LETTER

Solving Open Job-Shop Scheduling Problems by SAT Encoding

Miyuki KOSHIMURA[†a)], Hidetomo NABESHIMA[††], Hiroshi FUJITA[†], *and* Ryuzo HASEGAWA[†], *Members*

SUMMARY This paper tries to solve open Job-Shop Scheduling Problems (JSSP) by translating them into Boolean Satisfiability Testing Problems (SAT). The encoding method is essentially the same as the one proposed by Crawford and Baker. The open problems are ABZ8, ABZ9, YN1, YN2, YN3, and YN4. We proved that the best known upper bounds 678 of ABZ9 and 884 of YN1 are indeed optimal. We also improved the upper bound of YN2 and lower bounds of ABZ8, YN2, YN3 and YN4.

key words: *combinatorial problem, scheduling, SAT encoding, job-shop scheduling, makespan*

1. Introduction

A job-shop scheduling problem (JSSP) is a combinatorial optimization problem. There are several reports solving JSSP [1]. They are divided into approximation methods and exact methods. This paper tries to solve open JSSPs by SAT encoding which is one of the exact methods.

In recent years, the propositional satisfiability (SAT) problem has been studied intensely. The state-of-the-art SAT solvers can solve a SAT problem which consists of millions of clauses in a few minutes. SAT is the prototypical NP-complete problem, and every instance π of a problem in NP can be translated into an instance of SAT. The birth of high-speed SAT solvers motivates the translation approach which encodes a certain problem π into a SAT problem and solves it by a fast SAT solver. In this paper, we call this problem solving technique the *SAT encoding approach*. It seems to be promising to solve open problems with the SAT encoding approach. There are several reports on succeeding in solving open problems [2], [3].

This paper tries to solve six open JSSPs with the SAT encoding approach. The six are ABZ8, ABZ9 [4], YN1, YN2, YN3, and YN4 [5]. ABZ8 and ABZ9 are also known as one of ten tough problems. The encoding is based on Crawford and Baker [6].

2. SAT Scheduling

In this section, we introduce a job shop scheduling problem (JSSP) which is a typical scheduling problem, and explain the SAT encoding approach proposed by Crawford and Baker [6]. We call this encoding Crawford encoding.

Manuscript received March 8, 2010.

[†]The authors are with Kyushu University, Fukuoka-shi, 819–0395 Japan.

[††]The author is with University of Yamanashi, Kofu-shi, 400–8511 Japan.

a) E-mail: koshi@inf.kyushu-u.ac.jp

DOI: 10.1587/transinf.E93.D.2316

We represent a SAT problem in conjunctive normal form (CNF). This form consists of the logical AND of one or more clauses, which in turn consist of the logical OR of one or more literals. A literal is a propositional variable or its complement. We represent a clause as a set of literals. The solution of a SAT problem is a truth assignment satisfying all clauses in the problem. When such an assignment exists, we call the SAT problem *satisfiable*, otherwise, *unsatisfiable*.

A JSSP consists of a set of n jobs $\{J_1, \cdots, J_n\}$ and a set of m machines $\{M_1, \cdots, M_m\}$. Each job J_l is a sequence of operations $\langle O^l_1, \cdots, O^l_{q_l}\rangle$. Each operation O^l_i requires the exclusive use of a machine $M_{O^l_i}$ $(1 \le O^l_i \le m)$ for an uninterrupted duration p^l_i, its processing time. A *schedule* is a set of start times for each operation O^l_i. The time required to complete all the jobs is called the *makespan* L. The objective of the JSSP is to determine the schedule which minimizes L.

We introduce three kinds of propositional variables:

- $pr^{l,k}_{i,j}$ means that O^l_i precedes O^k_j.
- $sa^l_{i,t}$ means that O^l_i starts at time t or later.
- $eb^l_{i,t}$ means that O^l_i ends by time t or before.

We translate the JSSP into a SAT problem by the following rules [7]. We make the Crawford encoding somewhat more precise. In this translation, we assume that the makespan is at most L.

1. O^l_i precedes O^l_{i+1}:
   $pr^{l,l}_{i,i+1}$ $(1 \le l \le n, 1 \le i < q_l)$
2. If O^l_i and O^k_j require the same machines, then O^l_i precedes O^k_j or O^k_j precedes O^l_i. That is, if $M_{O^l_i} = M_{O^k_j}$, then we add the clause:
   $pr^{l,k}_{i,j} \lor pr^{k,l}_{j,i}$ $(1 \le l < k \le n)$ $(1 \le i \le q_l, 1 \le j \le q_k)$
3. O^l_i is not able to start before all previous operations $O^l_1, \cdots,$ and O^l_{i-1} end. It requires at least $t = \sum_{u=1}^{i-1} p^l_u$. That is, O^l_i starts at time t or later:
   $$sa^l_{i,t}\ (1 \le l \le n, 1 \le i \le q_l, t = \sum_{u=1}^{i-1} p^l_u)$$
4. Reversely, in order to complete all the jobs by L, it is necessary for O^l_i to end by time t or before:
   $$eb^l_{i,t}\ (1 \le l \le n, 1 \le i \le q_l, t = L - \sum_{u=i+1}^{q_l} p^l_u)$$
5. If O^l_i starts at or after time t, it starts at or after $t-1$:
   $sa^l_{i,t} \rightarrow sa^l_{i,t-1}$ $(1 \le l \le n, 1 \le i \le q_l, 1 \le t \le L)$

6. If O_i^l ends by t, then it ends by $t+1$:
$eb_{i,t}^l \rightarrow eb_{i,t+1}^l$ $(1 \leq l \leq n, 1 \leq i \leq q_l, 0 \leq t < L)$
7. If O_i^l starts at or after time t, then it cannot end before time $t + p_i^l - 1$:
$$sa_{i,t}^l \rightarrow \neg eb_{i,t+p_i^l-1}^l \quad \begin{array}{l}(1 \leq l \leq n, 1 \leq i \leq q_l)\\(0 \leq t < L - p_i^l + 1)\end{array}$$
8. If O_i^l starts at or after t and O_j^k follows O_i^l, then O_j^k cannot start until O_i^l is finished. That is, for $pr_{i,j}^{l,k}$ added by rule 1 and 2, we add the clause:
$sa_{i,t}^l \wedge pr_{i,j}^{l,k} \rightarrow sa_{j,t+p_i^l}^k$ $(0 \leq t \leq L - p_i^l)$

Let S be the SAT problem translated by the above Crawford encoding. If S is satisfiable, then JSSP can complete all the jobs by the makespan L. Let M be the truth assignment satisfying S. The start time t_i^l of each operation O_i^l is given by extracting the last $sa_{i,t}^l$ which is assigned as true in M. More precisely, t_i^l is given by t which satisfies the following expression:

$$(M \models sa_{i,t}^l) \wedge (\neg \exists u > t\ (M \models sa_{i,u}^l))$$

where $M \models x$ indicates x is assigned true in M.

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

3. Experimental Results

We tried to solve six open JSSPs: ABZ8, ABZ9, YN1, YN2, YN3, and YN4. These problems are obtained from the OR-Library†. ABZn consists of 20 jobs and 15 machines, and YNn consists of 20 jobs and 20 machines. Table 1 shows the best known lower bounds (LB) and upper bounds (UB) indicated in the literature [1], [8], [9].

For each problem P, we generate $UB - LB + 2$ SAT instances S_{LB-1}^P, $S_{LB}^P, \ldots,$ and S_{UB}^P with the Crawford encoding and solve them with the SAT solver MiniSat [10]. Even though we know S_{LB-1}^P is unsatisfiable because of the lower bound LB and S_{UB}^P is satisfiable because of the upper bound UB, we solve them in order to verify the values. We use MiniSat version 2.

All experiments were conducted on the cluster machine A of CFV (Collaborative Facilities for Verification) SAT-SUKI in AIST (National Institute of Advanced Industrial Science and Technology). The cluster machine A consists of 112 nodes. Each node is a Xeon X5260 3.3 GHz machine with 8 GB memory on Linux 2.6.26-2-amd64.

Table 1 LB and UB.

Problem	LB	UB
ABZ8	646	665
ABZ9	662	678
YN1	846	884
YN2	870	907
YN3	840	892
YN4	920	968

3.1 Solving ABZ9 and YN1

We succeeded to solve ABZ9 and YN1. We showed that the optimum solutions of ABZ9 and YN1 are 678 and 884 respectively. This means that the best known upper bounds of ABZ9 and YN1 are really optimal.

Tables 2 and 3 show the experimental results of S_N^{ABZ9} $(N = 661, \ldots, 678)$ and S_N^{YN1} $(N = 875, \ldots, 884)$ respectively. The second and third columns show the size of the SAT instance. The second shows the number of propositional variables and the third shows the number of clauses in the corresponding SAT instance. As N increases, the size of SAT instances increases linearly, while the CPU time increases exponentially. This reflects NP-hardness of JSSP.

The only exception is the result of S_{678}^{ABZ9}, which is much easier than S_{677}^{ABZ9}. The main reason is that S_{678}^{ABZ9} is satisfiable while S_{677}^{ABZ9} is unsatisfiable. We finish solving S_{678}^{ABZ9} when a truth assignment satisfying S_{678}^{ABZ9} is found. Therefore, we need not examine the whole search space of S_{678}^{ABZ9} while we must examine the whole search space

Table 2 Results of S_N^{ABZ9} $(N = 661, \ldots, 678)$.

N	Number of Variables	Number of Clauses	CPU (secs)	Satisfiable
661	403180	4402236	6595	no
662	403780	4409116	8946	no
663	404380	4415996	10363	no
664	404980	4422876	12673	no
665	405580	4429756	14279	no
666	406180	4436636	16616	no
667	406780	4443516	21905	no
668	407380	4450396	26847	no
669	407980	4457276	39692	no
670	408580	4464156	42939	no
671	409180	4471036	64439	no
672	409780	4477916	104448	no
673	410380	4484796	119798	no
674	410980	4491676	132638	no
675	411580	4498556	240493	no
676	412180	4505436	268354	no
677	412780	4512316	457021	no
678	413380	4519196	10530	yes

Table 3 Results of S_N^{YN1} $(N = 875, \ldots, 884)$.

N	Number of Variables	Number of Clauses	CPU (secs)	Satisfiable
875	708780	7799938	184492	no
876	709580	7809118	296314	no
877	710380	7818298	373687	no
878	711180	7827478	549011	no
879	711980	7836658	620052	no
880	712780	7845838	1058190	no
881	713580	7855018	1166480	no
882	714380	7864198	2007540	no
883	715180	7873378	1912690	no
884	715980	7882558	3149770	yes

†http://people.brunel.ac.uk/˜mastjjb/jeb/info.html

Table 4 Improvement of LB and UB.

Prob-lem	SAT instance	CPU (secs)	Satis-fiable	New LB	New UB
ABZ8	S^{ABZ8}_{658}	7611330	no	659	
YN2	S^{YN2}_{892}	5739010	no	893	
YN2	S^{YN2}_{905}	803983	yes		905
YN3	S^{YN3}_{879}	3419330	no	880	
YN4	S^{YN4}_{947}	3554680	no	948	

Table 5 New LB and UB.

Problem	LB	UB
ABZ8	659[b]	665
ABZ9	678[a]	
YN1	884[a]	
YN2	893[b]	905[c]
YN3	880[b]	892
YN4	948[b]	968

[a] The optimum solution found by us.
[b] The new LB found by us.
[c] The new UB found by us.

of S^{ABZ9}_{677}.

3.2 Improving LB and UB

We succeeded to improve LB of ABZ8, YN2, YN3 and YN4, and UB of YN2. Table 4 shows experimental results on the SAT instances proof of which gives the best LB or UB. Solving more difficult problems, for example, S^{YN2}_{895}, S^{YN4}_{950}, etc., needs more than 8 GB memory which is beyond our machine. Table 5 summarises the new LB and UB.

4. Concluding Remarks

We tried to solve six open JSSPs with the SAT encoding approach. The new optimum solutions are 678 of ABZ9 and 884 of YN1. For all other problems new lower bounds are obtained; and for YN2 a new upper bound is obtained. These results show the effectiveness of the SAT encoding approach for solving open combinatorial problems.

Four problems (ABZ8, YN2, YN3, and YN4) remain open. Solving these problems needs more than several months and 8 GB memory if we follow the current approach. In order to tackle these hard problems, parallel computing seems to be a promising approach [11], [12]. Therefore, future work includes parallelisation of SAT solving. We will also tackle other open problems and investigate new SAT encoding methods.

Acknowledgements

We are most grateful to SATSUKI in AIST for letting us use the cluster machine. This work was supported by JSPS KAKENHI (20240003).

References

[1] A.S. Jain and S. Meeran, "Deterministic job-shop scheduling: Past, present and future," Eur. J. Oper. Res., vol.113, pp.390–434, 1999.
[2] H. Zhang, Combinatorial Designs by SAT Solvers, Handbook of Satisfiability, ch. 17, pp.533–568, IOS Press, 2009.
[3] N. Tamura, A. Taga, S. Kitagawa, and M. Banbara, "Compiling finite linear csp into sat," Constraints, vol.14, pp.254–272, 2009.
[4] J. Adams, E. Balas, and D. Zawack, "The shifting bottleneck procedure for job shop scheduling," Manage. Sci., vol.34, no.3, pp.391–401, 1988.
[5] T. Yamada and R. Nakano, "A genetic algorithm applicable to large-scale job-shop problems," Proc. PPSN'92: Second International Conference on Parallel Problem Solving from Nature, pp.281–290, 1992.
[6] J.M. Crawford and A.B. Baker, "Experimental results on the application of satisfiability algorithms to scheduling problems," Proc. AAAI-94: 12th National Conference on Artificial Intelligence, pp.1092–1097, 1994.
[7] H. Nabeshima, T. Soh, K. Inoue, and K. Iwanuma, "Lemma reusing for SAT based planning and scheduling," Proc. ICAPS 2006: 16th International Conference on Automated Planning and Scheduling, pp.103–112, 2006.
[8] W. Brinkkötter and P. Brucker, "Solving open benchmark instances for the job-shop problem by parallel head-tail adjustments," J. Scheduling, vol.4, pp.53–64, 2001.
[9] C.Y. Zhang, P. Li, Y. Rang, and Z. Guan, "A very fast ts/sa algorithm for the job shop scheduling problem," Computer & Operations Research, vol.35, pp.282–294, 2008.
[10] N. Eén and N. Sörensson, "Minisat: A sat solver with conflict-clause minimization," Proc. SAT-05: 8th International Conference on Theory and Applications of Satisfiability Testing, pp.502–518, 2005.
[11] K. Inoue, T. Soh, S. Ueda, Y. Sasaura, M. Banbara, and N. Tamura, "A competitive and cooperative approach to propositional satisfiability," Discrete Appl. Math., vol.154, pp.2291–2306, 2006.
[12] K. Ohmura and K. Ueda, "c-sat: A parallel sat solver for clusters," Proc. SAT 2009: 12th International Conference on Theory and Applications of Satisfiability Testing, LNCS 5584, pp.524–537, Springer, 2009.