Journal of Universal Computer Science, vol. 29, no. 8 (2023), 911-937
submitted: 19/11/2022, accepted: 26/4/2023, appeared: 28/8/2023 CC BY-ND 4.0

Solving Restricted Preemptive Scheduling on Parallel Machines with SAT and PMS

Xiaojuan Liao
(University of Electronic Science and Technology of China, Chengdu, China
Chengdu University of Technology, Chengdu, China
https://orcid.org/0000-0001-6548-2256, liao_xiaojuan@126.com)

Hui Zhang*
(Chengdu University of Technology, Chengdu, China
https://orcid.org/0000-0002-5152-6382, *Corresponding author: zhanghui18@cdut.edu.cn)

Miyuki Koshimura
(Kyushu University, Fukuoka, Japan
https://orcid.org/0000-0002-1561-406X, koshi@inf.kyushu-u.ac.jp)

Rong Huang
(Donghua University, Shanghai, China
https://orcid.org/0000-0003-3248-4620, rong.huang@dhu.edu.cn)

Fagen Li
(University of Electronic Science and Technology of China, Chengdu, China
https://orcid.org/0000-0001-6289-1265, fagenli@uestc.edu.cn)

Abstract: Restricted preemption plays a crucial role in reducing total completion time while controlling preemption overhead. A typical version of restricted preemptive models is k-restricted preemptive scheduling, where preemption is only allowed after a task has been continuously processed for at least k units of time. Though solving this problem of minimizing the makespan on parallel machines is NP-hard in general, it is of vital importance to obtain the optimal solution for small-sized problems, as well as for evaluation of heuristics. This paper proposes optimal strategies to the aforementioned problem. Motivated by the dramatic speed-up of Boolean Satisfiability (SAT) solvers, we make the first step towards a study of applying a SAT solver to the k-restricted scheduling problem. We set out to encode the scheduling problem into propositional Boolean logic and determine the optimal makespan by repeatedly calling an off-the-shelf SAT solver. Moreover, we move one step further by encoding the problem into Partial Maximum Satisfiability (PMS), which is an optimized version of SAT, so that the explicit successive calls of the solver can be eliminated. The optimal makespan of the problem and the performance of the proposed methods are studied experimentally. Furthermore, an existing heuristic algorithm is evaluated by the optimization methods.

Keywords: restricted preemptive scheduling, parallel-machine scheduling, Boolean Satisfiability (SAT), Partial Maximum Satisfiability (PMS)
Categories: F.4, I.2, I.6, J.6, J.7
DOI: 10.3897/jucs.97743

1 Introduction

In this paper, we propose optimization methods for the k-restricted preemptive scheduling problem on parallel identical machines, where preemptions are only allowed after a task has been continuously executed for at least k units of time. The goal of the optimization is to minimize the makespan C, i.e., the maximum completion time on all the machines. In the three-field notation [Graham et al. 1979], this deterministic scheduling problem is denoted by $P|k - pmtn|C_{max}$ [Ecker and Hirschberg 1993].

The scheduling problem involves processing n tasks $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$ on m identical machines[1] $\mathcal{M} = \{M_1, M_2, \ldots, M_m\}$. Each machine can handle only one task at a time, and each task cannot be processed in parallel. Processing time of task $T_j \in \mathcal{T}$ is given by p_j, which is known as priori. Preemptions are restricted by granularity $k \in \mathbb{N}$, meeting the following condition: $\forall T_j \in \mathcal{T}$, if p_j is less than or equal to k, then preemption is prohibited, otherwise preemption may take place after T_j has been continuously processed for at least k units of time.

Ecker and Hirschberg [Ecker and Hirschberg 1993] revealed that the problem $P|k - pmtn|C_{max}$ is NP-hard, except for some special cases on two identical machines. For example, Pieńkosz and Prus [Pieńkosz and Prus 2015] showed that the two-machine problem can be solved in $O(n)$ time if one task has a processing time of at least $4k$, or if one task has a processing time of at least $2k$ and another has a processing time of at least $3k$. Knust et al. [Knust et al. 2019][2] presented an $O(n)$ algorithm for two identical machines, assuming all tasks' processing time to be at least $2k$.

From the best of our knowledge, except for the special cases on two identical machines, there are few studies of optimal algorithms in consideration of the $P|k - pmtn|C_{max}$ problem in the available literature. Alternatively, several approximation methods [Pieńkosz and Prus 2015, Barański 2011] were developed to produce sub-optimal solutions. Approximation methods can find near-optimal solutions that are acceptable in some situations (e.g. [Duo et al. 2020]), particularly when the number of tasks is huge. However, when faced with small-sized problems, obtaining global optimum is desirable, especially for critical systems where the scheduled application is executed many times. Another important application of optimal algorithms is to evaluate the schedules constructed by heuristic methods. If optimal solutions are available for comparison, the quality of solutions output by heuristics can be better judged [Malik et al. 2018].

To this end, this paper considers to optimally solve the $P|k - pmtn|C_{max}$ problem with the aid of Boolean Satisfiability (SAT) and its optimized extension. SAT is the first problem shown to be NP-complete [Cook 1971], which aims to determine whether there exists an assignment of Boolean variables that makes a propositional Boolean formula evaluate true. An algorithm for solving the SAT problem is called a SAT solver. In the last decade, we have witnessed major advances of SAT solvers: problems with thousands of variables are now solved in milliseconds by the state-of-art SAT solvers. Recently, a simple and unified incremental interface to a number of SAT solvers are provided by Python package PySAT [Ignatiev et al. 2018], equipped with a range of propositional encodings for linear (cardinality and pseudo-Boolean) constraints. Motivated by the great potential of SAT technology, in this work, we set out to reduce the $P|k - pmtn|C_{max}$ problem to propositional logic and exploit corresponding solvers to acquire the optimal

[1] The term "machine" can refer to any actual or abstract machine or processor that executes tasks. In this paper, we do not distinguish "machines" and "processors".

[2] Knust et al. [Knust et al. 2019] showed that the algorithm presented for problems with pliability also applies for those with k-restricted preemptions in special cases.

solutions. Specifically, we make the following contributions:

- We present several rules to encode the scheduling problem into a set of propositional Boolean formulas that can be tackled by any off-the-shelf SAT solver. Then binary search is conducted by calling the solver repeatedly until the optimal solution is identified.
- To eliminate the explicitly repeated calls of the solver, we reformulate the scheduling problem as an optimized problem of SAT, i.e., Partial Maximum Satisfiability (PMS). Thanks to the PMS characteristics that satisfy all hard constraints and maximum number of satisfied soft constraints, the optimal schedule can be acquired by running the PMS solver only once.
- We investigate how the optimal makespan changes with varying preemption granularity and changing number of machines. This is the first attempt to explore the optimal solution to the $P|k-pmtn|C_{max}$ problem. Moreover, the factors affecting the performance of the presented methods are also studied in extensive experiments.
- We evaluate an existing heuristic algorithm 2RS [Pieńkosz and Prus 2015] on two parallel identical machines, revealing that the heuristic 2RS can produce approximate solutions of high quality. This paves the way for evaluation of other heuristics, which can be accurately assessed in a similar manner.

This paper is structured as follows. Section 2 formalizes the parallel identical scheduling problem with k-restricted preemptions. Section 3 surveys several preemption models and algorithms for scheduling problems. Section 4 introduces a SAT-based optimization framework for encoding the $P|k-pmtn|C_{max}$ problem into a collection of propositional Boolean formulae that can be tackled by any SAT solver. Section 5 exhibits how to represent the $P|k-pmtn|C_{max}$ problem as a PMS problem to eliminate the solver's repeated calls. Section 6 studies the optimal solution and assesses the performance of the presented methods. Section 7 evaluates a heuristic algorithm using the optimal methods. Finally, Section 8 concludes the paper.

2 Problem Formulation

In the k-restricted preemptive scheduling model, preemption is permitted when a task is processed continuously for at least k units of time. That is, any task $T_j \in \mathcal{T}$ can be replaced by a chain of precisely $\ell_j = \lceil p_j/k \rceil$ indivisible subtasks $\langle T_j^1, \ldots, T_j^{\ell_j} \rangle$. The processing times of subtasks must adhere to two constraints. On the one hand, $\forall T_j \in \mathcal{T}, T_j^i \in T_j \backslash \{T_{\ell_j}\}$, where $\ell_j = \lceil p_j/k \rceil$, the processing time of T_j^i, represented by p_j^i, is either no less than k or equal to 0. In particular, if $p_j^i = 0$, T_j^i is void and cannot be allocated to any machine. Alternatively, if $p_j^i > 0$, T_j^i is valid and must be processed by a single machine. On the other hand, the sum of processing times of all subtasks in T_j equals p_j, i.e., $\forall T_j \in \mathcal{T}$, $\sum_{T_j^i \in T_j} p_j^i = p_j$. To simplify the problem at hand, several predefined constraints should be satisfied as follows.

- Tasks are independent and all available at time 0.
- Each machine can process at most one subtask at a time.

– Subtasks of each task should be processed one after another in form of a chain.

– Processing times of tasks are all integers.

Given a set of tasks $\mathcal{T}$ and that of identical machines $\mathcal{M}$, solving the scheduling problem is equivalent to identifying the following critical information:

– the start execution time s_j^i of each subtask $T_j^i \in T_j$ in $\mathcal{T}$,

– the processing time p_j^i of each subtask $T_j^i \in T_j$ in $\mathcal{T}$,

– the mapping from each valid subtask $T_j^i \in T_j$ in $\mathcal{T}$ to a machine in $\mathcal{M}$.

The objective of scheduling is to minimize the makespan C, which can be defined as the time difference between the start time of the earliest task and the completion time of the latest one. Assume that the earliest task starts at time 0, then the makespan C is given by $C = max\{(s_j^{\ell_j} + p_j^{\ell_j}) \mid T_j \in \mathcal{T}\}$. The optimal makespan is represented as $C^* = Min\big(max\{(s_j^{\ell_j} + p_j^{\ell_j}) \mid T_j \in \mathcal{T}\}\big)$. The range of the optimal makespan C^* can be estimated as follows.

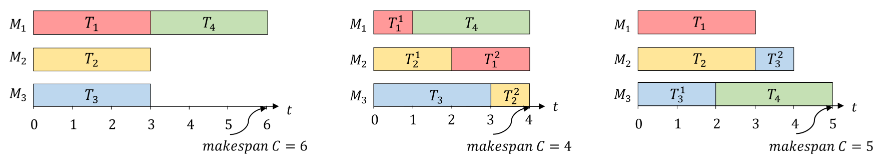
If the preemption granularity k is equal to 0, the k-restricted preemptive scheduling is equivalent to fully preemptive scheduling, which is polynomially solvable by McNaughton's wrap-around rule [McNaughton 1959]. The optimal makespan for $P|0-pmtn|C_{max}$ is:

$$C^* = max\{p_{max}, \sum_{T_j \in \mathcal{T}} p_j/m\}, \tag{1}$$

where p_{max} is the longest processing time of all the tasks, and $\sum_{T_j \in \mathcal{T}} p_j/m$ represents the average machine load. Formula (1) is usually used to compute the lower bound of C^* for any $k > 0$ [Pieńkosz and Prus 2015, Barański 2011], which is represented by C_{lb} in this paper.

The upper bound of the makespan C, denoted by C_{ub}, can be estimated by various heuristics, such as LPT (Longest Processing Time first) and LPT-modified algorithms [Ecker and Hirschberg 1993]. The former allocates tasks in a non-preempted manner to the first machine that becomes idle in the longest processing time order, and the latter is a modification of the LPT algorithm, which assigns part of T_j of length k to the first machine that becomes idle.

Consequently, the optimal makespan C^* is limited to the range $[C_{lb}, C_{ub}]$.



(a) Without preemption (b) Arbitrary preemption (c) 2-restricted preemption

Figure 1: Example schedule of no preemption, fully arbitrary preemption and 2-restricted preemption

Example. Let a task set $\mathcal{T} = \{T_1, T_2, T_3, T_4\}$. Each task has a processing time of 3, i.e., $\forall T_j \in \mathcal{T}$, $p_j = 3$. Suppose that the preemption granularity k is 2. That is, each task should be continuously executed for $k = 2$ units of time before being preempted. Figure 1 compares the optimal results of non-preemptive scheduling, fully preemptive scheduling and 2-restricted preemptive scheduling on three identical machines. Figure 1(a) shows the schedule generated by executing tasks without preemption, producing a makespan of 6. Figure 1(b) displays an average machine load with no constraints on preemptions, resulting in a makespan of 4. Note that the schedule in Figure 1(b) is not feasibly 2-restricted since the red task T_1 is preempted after just one time unit. Figure 1(c) exhibits a feasible 2-restricted schedule with a makespan equal to 5. This is also the shortest makespan for the exemplified 2-restricted preemptive scheduling problem.

In the 2-restricted preemptive example, each task is divided into $\lceil 3/2 \rceil = 2$ subtasks, i.e., $\forall T_j \in \mathcal{T}, T_j = \langle T_j^1, T_j^2 \rangle$. According to the optimal schedule illustrated in Figure 1(c), the start execution time, processing time, and mapping from subtask to machine are summarized in Table 1. Subtasks T_1^2, T_2^2 and T_4^2 are void, thus the start times and machine identifiers that execute these subtasks, are marked by slashes.

	Subtask 1			**Subtask 2**		
	StartTime	**ProcessingTime**	**MachineID**	**StartTime**	**ProcessingTime**	**MachineID**
T_1	0	3	1	/	0	/
T_2	0	3	2	/	0	/
T_3	0	2	3	3	1	2
T_4	2	3	3	/	0	/

Table 1: Critical information of 2-restricted preemptive scheduling in Figure 1(c)

3 Related Research

This section first introduces various preemption models and then investigates existing works of solving scheduling problems.

3.1 Preemption Models

In fully preemptive scheduling models, if preemption is allowed, it is assumed that any task can be preempted at any time and any finite number of times with negligible costs. This model is commonly accepted in many preemptive scheduling problems. The entry "$pmtn$" in the classical three-field notation $\alpha|pmtn|\gamma$ [Graham et al. 1979] stands for the admission of unlimited preemption. McNaughton's wrap-around rule [McNaughton 1959] was presented to optimally address the $P|pmtn|C_{max}$ problem in $O(n)$ time. The optimal makespan C^* is defined in Formula (1).

Considering that preemptions are not always free of cost, variants of preemption models were proposed as an alternative to avoid immoderate task preemptions. Ecker and Hirschberg [Ecker and Hirschberg 1993] presented two types of restricted preemptive models, called k-restricted preemptive and exact k-restricted preemptive models. In the k-restricted preemptive model, preemption is allowed when a task is processed continuously for at least k units of time, while in the exact-k-preemptive model, preemption is allowed when a task has been continuously executed exactly an integer multiple of k time units. Except for some special cases for two identical machines, the problem with arbitrary k is NP-hard for both the k-restricted and exact-k-restricted preemptive models. Braun and Schmidt [Braun and Schmidt 2003] defined an i-preemptive scheduling to bound the maximum number of preemptions by a nonnegative integer number i, demonstrating that the ratio of the optimal i-preemptive makespan versus the optimal fully preemptive makespan is bounded by $2m/(m+i-1)$. The worst case ratio can be guaranteed by a linear algorithm [Jiang et al. 2014]. Soper and Strusevich [Soper and Strusevich 2019] studied the problem with at most one preemption, revealing that the problem is NP-hard on three or more parallel machines. Parametric analysis of the solution quality with at most one preemption was reported in [Soper and Strusevich 2021]. Knust et al. [Knust et al. 2019] developed the concept of pliability to set the lower and upper bounds for the execution length of all operations in flow shop and open shop models, suggesting that most problems with pliability are NP-hard even in the case of two machines. Coffman and Even [Coffman and Even 1998] introduced the concept of limited preemption, where a preempted task cannot be moved from one machine to another. They showed that the worst case ratio of the makespan with no preemption to that with limited preemption is 4/3, and the worst case ratio of the makespan with limited preemption to that with unlimited preemption is 4/3 as well.

The vast majority of studies are only applicable to problems with fixed processing times. Recently, a preemptive model with variable position-dependent processing times has emerged, in which job processing times are non-increasing functions of job positions. Hence, preemption of a job reduces its processing time. Żurowski [Żurowski 2017] presented a two-machine preemptive scheduling model with variable position-dependent processing times, assuming that an arbitrary job can be preempted at most once. Two exact methods were developed to find optimal schedules for medium-sized instances in a relatively short time [Żurowski and Gawiejnowicz 2019].

3.2 Scheduling Algorithms

Attempts at finding optimal solutions to scheduling problems are mainly focused on dynamic programming and branch and bound algorithms [Kravchenko and Werner 2011, Adamu and Adewumi 2013]. In the last decade, formalizing scheduling problems as a class of generalized models, such as mathematical programming [Venugopalan and Sinnen 2014, Hung et al. 2019, Ying et al. 2019, Meng et al. 2020, Bofill et al. 2022], satisfiability modulo theory [Cheng et al. 2017, Malik et al. 2018, Bofill et al. 2020, Lee et al. 2022], SAT [Crawford and Baker 1994, Koshimura et al. 2010, Horbach 2010, Liu et al. 2011, Nieuwenhuis et al. 2021] and PMS [Bofill et al. 2015, Liao et al. 2019, Demirovic et al. 2019, Liao et al. 2021, Bofill et al. 2022], has received considerable attention. Motivated by the tremendous progress in academically and commercially available software for tackling these generalized problems, the scheduling problem can be efficiently addressed in an out-of-the-box manner without further knowledge in scheduling or coding experience from the user [Ying et al. 2019]. Refer to [Pinedo 2016] for a comprehensive understanding of deterministic scheduling models and the related combinatorial problems. The rest of this section surveys literature related to SAT and MaxSAT solving techniques applied to scheduling.

As a pioneering work in SAT formalization adapted in the scheduling community, Crawford and Baker [Crawford and Baker 1994] first encoded scheduling problems into propositional logical formulas, paving the way for subsequent work [Koshimura et al. 2010] that solved six types of open job-shop scheduling problems. Horbach [Horbach

2010] formulated the resource-constrained project scheduling problem as a SAT problem, closing several previously unsolved benchmark instances. Liu et al. [Liu et al. 2011] presented a SAT-based framework to address task graph scheduling with communication costs, outperforming mixed-integer linear programming-based approaches [Jin et al. 2005] by more than an order of magnitude.

More recently, the SAT- and MaxSAT-based technology has shown its great potential in a variety of scheduling applications, such as business-to-business scheduling, employee scheduling and course scheduling problems. Nieuwenhuis et al. [Nieuwenhuis et al. 2021] considered the employee scheduling of a car rental company, which assigns a work schedule to each employee, so that demands, legal and contractual constraints are all met. Experiments demonstrated that modern SAT-based technology can beat the mature branch-and-cut solving methods implemented in well-known state-of-the-art commercial solvers such as CPLEX or Gurobi. Demirovic et al. [Demirovic et al. 2019] provided the first weighted partial maxSAT formulation for the employee scheduling and nurse rostering problem. Challenging instances were presented to test and improve results of MaxSAT solvers in the MaxSAT community. Bofill et al. [Bofill et al. 2015] proposed a partial MaxSAT formulation for the problem of scheduling business-to-business meetings, aiming at minimizing the number of idle time periods in the participants' schedule. The comparative study [Bofill et al. 2022] showed that the MaxSAT approach is significantly superior to other formulations, such as mixed-integer programming and constraint programming, sometimes even achieving orders of magnitude shorter time than its competitors. [Achá and Nieuwenhuis 2014] applied SAT solvers to the curriculum-based course timetabling problem, yielding the best known solutions for 21 out of 32 standard benchmark instances and improving the previously best known solutions for 9. Additionally, 18 new lower bounds were obtained by applying a partial MaxSAT approach. Lemos et al. [Lemos et al. 2022] developed a tool UniCorT, equipped with pre-processing, a MaxSAT solver and a local search procedure to solve university course timetabling problems, winning the 5th place in 2019 International Timetabling Competition.

The aforementioned works schedule tasks in a non-preemptive way to achieve specific scheduling objectives. The limited preemptive counterpart on a single machine was tackled by SMT [Cheng et al. 2017, Wang et al. 2020] and PMS [Liao et al. 2019, Liao et al. 2021], where each task is predefined as a chain of several indivisible fragments, and preemptions are permitted only at fragments' boundaries. In contrast to the model with fixed preemption points, this paper considers a more flexible model with k-restricted preemption, in which the granularity k is introduced to ensure that the length of all portions in a task, except for the last one, is not less than k units. The current research state on this problem is summarized in Table 2. As far as we know, no optimal algorithms for this problem have been explored earlier, and we take the first step towards solving this problem with SAT and PMS techniques.

4 Boolean Satisfiability (SAT) formulation

The SAT problem is the first problem shown to be NP-complete [Cook 1971], which aims to determine whether there exists an assignment of Boolean variables that makes a propositional Boolean formula evaluate true. If such an assignment exists, the formula is satisfiable, otherwise, the formula is unsatisfiable. A propositional Boolean formula is typically written in Conjunction Normal Form (CNF), a widely-used expression joined by a conjunction (logic and) of one or more clauses. A clause is a disjunction (logic or)

Conditions	**Complexity**	**Solution quality**	**Literature**
General cases	NP-hard*	Optimal	[Ecker and Hirschberg 1993]
General cases	$O(n \cdot \log(n))$	Subptimal	[Barański 2011]
General cases with $m = 2$	Open	Optimal	[Knust et al. 2019]
General cases with $m = 2$	$O(n^2)$	Suboptimal	[Pieńkosz and Prus 2015]
General cases with $m = 2$	$O(n \cdot \log(n))$	Suboptimal	[Pieńkosz and Prus 2015]
$m = 2$ and k is fixed.	$O(n)$	Optimal	[Ecker and Hirschberg 1993]
$m = 2, p_1 \geq \frac{1}{2}\sum_{T_j \in \mathcal{T}} p_j$	$O(n)$	Optimal	[Ecker and Hirschberg 1993]
$m = 2, p_1 \geq 4k$	$O(n)$	Optimal	[Pieńkosz and Prus 2015]
$m = 2, p_1 \geq 2k, p_2 \geq 3k$	$O(n)$	Optimal	[Pieńkosz and Prus 2015]
$m = 2, p_j \geq 2k$	$O(n)$	Optimal	[Knust et al. 2019]

*No optimal algorithm is given.

Table 2: Research status of $P|k - pmtn|C_{max}$ problem

of one or more literals, where a literal is an occurrence of a Boolean variable (e.g., x) or its negation (e.g., $\neg x$).

To encode the $P|k-pmtn|C_{\max}$ problem into Boolean propositional logical formulas, we introduce fives types of Boolean variables.

- $b_j^{i,u}$ $(1 \leq j \leq n, 1 \leq i \leq \ell_j, 1 \leq u \leq m)$, which is true if T_j^i is allocated to machine M_u.
- $sd_j^{i,v}$ $(1 \leq j \leq n, 1 \leq i \leq \ell_j, 0 \leq v \leq \lfloor \log_2 C \rfloor)$ for each s_j^i, where C is the schedule length by which all tasks are expected to be completed. $sd_j^{i,v}$ is true if the v-th digit of s_j^i is 1 in the binary number, otherwise, the v-th digit of s_j^i is 0. That is, $s_j^i = \sum_{v=0}^{\lfloor \log_2 C \rfloor} 2^v \cdot sd_j^{i,v}$.
- $pd_j^{i,v}$ $(1 \leq j \leq n, 1 \leq i \leq \ell_j, 0 \leq v \leq \lfloor \log_2 p_j \rfloor)$ for each p_j^i, which is true if the v-th digit of p_j^i is 1 in the binary number, otherwise, the v-th digit of p_j^i is 0. That is, $p_j^i = \sum_{v=0}^{\lfloor \log_2 p_j \rfloor} 2^v \cdot pd_j^{i,v}$.
- pk_j^i $(1 \leq j \leq n, 1 \leq i \leq \ell_j)$, which is true if $p_j^i > 0$, otherwise, $p_j^i = 0$.
- $pr_{i,j}^{x,y}$ $(1 \leq i \leq n-1, i+1 \leq j \leq n, 1 \leq x \leq \ell_i, 1 \leq y \leq \ell_j)$, which is introduced for the case that T_i^x and T_j^y are executed by the same machine u. $pr_{i,j}^{x,y} = 1$ if T_i^x precedes T_j^y when $b_i^{x,u} \wedge b_j^{y,u}$, otherwise, T_j^y precedes T_i^x.

Making use of the above Boolean variables, we design several rules to encode the $P|k - pmtn|C_{\max}$ problem into a set of clauses. $PB(Exp)$ is a collection of clauses created by SAT encoding of the linear pseudo-Boolean expression Exp of the form $\sum_j a_j l_j \{\geq, \leq, =\} b$, where a_j and b are integers and l_j are literals. Pseudo-Boolean constraints arise in a number of important practical applications, and many approaches are devoted to the conversion of a pseudo Boolean formulations to CNF formulas [Roussel and Manquinho 2021], including BDD, sequential weight counters, sorting networks, adder networks and and binary merge. These encodings are all integrated in Python toolkit PySAT [Ignatiev et al. 2018].

(S1) $\forall T_j \in \mathcal{T}, \forall T_j^i, T_j^{i+1} \in T_j$, T_j^i precedes T_j^{i+1}:

$$PB\Big(\sum_{v=0}^{\lfloor \log_2 C \rfloor} 2^v \cdot sd_j^{i+1,v} \geq \sum_{v=0}^{\lfloor \log_2 C \rfloor} 2^v \cdot sd_j^{i,v} + \sum_{v=0}^{\lfloor \log_2 p_j \rfloor} 2^v \cdot pd_j^{i,v}\Big) \tag{2}$$

(S2) $\forall T_i, T_j \in \mathcal{T}$ $(i < j)$, $\forall T_i^x \in T_i$, $\forall T_j^y \in T_j$, $\forall M_u \in \mathcal{M}$, if T_j^x and T_j^y are assigned to the same machine, then T_j^x precedes T_j^y or T_j^y precedes T_j^x:

$$\begin{array}{ll} \neg b_i^{x,u} \vee \neg b_j^{y,u} \vee pr_{i,j}^{x,y} \vee C & (\forall C \in P_1) \\ \neg b_i^{x,u} \vee \neg b_j^{y,u} \vee \neg pr_{i,j}^{x,y} \vee C & (\forall C \in P_2) \end{array} \tag{3}$$

where

$$\begin{aligned} P_1 &= PB\Big(\sum_{v=0}^{\lfloor \log_2 C \rfloor} 2^v \cdot sd_j^{y,v} \geq \sum_{v=0}^{\lfloor \log_2 C \rfloor} 2^v \cdot sd_i^{x,v} + \sum_{v=0}^{\lfloor \log_2 p_i \rfloor} 2^v \cdot pd_i^{x,v}\Big) \\ P_2 &= PB\Big(\sum_{v=0}^{\lfloor \log_2 C \rfloor} 2^v \cdot sd_i^{x,v} \geq \sum_{v=0}^{\lfloor \log_2 C \rfloor} 2^v \cdot sd_j^{y,v} + \sum_{v=0}^{\lfloor \log_2 p_j \rfloor} 2^v \cdot pd_j^{y,v}\Big) \end{aligned} \tag{4}$$

(S3) $\forall T_j \in \mathcal{T}, \forall T_j^i \in T_j$, if T_j^i is valid, i.e., $p_j^i > 0$, then T_j^i is allocated and executed on a single machine:

$$\neg pk_j^i \vee \bigvee_{M_u \in \mathcal{M}} b_j^{i,u} \tag{5}$$

Furthermore, if a valid subtask T_j^i is not the last subtask of T_j, i.e., $T_j^i \in T_j \backslash \{T_j^\ell\}$, then $p_j^i \geq k$:

$$\neg pk_j^i \vee C \quad (\forall C \in PB(\sum_{v=0}^{\lfloor \log_2 p_j \rfloor} 2^v \cdot pd_j^{i,v} \geq k)) \tag{6}$$

(S4) $\forall T_j \in \mathcal{T}, \forall T_j^i \in T_j$, if T_j^i is void, i.e., $p_j^i = 0$, then T_j^i cannot be allocated on any machines[3]:

$$pk_j^i \vee \neg b_j^{i,u} \quad (\forall M_u \in \mathcal{M}) \tag{7}$$

Furthermore, $\forall v \in \{0, 1, \ldots, \lfloor \log_2 p_j \rfloor\}$, $pd_j^{i,v} = 0$ for any void subtask T_j^i:

$$pk_j^i \vee \neg pd_j^{i,v} \tag{8}$$

[3] Assigning void subtasks to machines is possible, but this would increase computation time especially for small k. The reason is explained as follows. That k is small means that tasks can be split into a large number of small pieces, thus much effort has to be made to allocate these (valid and void) subtasks. The rule (S4) prevents void subtasks from being executed, thus the solver does not bother to consider the allocation of these subtasks.

(S5) $\forall T_j \in \mathcal{T}$, the sum of processing time of all subtasks in T_j is equal to p_j:

$$PB\Big(\sum_{T_j^i \in T_j} \sum_{v=0}^{\lfloor \log_2 p_j \rfloor} 2^v \cdot pd_j^{i,v} = p_j\Big) \tag{9}$$

(S6) $\forall T_j \in \mathcal{T}$, T_j should end by C:

$$PB(\sum_{v=0}^{\lfloor \log_2 C \rfloor} 2^v \cdot sd_j^{\ell,v} + \sum_{v=0}^{\lfloor \log_2 p_j \rfloor} 2^v \cdot pd_j^{\ell,v} \leq C) \tag{10}$$

Up to this point, the $P|k-pmtn|C_{max}$ problem has been encoded as Boolean formulas that can be converted to a set of clauses. All of these clauses are conjuncted with logic $\underline{\text{and}}$ to form a Boolean propositional formula in CNF, and an off-the-shelf SAT solver is invoked to determine whether the formula is satisfiable. The initial value of makespan C is set to C_{lb}. If the solver returns a non-empty solution model, it indicates that the solver has successfully found a variable assignment that satisfies the input formula, thereby the optimal makespan is equal to C_{lb}. Otherwise (i.e., the solver returns an empty model), the given makespan C_{lb} is too short to complete all of the tasks. In this situation, we rescale the value of C by performing a binary search in the region $\big[C_{lb}+1, C_{ub}\big]$ to identify the optimal makespan C^*. The process of searching for the optimal makespan is summarized in Algorithm 1.

The input is a $P|k-pmtn|C_{max}$ problem denoted by $\{\mathcal{T}, m, k\}$, where $\mathcal{T}$ is a set of tasks with definite processing times. $m = |\mathcal{M}|$ is the number of identical machines and k is the preemption granularity. The output is the best solution to the scheduling problem, which includes a schedule **S** and a mapping **M**. Schedule **S** assigns an exact start execution time and processing time to each subtask in $T_j \in \mathcal{T}$, and **M** gives a mapping from each valid subtask in $T_j \in \mathcal{T}$ to a machine in $\mathcal{M}$. Function $Encoding(\mathcal{T}, m, k, C)$ yields a set of clauses created by constraints (S1)$\sim$(S6), and $CallSATSolver(\mathcal{C})$ calls a SAT solver to find a solution model $\mathcal{D}$ that satisfies all the clauses in $\mathcal{C}$. If such a model exists, it will be returned by $CallSATSolver(\mathcal{C})$, otherwise, an empty model $\varnothing$ will be returned.

Initially, we encode the problem $\{\mathcal{T}, m, k, C_{lb}\}$ into a set of clauses $\mathcal{C}$, assuming that all tasks can be finished with the minimum schedule length C_{lb} (line 1). A SAT solver is called to evaluate whether there exists a solution model that satisfies $\mathcal{C}$ (line 2). If a non-empty model $\mathcal{D}$ is returned (line 3), the optimal makespan is equal to C_{lb}. The variable assignment stored in $\mathcal{D}$ can produce the associated schedule **S** and makespan **M** (line 4). In particular, from the assignment of Boolean variables sd and pd, the start execution time and processing time of each subtask in $T_j \in \mathcal{T}$ can be calculated. If T_j^i is valid, pointed out by $pk_j^i = 1$, then there must exist a Boolean variable $b_j^{i,u}$ equal to 1, indicating that T_j^i is processed by M_u. Otherwise, $\forall M_u \in \mathcal{M}$, $b_j^{i,u} = 0$.

Lines 6-18 tackles the situation where $C^* > C_{lb}$. First, we define the lower and upper bounds of the optimal makespan as $left$ and $right$, which are initialized to $C_{lb}+1$ and C_{ub}, respectively (line 6), and then search for the optimal makespan in the region $[left, right]$. Given $mid = (left + right)/2$, we explore whether a non-empty model can be returned with the makespan mid (lines 8-10). If a non-empty solution model is returned (line 11), suggesting that any makespan greater than mid can satisfy the

Algorithm 1 Optimal Makespan Identification by SAT Formulation

```
Input:
    A parallel-machine $k$-restricted preemptive scheduling problem $\{\mathcal{T}, m, k\}$.
    $\mathcal{T}$ is a set of tasks;
    $m = |\mathcal{M}|$ is the number of identical machines;
    $k$ is the preemption granularity.
Output:
    Schedule S and mapping M.
 1: $\mathcal{C} = Encoding(\mathcal{T}, m, k, C_{lb})$
 2: $\mathcal{D} = CallSATSolver(\mathcal{C})$
 3: if $\mathcal{D} \neq \varnothing$ then
 4:     return $\{\mathbf{S}, \mathbf{M}\}$ based on model $\mathcal{D}$
        $\{C_{lb}$ is the optimal makespan.$\}$
 5: end if
 6: $left \leftarrow C_{lb} + 1; right \leftarrow C_{ub}$
    {Perform binary search for satisfaction between $[C_{lb} + 1, C_{ub}]$.}
 7: while $left < right$ do
 8:     $mid \leftarrow \lfloor left + right \rfloor / 2$
 9:     $\mathcal{C} = Encoding(\mathcal{T}, m, k, mid)$
10:     $\mathcal{D} = CallSATSolver(\mathcal{C})$
11:     if $\mathcal{D} \neq \varnothing$ then
12:         $right = mid$
13:     else
14:         $left = mid + 1$
15:     end if
16: end while
17: $C^* \leftarrow right$
18: return $\{\mathbf{S}, \mathbf{M}\}$ based on model $\mathcal{D}$
```

problem, then the search space can be narrowed by reducing the upper bound to mid (line 12). Conversely, if the returned model is empty (line 13), indicating that any makespan less than mid is too small to complete all the tasks, the search space can be reduced by increasing the lower bound to $mid + 1$ (line 14). The search for the optimal makespan is repeatedly performed until the entire space in $[left, right]$ is traversed (line 7). Finally, the identified optimal makespan is equal to $right$ (line 17), meeting both of the following conditions:

(1) C^* makes the SAT solver output SAT.

(2) $C^* - 1$ makes the SAT solver output UNSAT.

5 Partial Maximum Satisfiability (PMS) formulation

As can be seen from the SAT encoding given in Section 4, if the optimal makespan is larger than C_{lb}, repeated calls of the SAT solver are inevitable to search for the optimal makespan in region $[C_{lb} + 1, C_{ub}]$. This is because that a SAT solver only determines whether a propositional Boolean formula is satisfiable and returns an empty model with no more information if the formula is unsatisfiable. In this section, we develop a Partial Maximum Satisfiability (PMS) formulation to eliminate the solver's repeated calls. PMS is an optimized version of SAT that distinguishes between hard and soft clauses in a propositional Boolean formula. Solving PMS amounts to finding an optimal assignment that satisfies all hard clauses and the maximum number of soft clauses. To formulate $P|k-pmtn|C_{\max}$ problem with PMS, we introduce three types of Boolean variables.

- $x_{i,j,t}$ $(1 \leq i \leq m, 1 \leq j \leq n, 0 \leq t < C_{ub})$, which is true if machine M_i handles task T_j from time t to $t+1$.
- $y_{j,t}$ $(1 \leq j \leq n, p_j \leq t \leq C_{ub})$, which is true if task T_j finished by time t.
- z_t $(C_{lb} \leq t \leq C_{ub})$, which is true if all tasks in $\mathcal{T}$ finish by time t.

In what follows, we introduce several rules to encode the problem into a set of hard and soft clauses. Rules (H1) $\sim$ (H7) generate hard clauses that should be strictly satisfied, and (S) generates a set of soft clauses that are expected to be satisfied as many as possible. $CNF(Expr)$ represents a CNF formula encoded from cardinality constraint $Expr$. A cardinality constraint is a special type of pseudo-Boolean constraint that limits the number of literals evaluating true from a given set of literals [Roussel and Manquinho 2021].

(H1) Each machine can handle only one task at a time:

$$CNF(\sum_{j=1}^{n} x_{i,j,t} \leq 1) \quad (1 \leq i \leq m, 0 \leq t < C_{ub}) \tag{11}$$

(H2) Each task cannot be processed in parallel:

$$CNF(\sum_{i=1}^{m} x_{i,j,t} \leq 1) \quad (1 \leq j \leq n, 0 \leq t < C_{ub}) \tag{12}$$

(H3) Processing time of task T_j is p_j:

$$CNF(\sum_{i=1}^{m} \sum_{t=0}^{C_{ub}-1} x_{i,j,t} = p_j) \quad (1 \leq j \leq n) \tag{13}$$

(H4) Each portion of the preempted task is processed continuously for at least k unit of time, except for the last portion:

$$\begin{aligned}
&\neg x_{i,j,0} \vee y_{j,p} && \text{if } p = p_j, \\
&\neg x_{i,j,0} \vee x_{i,j,p} && \text{otherwise.} && (14)\\
&x_{i,j,t} \vee \neg x_{i,j,t+1} \vee x_{i,j,(t+1)+p} \vee y_{j,(t+1)+p} && \text{if } t+p+1 \geq p_j, \\
&x_{i,j,t} \vee \neg x_{i,j,t+1} \vee x_{i,j,(t+1)+p} && \text{otherwise.} && (15)\\
&(1 \leq i \leq m, 1 \leq j \leq n) \\
&(0 \leq t < C_{ub} - p - 1, 0 < p < \min\{k, p_j + 1\})
\end{aligned}$$

Formulas (14) and (15) make restrictions for the processing times of subtasks which start from $t = 0$ and $t > 0$, respectively. In particular, for each $p \in (0, \min\{k, p_j + 1\})$, formula (14) states that if T_j starts from time 0, it should keep its execution state at time p, unless T_j is completed by p; on the other hand, formula (15) states that if a subtask of T_j starts from time $t + 1$ $(0 \leq t < C_{ub} - p - 1)$, then its execution state should be maintained at $t + p + 1$, unless it is completed by $t + p + 1$.

(H5) If task T_j finishes by time t, then it must finish by $t + 1$:

$$\neg y_{j,t} \vee y_{j,t+1} \quad (1 \leq j \leq n, p_j \leq t < C_{ub}) \tag{16}$$

(H6) If task T_j finishes by time t, then it must start before time t:

$$\neg y_{j,t} \vee \neg x_{i,j,t} \quad (1 \leq i \leq m, 1 \leq j \leq n, p_j \leq t < C_{ub}) \tag{17}$$

(H7) Relationship between $y_{j,t}$ and z_t:

$$\begin{gathered} \neg z_t \vee y_{j,t} \\ (C_{lb} \leq t \leq C_{ub}, 1 \leq j \leq n) \end{gathered} \tag{18}$$

(S) All tasks in $\mathcal{T}$ are expected to complete by time t:

$$z_t \quad (C_{lb} \leq t \leq C_{ub}) \tag{19}$$

Connecting all the hard clauses in (H1) $\sim$ (H7) and soft clauses in (S) with logic operator <u>and</u>, we convert the scheduling problem to a PMS problem that can be solved by any off-the-shelf PMS solver. If all the soft clauses are satisfied, the optimal makespan is C_{lb}, otherwise, the optimal makespan is larger than C_{lb}. Assume the number of satisfied soft clauses to be n_s, then the optimal makespan is $C^* = C_{ub} - n_s + 1$. The corresponding schedule and mapping can be derived from the assignment of Boolean variables $x_{i,j,t}$ $(1 \leq i \leq m, 1 \leq j \leq n, 0 \leq t < C_{ub})$.

6 Experiments

In this section, we investigate the optimal makespan and evaluate the performance of the proposed methods. Solvers for SAT and PMS are Glucose (version 3.0) and LSUPlus, respectively. Both solvers are integrated in Python toolkit PySAT [Ignatiev et al. 2018].

Glucose [Audemard et al. 2013] is a standard incremental SAT solver based on Minisat [Eén 2003], and LSUPlus is an extension of Linear search SAT-UNSAT algorithm (LSU) [Morgado 2013] for Maximum Satisfiability that supports native cardinality constraints provided by MiniCard solver [Liffiton and Maglalang 2012]. In contrast to most encodings that represent constraint $\sum_{i=1}^{n} x_i \leq k$ as a list of clauses, the native encoding produces only one clause for the constraint, which is suitable to encode the cardinality constraints introduced in Section 5. All the experiments were carried out on the Ubuntu-20.04 virtual platform, equipped with a 3.0-GHz Intel i7-9700 processor and 16-GB RAM.

6.1 Experimental design

The performance is tested on a set of randomly generated problem instances. In the base case, the number of tasks n is set to 10 and that of machines m is set to 4. The processing time p_j of each task $T_j \in \mathcal{T}$ is chosen at random from 10 to 20, and preemption granularity k is set to 9.

Changes in the values of parameters in the base case may affect the performance of SAT and PMS techniques. We construct different scenarios to test these possible changes. Parametric settings of these scenarios are summarized in Table 3. In the $P|k-pmtn|C_{max}$ problem, preemption granularity k and number of machines m are the most important parameters. We will examine the changing trend of the optimal makespans and the solving efficiency under different values of k and m. Specifically, scenario 1 investigates the impact of k, where k is enumerated from 5 to 19. If k is larger than 19, no task can be preempted, which is beyond the scope of our discussion[4]. Scenario 2 examines the influence of m, which is set from 2 to 10. In order to test the scalability of the methods, we establish another two scenarios with expanding number of tasks and processing time. In scenario 3, the number of tasks n is extended from 10 to 25 in increments of 5. In scenario 4, the processing time of each task is enlarged. For each task $T_j \in \mathcal{T}$, p_j is randomly selected from t to $2t$, with t ranging from 10 to 50 in increments of 10. Given each specified interval of processing time $[t, 2t]$, the value of k is fixed to $t/2$ so that each task can be preempted at least once while not being too small to make the problem trivial.

Scenario	$\boldsymbol{k}$	$\boldsymbol{m}$	$\boldsymbol{n}$	$\boldsymbol{p_j}$
1	5, 6,. . ., 19	4	10	$DU(10, 20)$
2	9	2, 3, . . ., 10	10	$DU(10, 20)$
3	9	4	10, 15, . . ., 25	$DU(10, 20)$
4	$0.9t$	4	10	$DU(t, 2t)$, $t = 10, 20, \ldots, 50$

Table 3: Experimental design

6.2 Evaluation on the optimal makespan

For better clarity and concise presentation, we combine scenarios 1 and 2 to show how the optimal makespan changes with variable preemption granularity (where k ranges

[4] The reason why we increase the value of k from 5 is that the heuristics [Barański 2011, Pieńkosz and Prus 2015] can easily find the optimal solution when k is less than the half of the processing time.

from 5 to 19 in increment of 2) and number of machines (where m ranges from 2 to 10). For each fixed parameter configuration, 10 instances were generated. Thus, a total of $9 \times 8 \times 10 = 720$ instances were evaluated. Figure 2 displays the result, where each data point represents the average makespan of 10 instances. The lower bound is computed by McNaughton's wrap-around rule with arbitrary preemption, and the upper bound is estimated by two heuristics, i.e., the LPT algorithm that disallows preemption and the LPT-modified algorithm that allows preemption to occur at boundaries of subtasks of length k.

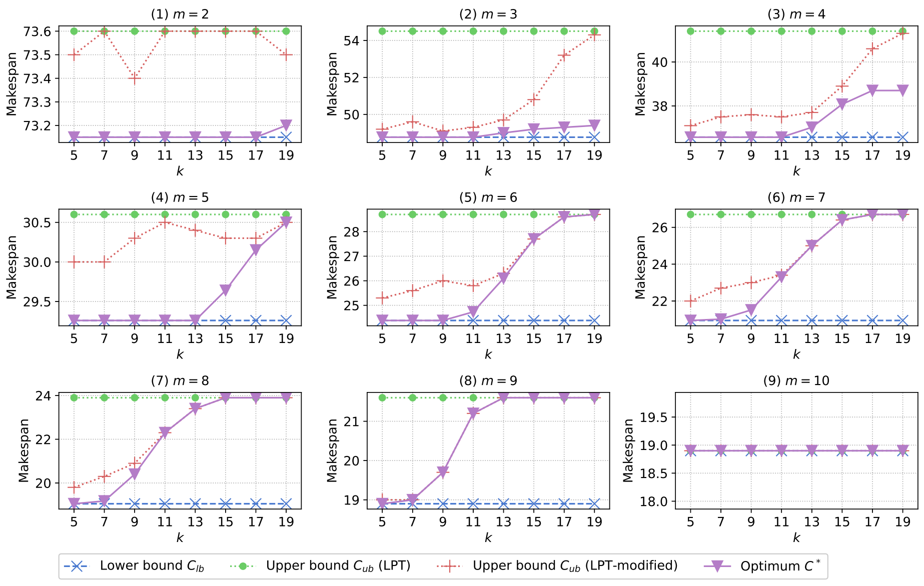
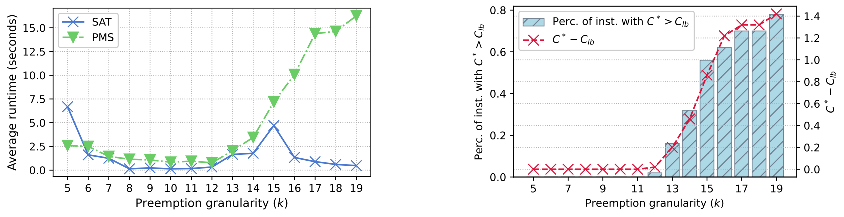


Figure 2: Optimal makespan under the setting of $n = 10$ and $p_j \sim DU(10, 20)$

As shown in Fig. 2, given a fixed k, the optimal makespan C^* decreases as m increases. This is naturally anticipated since more machines would enable more tasks to be processed in parallel and completed in a shorter time span. It can be also observed that with the increase of m, tasks have to be split into smaller pieces to guarantee $C^* = C_{lb}$. For instance, when $m \leq 6$, limiting k to no larger than 9 can guarantee an average machine load that achieves $C^* = C_{lb}$, while when $m = 7$, the upper limit of k is reduced to 7 to make $C^* = C_{lb}$, and further decreased to 5 when $m \geq 8$. The case of $m = 10$ (which equals n) is trivial since each task can be scheduled on a single machine from time 0 and processed to completion without preemption. In consequence, the lower and upper bounds, as well as the optimal makespan, all overlap. In the rest of this section, we omit the special case that m equals n.

Given a fixed m where $2 \leq m \leq 9$, the optimal makespan presents an increasing trend and gradually deviates from C_{lb} as k goes up. We explain the reasons as follows. When k is small enough (e.g., $k = 5$), tasks can be divided into a large number of small pieces to ensure that all machines' loads (schedule lengths) are identical. At this point, finding a feasible schedule that makes $C^* = C_{lb}$ is an easy task. As k grows, the



(a) Average runtime (b) Deviation of C^* from C_{lb}

Figure 3: Statistics with various preemption granularities k (scenario 1)

number of subtasks decreases while the length of each subtask increases. In this situation, finding an average machine load becomes impossible, leading to a longer makespan that is greater than C_{lb}.

The optimization methods can be exploited to evaluate approximation algorithms. As can be seen in Fig. 2, the LPT-modified algorithm performs no worse than the LPT algorithm in all cases. By comparing the solution of the LPT-modified algorithm with the optimal makespan C^*, we can safely conclude that the LPT-modified algorithm is able to produce the optimal solution when $m = 9$. In Section 7, more details will be revealed on the evaluation of heuristics.

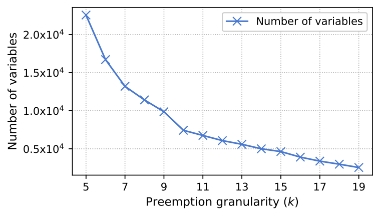
6.3 Evaluation on critical parameters

In this subsection, we test the impact of critical parameters k and m on the performance of SAT and PMS techniques. C_{lb} and C_{ub} are calculated by McNaughton's rule and the LPT-modified algorithm, respectively. For each fixed parameter configuration, 50 instances were generated. Overall, we built 1150 instances in total for testing, with 750 and 400 instances generated for scenarios 1 and 2, respectively. A time limit of 300 seconds is imposed on each instance. If a method fails to produce the optimal result within the time limit, it is terminated and applied to the next instance. Metrics for comparison include (1) the proportion of instances solved within the time limit and (2) the average computation time spent on the solved instances.

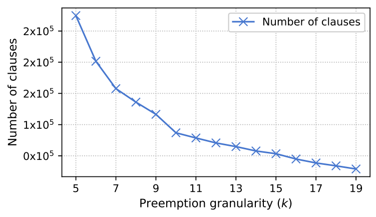
6.3.1 Study on preemption granularity *k*

This subsection discusses the effect of preemption granularity k on the performance of SAT and PMS formulations. The summary information is depicted in Fig. 3. Figure 3(a) shows the solving efficiency of two methods, in which each data point is the average computation time of 50 instances. Generally, when $k \leq 11$, performances of both SAT and PMS enhance with the increase of k. By contrast, when $k \geq 12$, the average runtime of SAT presents an up-down trend, while that of PMS jumps considerably as k goes up. In what follows, we will elaborate on the reasons of such performance changes.

When k is small enough ($k \leq 11$), tasks can be split into small pieces to guarantee a machine load balanced schedule. Figure 3(b) shows that when $k \leq 11$, C^* is equal to C_{lb} without exception. In this case, the SAT-based method only needs to call the solver once to verify that C_{lb} is large enough to make all the clauses satisfiable. Similarly, PMS can also satisfy all the clauses (including hard and soft clauses). Therefore, the

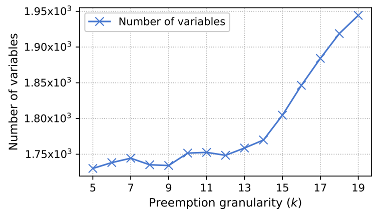


(a) Number of variables

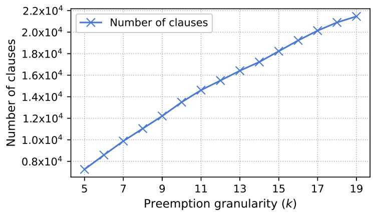


(b) Number of clauses

Figure 4: Numbers of variables and clauses generated by SAT formulation in scenario 1



(a) Number of variables



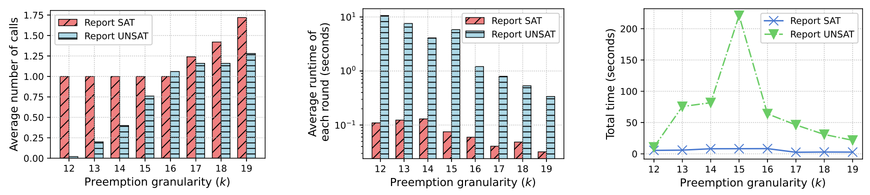
(b) Number of clauses

Figure 5: Numbers of variables and clauses generated by PMS formulation in scenario 1

solving efficiency of SAT and PMS is solely dependent upon the time taken to identify the satisfiable solution, which is affected by the search space and related constraints. In SAT, as depicted in Fig. 4, fewer Boolean variables and clauses are generated when k gets larger. The reduction on the number of variables and clauses decreases the problem size and thus improves the search speed. For PMS, as shown in Fig. 5, when $k \leq 11$, the number of variables fluctuates without substantial increase[5], while that of clauses grows gradually. The constant number of variables reflects the limited scope of the search space, and the increasing number of clauses constrain the search direction more strictly. This enables the solver to prune the search space more efficiently, thereby reducing the average computation time effectively. Therefore, as shown in Fig. 3(a), when $k \leq 11$, the overall performance of both SAT and PMS improves as k goes up.

When k is increased to larger than 11, the optimal makespan C^* gradually deviates from the lower bound C_{lb} (as shown in Fig. 3(b)). This leads to a situation that is more complicated. In SAT, to precisely find out the minimum makespan that makes the formula satisfiable, the SAT solver has to be called repeatedly and indicate that not all the clauses can be satisfied when the tested makespan is between C_{lb} and $C^* - 1$. Likewise, in PMS, not all the soft clauses $z_t(C_{lb} \leq t \leq C_{ub})$ can be satisfied when $C^* > C_{lb}$. To prove that all the clauses are satisfiable (i.e., to report satisfiability), the solver only needs to find one of the feasible solutions to make the given CNF formula true and stops searching as soon as the solution has been identified. In comparison, to declare that not all the clauses are satisfiable (i.e., to report unsatisfiability), the solver has to traverse the entire search space to ensure that no variable assignment can satisfy the given formula. Apparently,

[5] The number of variables is closely related to C_{lb} and C_{ub}. If both C_{lb} and C_{ub} are independent of k, the number of variables is constant. The reason why the number fluctuates here is that C_{ub} computed by LPT-modified algorithm undulates with the increase of k.



(a) Average number of calls (b) Average time in each round (c) Total time of 50 instances

Figure 6: Statistics of reporting satisfiability and unsatisfiability by SAT solver when $k \geq 12$

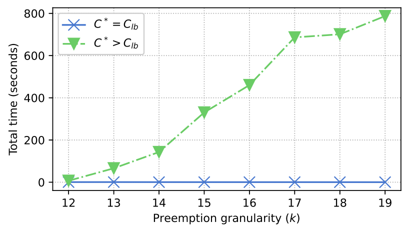
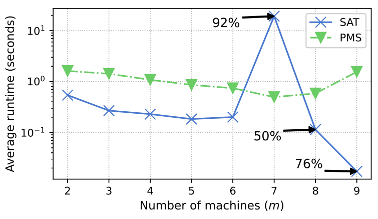


Figure 7: Total time of solving instances with $C^* = C_{lb}$ *and* $C^* > C_{lb}$ *by PMS solver when* $k \geq 12$

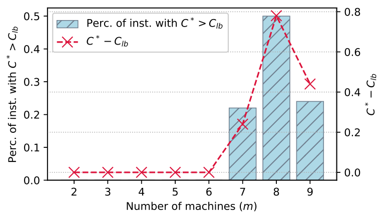
it can be inferred that reporting an unsatisfiable result is more time-consuming than searching for a satisfiable one in a space of the same size. In what follows, we will collect the time of tackling satisfiable and unsatisfiable instances to demonstrate the great impact of reporting unsatisfiability on the solving efficiency of SAT and PMS. Statistics of SAT and PMS are shown in Figs. 6 and 7, respectively.

In the SAT-based method, to solve an instance with $C^* > C_{lb}$, the SAT solver may report unsatisfiability several times during binary search. Figure 6(a) shows that there is no huge gap between the numbers of satisfiable and unsatisfiable reports, and Figure 6(b) shows that in each round of calls, the average time of declaring unsatisfiability is substantially longer than that of reporting satisfiability, although the time of reporting unsatisfiability generally shows a downward trend. Such decline is due to the reduced search space led by the decreasing number of variables and clauses (as shown in Fig. 4). Overall, Figure 6(c) compares the total time taken to report satisfiability and unsatisfiability at each $k \geq 12$, suggesting that it is the time of declaring unsatisfiability for instances with $C^* > C_{lb}$ that has a greater impact on the solving efficiency of the SAT technique. The up-down trend of the total runtime is jointly led by the increasing number of unsatisfiable reports and the decreasing time of each unsatisfiable report.

Figure 7 compares the PMS's total time of solving different types of instances when $k \geq 12$. Solving instances with $C^* > C_{lb}$ requires identifying the minimum number of unsatisfiable soft clauses while solving those with $C^* = C_{lb}$ only needs to give a feasible variable assignment that satisfies all the clauses. Evidently, as shown in Fig. 7, the total time spent on instances with $C^* > C_{lb}$ is significantly longer than solving instances with $C^* = C_{lb}$, which always stands at a very low level. This demonstrates that the time of solving instances with $C^* > C_{lb}$ is the dominant factor to affect the PMS's computation time.

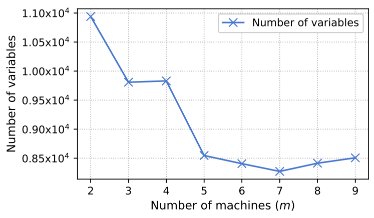


(a) Average runtime

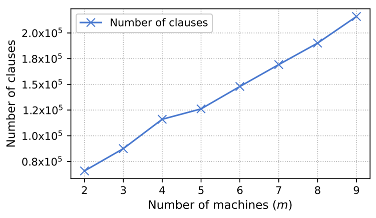


(b) Deviation of C^* from C_{lb}

Figure 8: Statistics with various numbers of machines m (scenario 2)



(a) Number of variables

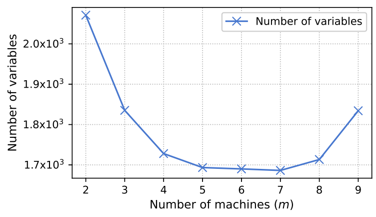


(b) Number of clauses

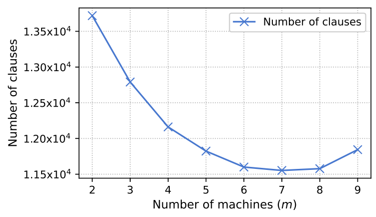
Figure 9: Numbers of variables and clauses generated by SAT in scenario 2

6.3.2 Study on number of machines m

This subsection discusses the solving efficiency of SAT and PMS with changing number of machines m. Although Fig. 2 shows that when $m = 9$, the heuristic LPT-modified algorithm can output the optimal schedule, we still evaluate the performance of SAT and PMS in this case to maintain the integrity of evaluation. The summary statistics when m ranges from 2 to 9 is exhibited in Fig. 8. Figure 8(a) depicts the change of performance with various m, in which each data point is the average computation time of the solved problem instances. A number with an arrow in the figure denotes the proportion of instances solved within the time limit and is omitted if the solver addressed all the 50 instances. Generally, Figure 8(a) shows that the performance of SAT fluctuates significantly, while that of PMS remains relatively stable with the increasing m. In what follows, the performance variations of both methods will be analyzed.



(a) Number of variables



(b) Number of clauses

Figure 10: Numbers of variables and clauses generated by PMS in scenario 2

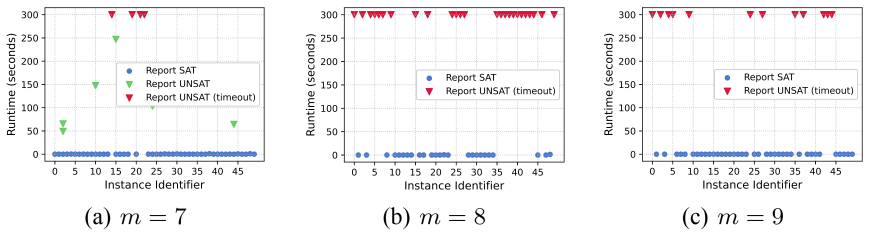


Figure 11: Time of reporting satisfiability and unsatisfiability by SAT solver for each instance when $m = 7, 8, 9$

As depicted in Fig. 8(b), when $m \leq 6$, C^* is equal to C_{lb}, which means that the SAT-based method only needs to call the SAT solver once to determine the optimal solution. Figure 8(a) shows that in this situation, the average computation time of SAT presents a slight decrease from 0.5 to 0.2 seconds. The gradual decrease in runtime is due to the reduced search space. As shown in Fig. 9, the number of variables generated by SAT shows an overall decline, while that of clauses presets a steady rise. The reduction of the number of variables cuts down the search space, and the increase of the number of clauses sets more constraints for a limited number of variables. This enables the solver to trim the search space more efficiently, so as to speed up the process of finding out a feasible assignment of variables.

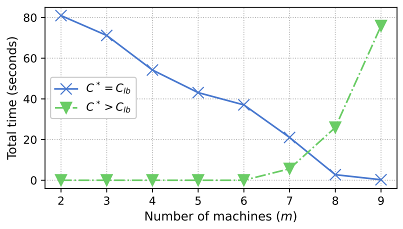
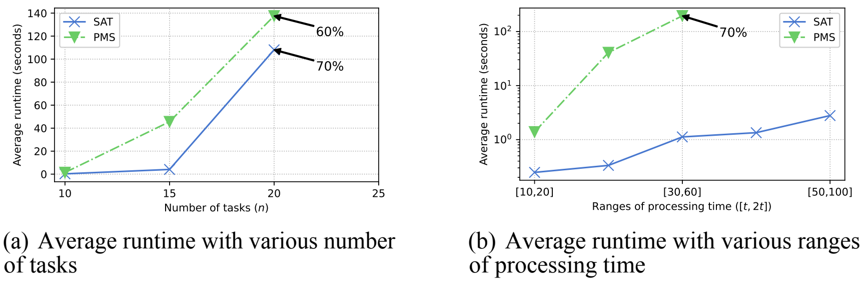


Figure 12: Total time of solving instances with $C^* = C_{lb}$ and $C^* > C_{lb}$ by PMS solver with various m

When $m \geq 7$, the optimal makespan C^* deviates from C_{lb} (as depicted in Fig. 8(b)), and the performance of SAT goes unstable accordingly (as shown in Fig. 8(a)). To investigate the main factors that influence the performance, in Fig. 11, we collect the runtime of each unsatisfiable and ununsatisfiable report for each instance when $m = 7, 8, 9$. If the solver failed to report a result within the time limit, its runtime is marked with a red triangle at 300 seconds. Clearly, according to Fig. 11, the time spent in reporting unsatisfiability is considerably longer than that taken by reporting satisfiability, which suggests that reporting unsatisfiability by traversing the search space is the main reason that affects the efficiency. Another worth noting fact is that the solver accomplished the majority of instances within the time limit when $m = 7$, while suffered timeout when trying to report every unsatisfiable result when $m = 8$ and $m = 9$. This is because that the case of $m = 7$ generates fewer variables and clauses than that of $m = 8$ and $m = 9$ (as depicted in Fig. 9), resulting in smaller-sized problem to be tackled. Therefore, declaring unsatisfiability when $m = 7$ takes less time than the other two cases.

In PMS, Figure 8(a) shows that the performance levels off with increasing m. The average computation time of PMS is mainly influenced by the number of variables and clauses (as shown in Fig. 10), which maintains the same down and up trend as m goes up. It is worthy to be noted that when $m = 9$, the number of variables and clauses is significantly less than that when $m = 2$ (as shown in Fig. 10), while the computation time in these two cases is almost identical (as shown in Fig. 8(a)). The reason can be explained by observing Fig. 12, which compares the total time spent in solving instances with $C^* = C_{lb}$ and $C^* > C_{lb}$. When $m \leq 6$, the computation time of PMS is solely determined by the time of solving instances with $C^* = C_{lb}$, while when $m > 6$, the time of solving instances with $C^* > C_{lb}$ accounts for a major part of the total computation time. When $C^* > C_{lb}$, the solver has to figure out the minimum number of unsatisfiable soft clauses, which is more time-consuming than proving a satisfiable formula even given the search space of the same size. This explains why the search space of $m = 9$ is much smaller than that of $m = 2$, while the total computation time of PMS in these two cases is almost the same.

6.4 Evaluation on Scalability



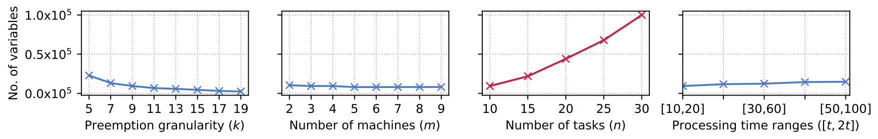
(a) Average runtime with various number of tasks

(b) Average runtime with various ranges of processing time

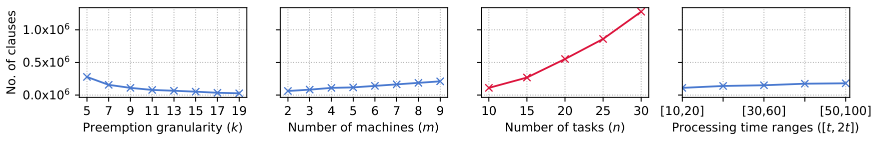
Figure 13: Performance evaluation on scalability

This subsection investigates the scalability of SAT- and PMS-based methods when the number of tasks and the range of processing time are extended. Parametric settings are consistent with scenarios 3 and 4 described in Subsection 6.1. For each fixed parametric configuration, 10 instances are generated.

Figure 13 displays the performance evaluation with various task numbers and processing time ranges. Obviously, as depicted in Fig. 13(a), with the increase of n, the solving efficiency of both SAT and PMS techniques deteriorates so seriously that no instances can be solved when n reaches 25. This indicates that the SAT- and PMS-based methods are both inappropriate for solving scheduling with large number of tasks. With respect to the expanding processing time ranges, Figure 13(b) shows that with the increase of processing time, the average runtime of PMS soars rapidly and no instances can be solved within the time limit when t is increased to 40, whereas that of SAT increases moderately from 0.2 to 2 seconds.

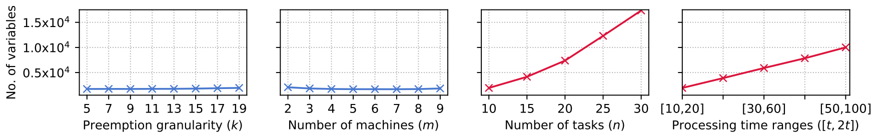


(a) Number of Boolean variables

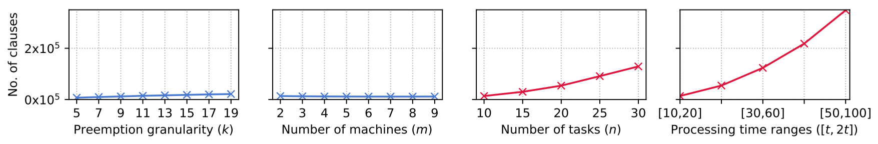


(b) Number of clauses

Figure 14: Numbers of variables and clauses generated by SAT in scenarios 1-4



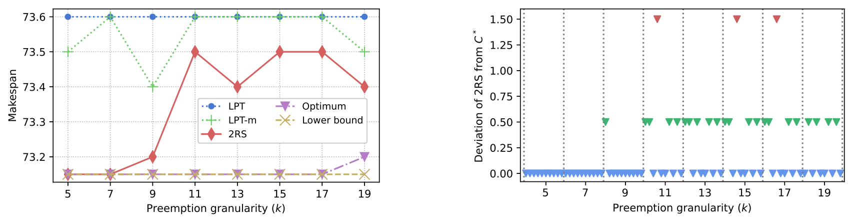
(a) Number of Boolean variables



(b) Number of clauses

Figure 15: Numbers of variables and clauses generated by PMS in scenarios 1-4

The optimal makespan C^* of each instance in scenarios 3 and 4 is equal to C_{lb} without exception, thus both SAT- and PMS-based methods only need their corresponding solvers to find a feasible variable assignment to satisfy all the clauses. The computation time is mainly affected by the search space and the associated constraints, embodied in the number of Boolean variables and clauses. Figures 14 and 15 exhibit the numbers of variables and clauses generated by SAT and PMS. To observe the results more clearly, we compare the problem scales in all the four scenarios. As illustrated in Figure 14 (red curves), the numbers of variables and clauses increase dramatically with the increasing number of tasks, but remains reasonably modest with the expansion of processing time ranges. This explains why the performance of SAT degenerates drastically as the number of tasks increases while keeps efficient with expanding ranges of processing time. Figure 15 (red curves) shows that the numbers of variables and clauses generated by PMS increase noticeably when either the number of tasks or the processing time increases. As a result, in scenarios 3 and 4, the performance of PMS suffers from sharp degeneration accordingly.



(a) Average values of makespan for various values of k.

(b) Deviation of 2RS's solution from the optimal makespan

Figure 16: Evaluation of 2RS

7 Evaluating heuristics

$Pie\acute{n}kosz$ et al. [Pieńkosz and Prus 2015] presented the 2RS heuristic algorithm, which considers the k-restricted preemptive scheduling problem on two parallel identical machines, denoted by $P2|k-pmtn|C_{max}$. They found that when the preemption granularity k is relatively small, the makespan of schedules computed by 2RS is equal to C_{lb}, implying that the makespan found by 2RS is optimal. However, as k increases, the makespan output by 2RS becomes larger than C_{lb}. Due to the lack of the optimal solution, the quality of the reported makespan is uncertain in this case.

In this section, we evaluate the performance of 2RS on 80 randomly generated instances using the optimization methods. Each instance contains 10 tasks with processing times randomly generated from 10 to 20. The value of k is enumerated in increments of 2 from 5 to 19. As described in [Pieńkosz and Prus 2015], the computational complexity of 2RS is $O(n^2)$. In our experiments, 2RS produces approximate solutions in around 1.5 milliseconds, which is substantially faster than the optimal methods.

Figure 16(a) displays the average values of the makespan obtained by heuristics for different k values. Each data point represents the average makespan of 10 instances. For better comparison, we collect the lower bound computed by McNaughton's rule and two makespans output by two classical heuristics, i.e., the LPT algorithm and its modification, denoted by LPT and LPT-m in Fig. 16(a), respectively. It is obvious from Fig. 16(a) that though 2RS is always superior to LPT and LPT-modified algorithms no matter how k is changed, it performs worse when addressing problems with larger k. To be specific, when k is small (i.e, $k \leq 7$), the makespan computed by 2RS is equal to C^*, while when $k \geq 9$, 2RS's solution becomes larger than C^*.

Figure 16(b) depicts the difference between 2RS's output and C^* on each instance. When $k \leq 7$, 2RS managed to find C^* for all the instances, while when $k = 9$, 2RS failed to identify C^* in one out of 10 instances. More seriously, when $k \geq 11$, 2RS could only identify C^* for merely about half of the instances. Though the percentage of instances optimally solved by 2RS is less satisfactory, the maximum difference between 2RS's output and C^* is just 1.5, which is quite small when compared to C^*. This demonstrates that 2RS is capable of obtaining high-quality approximation solutions, even if such solutions are not exactly equivalent to the optima.

8 Conclusions

In this paper, we studied the problem $P|k-pmtn|C_{\max}$, i.e., the parallel-machine k-restricted preemptive scheduling problem to minimize the makespan. In order to obtain the optimal solution, we presented two optimization methods based on the formulations of Boolean Satisfiability (SAT) and Partial Maximum Satisfiability (PMS). This is the first attempt at finding the optimal makespan for the $P|k-pmtn|C_{max}$ problem. Through a series of experiments, we made a detailed presentation on how the optimal solution changes with various values of preemption granularity and machine number. In addition, the factors that affect the solving performance of the proposed methods were discussed. Finally, existing heuristic algorithms LPT, 2RS was evaluated. With the help of optimization methods, the quality of solutions output by heuristic algorithms can be accurately judged.

Acknowledgements

This research was funded by National Natural Science Foundation of China [Grant Number 61806171], Sichuan Province Science and Technology Program [Grant Number 2022YFG0198] and JSPS KAKENHI [Grant Numbers JP17K00307, JP19H04175].

References

[Achá and Nieuwenhuis 2014] Achá, R. J. A., Nieuwenhuis, R.: "Curriculum-based course timetabling with SAT and MaxSAT"; Annals of Operations Research 218, 1 (2014) 71-91.

[Adamu and Adewumi 2013] Adamu, Muminu O., Adewumi, Aderemi O.,: "A survey of single machine scheduling to minimize weighted number of tardy jobs"; Journal of Industrial & Management Optimization, 10, 1 (2013) 219–241.

[Audemard et al. 2013] Audemard, G., Lagniez, J.-M., Simon, L.: "Improving glucose for incremental SAT solving with assumptions: Application to mus extraction"; Theory and Applications of Satisfiability Testing, Int. Conf., Lect. Notes in Comp. Sci., Springer, Berlin (May 2013), 369-375.

[Barański 2011] Barański, T.: "Task scheduling with restricted preemptions"; Proc. Federated Conference on Computer Science and Information Systems, Szczecin, Poland (2011), 231–238.

[Bofill et al. 2015] Bofill, M., Garcia, M., Suy, J., Villaret, M.: "MaxSAT-based scheduling of B2B meetings"; Integration of AI and OR Techniques in Constraint Programming, Barcelona, Spain (2015) 65-73.

[Bofill et al. 2020] Bofill, M., Coll, J., Suy, J., Villaret, M.: "SMT encodings for resource-constrained project scheduling problems"; Computers & Industrial Engineering, 149: 106777 (2020) 1-20.

[Bofill et al. 2022] Bofill, M., Coll, J., Garcia, M., Giráldez-Cru, J., Pesant, G., Suy, J., Villaret, M.: "Constraint Solving Approaches to the Business-to-Business Meeting Scheduling Problem"; Journal of Artificial Intelligence Research, 74 (2022) 263-301.

[Bofill et al. 2022] Bofill, M., Coll, J., Martín, G., Suy, J., Villaret, M.: "The sample analysis machine scheduling problem: definition and comparison of exact solving approaches"; Computers & Operations Research, 142, 105730 (2022) 1-13.

[Braun and Schmidt 2003] Braun, O., and Schmidt, G.: "Parallel processor scheduling with limited number of preemptions"; SIAM Journal on Computing, 32, 3 (2003) 671–680.

[Cheng et al. 2017] Cheng, Z., Zhang, H., Tan, Y. Lim, Y.: "SMT-based scheduling for overloaded real-time systems"; IEICE Transactions on Informations and Systems, E100-D, 5 (2017) 1055-1066.

[Coffman and Even 1998] Coffman, E. G., Even, S.: "A note on limited preemption"; Parallel Processing Letters, 8, 1 (1998): 3–6.

[Cook 1971] Cook, S. A.: "The complexity of theorem-proving procedures"; Proc. 3^rd Annual ACM Symposium on Theory of Computing, Ohio, USA (May 1971) 151–158.

[Crawford and Baker 1994] Crawford, J. M., Baker, A. B.: "Experimental results on the application of satisfiability algorithms to scheduling problems"; Proc. 12^{th} AAAI National Conference on Artificial Intelligence, CA, USA (1994) 1092–1097.

[Demirovic et al. 2019] Demirovic, E., Musliu, N., Winter, F.: "Modeling and solving staff scheduling with partial weighted maxSAT"; Annals of Operations Research, 275, 1 (2019) 79-99.

[Duo et al. 2020] Duo, B., Wu, Q., Yuan, X., Zhang, R.: "Energy efficiency maximization for full-duplex UAV secrecy communication"; IEEE Trans. Veh. Technol., 69, 4 (2020) 4590–4595.

[Ecker and Hirschberg 1993] Ecker, K., Hirschberg, R.: "Task scheduling with restricted preemptions"; Proc. 5^{th} Parallel Architectures and Languages Europe, Berlin, Heidelberg, Germany (June 1993) 464–475.

[Eén 2003] Eén, N., Sörensson, N.: "An extensible SAT-solver"; Proc. 6^{th} Theory and Applications of Satisfiability Testing, Italy (2003) 502–518.

[Graham et al. 1979] Graham, R. L., Lawler, E. L., Lenstra, J. K., Kan, A. H. G. R.: "Optimization and approximation in deterministic sequencing and scheduling: a survey". Annals of Discrete Mathematics, 5, 1 (1979) 287–326.

[Horbach 2010] Horbach, A.: "A Boolean satisfiability approach to the resource-constrained project scheduling problem"; Annals of Operations Research, 181, 1 (2010) 89–107.

[Hung et al. 2019] Hung, H. C., Lin, B. M. T., Posner, M. E., Wei, J. M.: "Preemptive parallel-machine scheduling problem of maximizing the number of on-time jobs"; Journal of Scheduling, 22, 4 (2019) 413-431.

[Ignatiev et al. 2018] Ignatiev, A., Morgado, A., Marques-Silva, J.: "PySAT: A Python toolkit for prototyping with SAT oracles"; Proc. of 21^{st} Theory and Applications of Satisfiability Testing, Oxford, UK (July 2018) 428–437.

[Jiang et al. 2014] Jiang, Y., Weng, Z., Hu, J.: "Algorithms with limited number of preemptions for scheduling on parallel machines"; Journal of Combinatorial Optimization, 27, 4 (2014):711–723.

[Jin et al. 2005] Jin, Y., Satish, N., Ravindran, K., Keutzer, K.: "An automated exploration framework for FPGA-based soft multiprocessor systems"; Proc. 3^rd Hardware/Software Codesign and System Synthesis, Jersey, NJ, USA (Sep. 2005) 273–278.

[Knust et al. 2019] Knust, S., Shakhlevich, N., Waldherr, S., Weiβ, C.: "Shop scheduling problems with pliable jobs"; Journal of Scheduling, 22, 6 (2019) 635–661.

[Koshimura et al. 2010] Koshimura, M., Nabeshima, H., Fujita, H., Hasegawa, H.: "Solving open job-shop scheduling problems by SAT encoding"; IEICE Transactions on Informations and Systems, E93-D, 8 (2010) 2316–2318.

[Kravchenko and Werner 2011] Kravchenko, S. A., Werner, F.: "Parallel machine problems with equal processing times: a survey"; Journal of Scheduling, 14, 5 (2011) 435–444.

[Lee et al. 2022] Lee, D., Lin, B., Cheng, C. K.: "SMT-based contention-free task mapping and scheduling on 2D/3D SMART NoC with mixed dimension-order routing. ACM Transactions on Architecture and Code Optimization, 19, 1 (2022): 5:1-5:21.

[Lemos et al. 2022] Lemos, A., Monteiro, P. T., Lynce, I.: "Introducing UniCorT: an iterative university course timetabling tool with MaxSAT"; Journal of Scheduling, 25, 4 (2022) 371-390.

[Liao et al. 2019] Liao, X., Zhang, H., Koshimura, M., Huang, R., Yu, W.: "Maximum satisfiability formulation for optimal scheduling in overloaded real-time systems"; Proc. 16^{th} Pacific Rim Int. Conf. on Artificial Intelligence, Yanuca Island, Fiji (Aug 2019) 618–631.

[Liao et al. 2021] Liao, X., Zhang, H., Koshimura, M., Huang, R., Yu, W., Li, F.: "Modeling and solving scheduling in overloaded situations with weighted partial MaxSAT", Mathematical Problems in Engineering, 2021 (2021) 1–17.

[Liffiton and Maglalang 2012] Liffiton, M. H., Maglalang, J. C.: "A cardinality solver: More expressive constraints for free"; Proc. 15^{th} Theory and Applications of Satisfiability Testing, Berlin, Heidelberg, Germany (2012) 485–486.

[Liu et al. 2011] Liu, W., Gu, Z., Xu, J., Wu, X., Ye, Y.: "Satisfiability modulo graph theory for task mapping and scheduling on multiprocessor systems"; IEEE Transactions on Parallel and Distributed Systems, 22, 8, (2011) 1382–1389.

[Malik et al. 2018] Malik, A., Walker, C. G. O'Sullivan M. J., Sinnen, O.: "Satisfiability modulo theory (SMT) formulation for optimal scheduling of task graphs with communication delay"; Computers & Operations Research, 89 (2018) 113–126.

[McNaughton 1959] McNaughton, R.: "Scheduling with deadlines and loss functions"; Management Science, 6 (1959) 1–11.

[Meng et al. 2020] Meng, L., Zhang, C., Ren, Y., Zhang, B., Lv, C.: "Mixed-integer linear programming and constraint programming formulations for solving distributed flexible job shop scheduling problem"; Computers & Industrial Engineering, 142, 106347 (2020) 1-13.

[Morgado 2013] Morgado, A., Heras, F., Liffiton, M., Planes, J., Marques-Silva, J.: "Iterative and core-guided maxsat solving: A survey and assessment"; Constraints, 18, 4 (2013) 478–534.

[Nieuwenhuis et al. 2021] Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., Rollon, E.: "Employee scheduling with SAT-based pseudo-Boolean constraint solving"; IEEE Access, 9 (2021) 142095-142104.

[Pieńkosz and Prus 2015] Pieńkosz, K., Prus, A.: "Task scheduling with restricted preemptions on two parallel processors"; Proc. 20^{th} International Conference on Methods and Models in Automation and Robotics (MMAR), Miedzyzdroje, Poland (08 2015) 58–61.

[Pinedo 2016] Pinedo, M. L.: "Deterministic Models"; Scheduling: Theory, Algorithms, and Systems - Fifth Edition, Springer press (2016) 1-242.

[Roussel and Manquinho 2021] Roussel, O., Manquinho, V. M.: "Pseudo-boolean and cardinality constraints"; Handbook of satisfiability - Second Edition, 336, IOS press (2021) 1087-1129.

[Soper and Strusevich 2019] Soper. A. J., Strusevich. V. A.: "Schedules with a single preemption on uniform parallel machines"; Discrete Applied Mathematics, 261 (2019) 332–343.

[Soper and Strusevich 2021] Soper. A. J., Strusevich. V. A.: "Parametric analysis of the quality of single preemption schedules on three uniform parallel machines"; Annals of Operations Research, 298, 1 (2021) 469–495.

[Venugopalan and Sinnen 2014] Venugopalan, S., Sinnen, O.: "ILP formulations for optimal task scheduling with communication delays on parallel Systems"; IEEE Transactions on Parallel and Distributed Systems, 26, 1 (2014) 142–151.

[Wang et al. 2020] Wang, S., Liao, X., Wang, M., Chang, L., Yang, H., Wang, T.: "An improved SMT-based scheduling for overloaded real-time systems"; Engineering Letters, 28, 1 (2020) 112-122.

[Ying et al. 2019] Ying, K. C., Cheng, C. Y., Lin, S. W., Hung, C. Y.: "Comparative analysis of mixed integer programming formulations for single-machine and parallel-machine scheduling problems"; IEEE Access, 7 (2019) 152998-153011.

[Żurowski 2017] Żurowski, M.,: "Preemptive scheduling of jobs with a learning effect on two parallel machines"; Operations Research Proceedings, Springer (2017): 587–592.

[Żurowski and Gawiejnowicz 2019] Żurowski, M., Gawiejnowicz, S.: "Scheduling preemptable position dependent jobs on two parallel identical machines"; Computers & Industrial Engineering, 132 (2019) 373–384.