scheduling rules," *Industrial scheduling*, pp. 225–251, 1963.

[26] 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.

[27] N. Bjørner and A.-D. Phan, "νZ-maximal satisfaction with Z3.," *SCSS*, vol. 30, pp. 1–9, 2014.

[28] R. Sebastiani and P. Trentin, "Pushing the envelope of optimization modulo theories with linear-arithmetic cost functions," in *Proc. Int. Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS'15*, ser. LNCS, vol. 9035, Springer, 2015.