A Scalable Method for a Particular Scheduling Problem Based on Boolean Satisfiability

Hai Lin[1,a]

[1]College of Software, Shenyang Normal University 253 Huanghe North Street, Shenyang, 110034, China

[a]jlu_linhai@163.com

Keywords: Boolean satisfiability, scheduling, SAT solving

Abstract. In recent years, people made a lot of progress for solving Boolean satisfiability problem. Existing solvers have been successful in dealing with problems with thousands of Boolean variables in reasonable amount of time. And there is a trend to use Boolean satisfiability solvers to deal with practical problems. In this paper, we investigate a method for reducing a particular type of scheduling problem to Boolean satisfiability. Hence we are able to use existing solvers to solve practical scheduling problems and get good performance.

Introduction

Boolean satisfiability problem is considered one of the most classical problems in computing. Traditionally, it is the first NP-complete problem and has been receiving a lot of attention. But the nature of the problem determines that it is nearly impossible to find any tractable method for solving this problem. However, in the last two decades, people made huge progress in solving Boolean satisfiability problem [1,2,3,4]. Some major techniques involve watched-literal technique [5] and clause learning technique [6]. Due to these advanced techniques, existing Boolean satisfiability solvers are able to perform well on practical problems. These problems may contain thousands of Boolean variables, or even millions in some cases. So there is a trend to use Boolean satisfiability solvers to tackle problems of practical interests.

In this paper, we propose a method, which can reduce a scheduling problem to Boolean satisfiability problem. The idea is as follows. In some process of manufacturing, there are some jobs to be done and some limited resources to use. There are some constraints, which say a particular job can only use some particular resources, but there may be alternative resources. For example, to complete job A, one may either use material M_1 and line L_1, or use material M_2 and line L_4. And the goal is to find a way of taking maximum use of available resources to complete all the jobs, subject to the constraints. We will encode the constraints using Boolean formulas. Thus, the scheduling problem can be reduced to a Boolean satisfiability problem.

The rest of the paper is structured as follows. The next section explains precisely what problem we are trying to solve. And then we briefly review Boolean satisfiability problem and methods for solving the problem. In the following section, we explain our method for reducing the scheduling problem to Boolean satisfiability problem. Then we conclude in the last section.

Scheduling Problem

In this paper, we consider the following scenario. Suppose that, in some process of manufacturing, there are m jobs $\{J_1, J_2, \ldots J_m\}$ and n resources $\{R_1, R_2, \ldots, R_n\}$. Here resources have a broad sense. They can be human resources, material resources and machine resources. Each job can only use some particular resources. We associate each job J_i with a set of sets $\{\{R_{k1}, R_{k2},\ldots,R_{kp}\}, \ldots,\{R_{w1},R_{w2},\ldots,R_{wq}\}\}$, meaning that the job J_i can be accomplished using either resources $R_{k1}, R_{k2},\ldots,R_{kp}$ or resources $R_{w1},R_{w2},\ldots,R_{wq}$. For simplicity, we assume that each resource can only be used to complete one job. The goal is to find a way of assigning each job some resources so that all the jobs can be accomplished. Here is a simple example.

Example 1. Suppose that we have 2 jobs $\{J_1, J_2\}$ and 5 resources $\{R_1, R_2, R_3, R_4, R_5\}$, in which R_1 means the first machine, R_2 means the second machine, R_3 means the first person, R_4 means the second person and R_5 mean the third person. We associate job J_1 with the set $\{\{R_1, R_3\}, \{R_1, R_4, R_5\}\}$, meaning that job J_1 can be complete using the first machine and the first person, or using the first machine and the second person and the third person. We associate job J_2 with the set $\{\{R_2, R_3, R_4\}, \{R_2, R_5\}\}$, meaning that job J_2 can be complete using the second machine and the first person and the second person, or using the second machine and the third person. We would like to find a way of assigning each job some resources so that both of them can be accomplished. The solution would be the following. J_1 is assigned R_1 and R_3, J_2 is assigned R_2 and R_5.

Boolean Satisfiability

In this section, we review the Boolean satisfiability problem and major techniques for solving the problem. We begin with some basic terminologies. A Boolean variable is a variable that is either true or false. A literal is a Boolean variable of its negation. A clause is the disjunction of a set of literals. A CNF(Conjunctive Normal Form) formula is a conjunction of clauses. For simplicity, we write a clause as a set of literals, and we write a CNF formula a set of sets. Here is an example. $\{\{x_1, x_2\}, \{x_1, \sim x_2\}\}$ simply means $((x_1 \vee x_2) \wedge (x_1 \vee \sim x_2))$.

The Boolean satisfiability problem can be described as follows. Given a CNF formula, the goal is to find an assignment to each Boolean variable such that each clause in the formula has at least one literal being true. If we can find such an assignment, the formula is satisfiable, otherwise it is unsatisfiable. In the above example, if we assignment true to x_1 and assign false to x_2, the formula is true. So the formula is satisfiable, and $\{x_1$=true, x_2=false$\}$ is a satisfying assignment.

In order to solve Boolean satisfiability problem, one needs to do an exhaustive search in all possible assignments to all the Boolean variables. If we arrangement all possible assignments in a tree, the nodes in the tree is exponential in the number of Boolean variables. So the naïve search would not be practical. There are some techniques to speed up the search. For example, if we know for sure the value of some Boolean variable, we make that decision as early as possible to make the size of the search tree small. In addition, we use a particular indexing technique, called watched-literal technique, to minimize the number of clause access. Another useful technique is called clause-learning. The idea is as follows. If we get a conflict in some part of the search tree, we try to learn from that failure by deducing a clause, which is a logical consequence of the original formula. The learnt clause will be useful to prevent us from making the same mistake in other parts of the search tree. Due to these techniques, modern Boolean satisfiability solvers are able to handle problems with huge number of Boolean variables.

Our Method for Scheduling

In this section, we present our method for reducing the scheduling problem to Boolean satisfiability problem. The idea is as follows. We create some Boolean variables and encode the constraints in the scheduling problem as a Boolean formula using these Boolean variables. The encoding has the property that all the jobs can be accomplished if and only if the Boolean formula is satisfiable and the satisfying assignment corresponds to possible scheduling.

First, we create some Boolean variables of the form U_{mn}, which means that job J_m uses resource R_n in the scheduling. According to the constraints, each resource can only be assigned to one job. To encode this constraint, for each different pair of jobs J_m and J_n, and for each resource R_k, we create the following clause.

$$\sim U_{mk} \vee \sim U_{nk} \qquad (1)$$

Intuitively, the above clause means that either job J_m cannot use resource R_k or job J_n cannot use resource R_k. In other words, job J_m and job J_n cannot both use resource R_k.

We still have another type of constraint to consider. Remember that each job can only use some

particular recourses. Suppose that we associate a job J_i with a set of sets $\{\{R_{k1}, R_{k2},\ldots,R_{kp}\}, \ldots,\{R_{w1},R_{w2},\ldots,R_{wq}\}\}$, meaning that the job J_i can be accomplished using either resources $R_{k1}, R_{k2},\ldots,R_{kp}$ or resources $R_{w1},R_{w2},\ldots,R_{wq}$. Then we will create a formula below.

$$(U_{ik1} \wedge U_{ik2} \wedge \ldots U_{ikp}) \vee \ldots \vee (U_{iw1} \wedge U_{iw2} \wedge \ldots U_{iwp}) \qquad (2)$$

Suppose that we take the conjunction of all the formulas corresponding to all the constraints, and get a formula F. F has the property that all the jobs can be accomplished if and only if the F is satisfiable.

Let us consider Example 1. Using our method described above, we can encode the constraints in Example 1 as the following formula.

$$(\sim U_{11} \vee \sim U_{21}) \wedge (\sim U_{12} \vee \sim U_{22}) \wedge (\sim U_{13} \vee \sim U_{23}) \wedge (\sim U_{14} \vee \sim U_{24}) \wedge (\sim U_{15} \vee \sim U_{25})$$
$$\wedge ((U_{11} \wedge U_{13}) \vee (U_{11} \wedge U_{14} \wedge U_{15})) \wedge ((U_{22} \wedge U_{23} \wedge U_{24}) \vee (U_{22} \wedge U_{25}))$$

The above formula is satisfiable, thus there is a way to assign resources so that all the jobs can be accomplished. In fact, if we give the above formula to a Boolean satisfiability solver, the solver may return the following satisfying assignment.

$$\{U_{11}=\text{true},U_{21}=\text{false},U_{12}=\text{false},U_{22}=\text{true},U_{13}=\text{true},U_{23}=\text{false},U_{14}=\text{false},U_{24}=\text{false},U_{15}=\text{false},U_{25}=\text{true}\}$$

The interpretation for the above satisfying assignment is simply that J_1 is assigned R_1 and R_3, J_2 is assigned R_2 and R_5.

Summary

In this paper, we consider a particular type of scheduling problem. We show that this problem can be reduced to the Boolean satisfiability problem. This reduction is based on a suitable encoding of the constraints in the scheduling problem. The encoding produces a Boolean formula, which has the property that all the jobs can be accomplished if and only if the Boolean formula is satisfiable and the satisfying assignment corresponds to possible scheduling. This method is scalable because the underlying Boolean satisfiability solvers are scalable and the reduction does not blowup the size of the problem.

Acknowledgement

This work is supported by Liaoning Provincial Natural Science Foundation under grant 201202202, Scientific Research Foundation of Liaoning Provincial Education Department under grant L2012388 and L2014440.

References

[1] Een, N. and Sörensson, N. (2003) 'An extensible SAT-solver', Proceeding of the International Conference on Theory and Applications of Satisfiability Testing.

[2] Hantao Zhang, SATO: An Efficient Propositional Prover, Proceedings of the 14th International Conference on Automated Deduction, p.272-275, July 13-17, 1997

[3] Sebastian Ordyniak , Daniel Paulusma , Stefan Szeider, Satisfiability of acyclic and almost acyclic CNF formulas (II), Proceedings of the 14th international conference on Theory and application of satisfiability testing, June 19-22, 2011, Ann Arbor, MI

[4] João P. Marques-Silva , Karem A. Sakallah, GRASP: A Search Algorithm for Propositional Satisfiability, IEEE Transactions on Computers, v.48 n.5, p.506-521, May 1999

[5] Matthew W. Moskewicz , Conor F. Madigan , Ying Zhao , Lintao Zhang , Sharad Malik, Chaff: engineering an efficient SAT solver, Proceedings of the 38th annual Design Automation Conference, p.530-535, June 2001, Las Vegas, Nevada, USA

[6] Audemard, G., Katsirelos, G. and Simon, L. (2010) 'A restriction of extended resolution for clauses learning SAT solvers', Proceeding of the Twenty-Fourth AAAI Conference on Artificial Intelligence, pp.15-20.